DBMS_ADDM

14.1  DBMS_ADDM Security Model 14-1
14.2  Summary of DBMS_ADDM Subprograms 14-1
  14.2.1  ANALYZE_DB Procedure 14-2
  14.2.2  ANALYZE_INST Procedure 14-3
  14.2.3  ANALYZE_PARTIAL Procedure 14-4
  14.2.4  COMPARE_CAPTURE_REPLAY_REPORT Function 14-5
  14.2.5  COMPARE_DATABASES Function 14-6
  14.2.6  COMPARE_INSTANCES Function 14-7
  14.2.7  COMPARE_REPLAY_REPLAY_REPORT Function 14-8
  14.2.8  DELETE Procedure 14-8
  14.2.9  DELETE_FINDING_DIRECTIVE Procedure 14-9
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.2.10</td>
<td>DELETE_PARAMETER_DIRECTIVE Procedure</td>
<td>14-9</td>
</tr>
<tr>
<td>14.2.11</td>
<td>DELETE_SEGMENT_DIRECTIVE Procedure</td>
<td>14-10</td>
</tr>
<tr>
<td>14.2.12</td>
<td>DELETE_SQL_DIRECTIVE Procedure</td>
<td>14-10</td>
</tr>
<tr>
<td>14.2.13</td>
<td>GET_ASH_QUERY Function</td>
<td>14-11</td>
</tr>
<tr>
<td>14.2.14</td>
<td>GET_REPORT Function</td>
<td>14-11</td>
</tr>
<tr>
<td>14.2.15</td>
<td>INSERT_FINDING_DIRECTIVE Procedure</td>
<td>14-12</td>
</tr>
<tr>
<td>14.2.16</td>
<td>INSERT_PARAMETER_DIRECTIVE Procedure</td>
<td>14-13</td>
</tr>
<tr>
<td>14.2.17</td>
<td>INSERT_SEGMENT_DIRECTIVE Procedure</td>
<td>14-13</td>
</tr>
<tr>
<td>14.2.18</td>
<td>INSERT_SQL_DIRECTIVE Procedure</td>
<td>14-15</td>
</tr>
<tr>
<td>14.2.19</td>
<td>REAL_TIME_ADDM_REPORT Function</td>
<td>14-16</td>
</tr>
</tbody>
</table>

15 DBMS_ADVANCED_REWRITE

15.1 DBMS_ADVANCED_REWRITE Security Model | 15-1 |
15.2 Summary of DBMS_ADVANCED_REWRITE Subprograms | 15-1 |
15.2.1 ALTER_REWRITE_EQUIVALENCE Procedure | 15-2 |
15.2.2 BUILD_SAFE_REWRITE_EQUIVALENCE Procedure | 15-3 |
15.2.3 DECLARE_REWRITE_EQUIVALENCE Procedures | 15-3 |
15.2.4 DROP_REWRITE_EQUIVALENCE Procedure | 15-6 |
15.2.5 VALIDATE_REWRITE_EQUIVALENCE Procedure | 15-6 |

16 DBMS_ADVISOR

16.1 DBMS_ADVISOR Deprecated Subprograms | 16-1 |
16.2 DBMS_ADVISOR Security Model | 16-2 |
16.3 Summary of DBMS_ADVISOR Subprograms | 16-2 |
16.3.1 ADD_SQLWKLD_REF Procedure | 16-5 |
16.3.2 ADD_SQLWKLD_STATEMENT Procedure | 16-6 |
16.3.3 ADD_STS_REF Procedure | 16-8 |
16.3.4 CANCEL_TASK Procedure | 16-9 |
16.3.5 COPY_SQLWKLD_TO_STS Procedure | 16-10 |
16.3.6 CREATE_FILE Procedure | 16-11 |
16.3.7 CREATE_OBJECT Procedure | 16-13 |
16.3.8 CREATE_SQLWKLD Procedure | 16-14 |
16.3.9 CREATE_TASK Procedures | 16-15 |
16.3.10 DELETE_SQLWKLD Procedure | 16-17 |
16.3.11 DELETE_SQLWKLD_REF Procedure | 16-18 |
16.3.12 DELETE_SQLWKLD_STATEMENT Procedure | 16-19 |
16.3.13 DELETE_STS_REF Procedure | 16-20 |
16.3.14 DELETE_TASK Procedure | 16-21 |
16.3.15 EXECUTE_TASK Procedure | 16-22 |
| 16.3.16 | GET_REC_ATTRIBUTES Procedure | 16-23 |
| 16.3.17 | GET_TASK_REPORT Function | 16-24 |
| 16.3.18 | GET_TASK_SCRIPT Function | 16-25 |
| 16.3.19 | IMPLEMENT_TASK Procedure | 16-27 |
| 16.3.20 | IMPORT_SQLWKLD_SCHEMA Procedure | 16-27 |
| 16.3.21 | IMPORT_SQLWKLD_SQLCACHE Procedure | 16-29 |
| 16.3.22 | IMPORT_SQLWKLD_STS Procedure | 16-30 |
| 16.3.23 | IMPORT_SQLWKLD_SUMADV Procedure | 16-32 |
| 16.3.24 | IMPORT_SQLWKLD_USER Procedure | 16-34 |
| 16.3.25 | INTERRUPT_TASK Procedure | 16-35 |
| 16.3.26 | MARK_RECOMMENDATION Procedure | 16-36 |
| 16.3.27 | QUICK_TUNE Procedure | 16-37 |
| 16.3.28 | RESET_SQLWKLD Procedure | 16-38 |
| 16.3.29 | RESET_TASK Procedure | 16-39 |
| 16.3.30 | SET_DEFAULT_SQLWKLD_PARAMETER Procedure | 16-40 |
| 16.3.31 | SET_DEFAULT_TASK_PARAMETER Procedure | 16-41 |
| 16.3.32 | SET_SQLWKLD_PARAMETER Procedure | 16-42 |
| 16.3.33 | SET_TASK_PARAMETER Procedure | 16-43 |
| 16.3.34 | TUNE_MVIEW Procedure | 16-59 |
| 16.3.35 | UPDATE_OBJECT Procedure | 16-61 |
| 16.3.36 | UPDATE_REC_ATTRIBUTES Procedure | 16-63 |
| 16.3.37 | UPDATE_SQLWKLD_ATTRIBUTES Procedure | 16-64 |
| 16.3.38 | UPDATE_SQLWKLD_STATEMENT Procedure | 16-65 |
| 16.3.39 | UPDATE_TASK_ATTRIBUTES Procedure | 16-67 |

17  DBMS_ALERT

| 17.1  | DBMS_ALERT Overview | 17-1 |
| 17.2  | DBMS_ALERT Security Model | 17-1 |
| 17.3  | DBMS_ALERT Constants | 17-1 |
| 17.4  | DBMS_ALERT Restrictions | 17-2 |
| 17.5  | DBMS_ALERT Exceptions | 17-2 |
| 17.6  | DBMS_ALERT Operational Notes | 17-2 |
| 17.7  | DBMS_ALERT Examples | 17-3 |
| 17.8  | Summary of DBMS_ALERT Subprograms | 17-4 |
| 17.8.1 | REGISTER Procedure | 17-4 |
| 17.8.2 | REMOVE Procedure | 17-5 |
| 17.8.3 | REMOVEALL Procedure | 17-5 |
| 17.8.4 | SET_DEFAULTS Procedure | 17-6 |
| 17.8.5 | SIGNAL Procedure | 17-6 |
| 17.8.6 | WAITANY Procedure | 17-7 |
### 20.4 Procedures

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.4.11</td>
<td>EXECUTE_ERROR Procedure</td>
<td>20-27</td>
</tr>
<tr>
<td>20.4.12</td>
<td>GET_ERROR_MESSAGE Function</td>
<td>20-29</td>
</tr>
<tr>
<td>20.4.13</td>
<td>HANDLE_COLLISIONS</td>
<td>20-31</td>
</tr>
<tr>
<td>20.4.14</td>
<td>REMOVE_STMT_HANDLER</td>
<td>20-32</td>
</tr>
<tr>
<td>20.4.15</td>
<td>SET_CHANGE_HANDLER Procedure</td>
<td>20-33</td>
</tr>
<tr>
<td>20.4.16</td>
<td>SET_DML_CONFLICT_HANDLER Procedure</td>
<td>20-35</td>
</tr>
<tr>
<td>20.4.17</td>
<td>SET_DML_HANDLER</td>
<td>20-41</td>
</tr>
<tr>
<td>20.4.18</td>
<td>SET_ENQUEUE_DESTINATION Procedure</td>
<td>20-46</td>
</tr>
<tr>
<td>20.4.19</td>
<td>SET_EXECUTE</td>
<td>20-48</td>
</tr>
<tr>
<td>20.4.20</td>
<td>SET_GLOBAL_INSTANTIATION_SCN Procedure</td>
<td>20-49</td>
</tr>
<tr>
<td>20.4.21</td>
<td>SET_KEY_COLUMNS Procedures</td>
<td>20-52</td>
</tr>
<tr>
<td>20.4.22</td>
<td>SET_PARAMETER Procedure</td>
<td>20-55</td>
</tr>
<tr>
<td>20.4.23</td>
<td>SET_REPERROR_HANDLER Procedure</td>
<td>20-79</td>
</tr>
<tr>
<td>20.4.24</td>
<td>SET_SCHEMA_INSTANTIATION_SCN Procedure</td>
<td>20-80</td>
</tr>
<tr>
<td>20.4.25</td>
<td>SET_TABLE_INSTANTIATION_SCN Procedure</td>
<td>20-83</td>
</tr>
<tr>
<td>20.4.26</td>
<td>SET_UPDATE_CONFLICT_HANDLER Procedure</td>
<td>20-86</td>
</tr>
<tr>
<td>20.4.27</td>
<td>SET_VALUE_DEPENDENCY Procedure</td>
<td>20-89</td>
</tr>
<tr>
<td>20.4.28</td>
<td>START_APPLY</td>
<td>20-90</td>
</tr>
<tr>
<td>20.4.29</td>
<td>STOP_APPLY</td>
<td>20-91</td>
</tr>
</tbody>
</table>

### 21 DBMS_AQ

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.1</td>
<td>DBMS_AQ Security Model</td>
<td>21-1</td>
</tr>
<tr>
<td>21.2</td>
<td>DBMS_AQ Constants</td>
<td>21-2</td>
</tr>
<tr>
<td>21.3</td>
<td>DBMS_AQ Data Types</td>
<td>21-3</td>
</tr>
<tr>
<td>21.4</td>
<td>DBMS_AQ Operational Notes</td>
<td>21-6</td>
</tr>
<tr>
<td>21.5</td>
<td>Summary of DBMS_AQ Subprograms</td>
<td>21-6</td>
</tr>
<tr>
<td>21.5.1</td>
<td>BIND_AGENT Procedure</td>
<td>21-7</td>
</tr>
<tr>
<td>21.5.2</td>
<td>DEQUEUE Procedure</td>
<td>21-7</td>
</tr>
<tr>
<td>21.5.3</td>
<td>DEQUEUE_ARRAY Function</td>
<td>21-10</td>
</tr>
<tr>
<td>21.5.4</td>
<td>ENQUEUE Procedure</td>
<td>21-11</td>
</tr>
<tr>
<td>21.5.5</td>
<td>ENQUEUE_ARRAY Function</td>
<td>21-12</td>
</tr>
<tr>
<td>21.5.6</td>
<td>LISTEN Procedures</td>
<td>21-13</td>
</tr>
<tr>
<td>21.5.7</td>
<td>POST Procedure</td>
<td>21-15</td>
</tr>
<tr>
<td>21.5.8</td>
<td>REGISTER Procedure</td>
<td>21-15</td>
</tr>
<tr>
<td>21.5.9</td>
<td>UNBIND_AGENT Procedure</td>
<td>21-16</td>
</tr>
<tr>
<td>21.5.10</td>
<td>UNREGISTER Procedure</td>
<td>21-16</td>
</tr>
</tbody>
</table>

### 22 DBMS_AQADM

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1</td>
<td>DBMS_AQADM Security Model</td>
<td>22-1</td>
</tr>
</tbody>
</table>
22.2 DBMS_AQADM Constants 22-2
22.3 DBMS_AQADM Subprogram Groups 22-3
   22.3.1 DBMS_AQADM Queue Table Subprograms 22-3
   22.3.2 DBMS_AQADM Privilege Subprograms 22-4
   22.3.3 DBMS_AQADM Queue Subprograms 22-4
   22.3.4 DBMS_AQADM Subscriber Subprograms 22-5
   22.3.5 DBMS_AQADM Propagation Subprograms 22-5
   22.3.6 DBMS_AQADM Miscellaneous Subprograms 22-5
   22.3.7 DBMS_AQADM Agent Subprograms 22-6
   22.3.8 DBMS_AQADM Alias Subprograms 22-6
22.4 Summary of DBMS_AQADM Subprograms 22-7
   22.4.1 ADD_ALIAS_TO_LDAP Procedure 22-9
   22.4.2 ADD_SUBSCRIBER Procedure 22-10
   22.4.3 ALTER_AQ_AGENT Procedure 22-11
   22.4.4 ALTER_PROPAGATION_SCHEDULE Procedure 22-12
   22.4.5 ALTER_QUEUE Procedure 22-13
   22.4.6 ALTER_QUEUE_TABLE Procedure 22-14
   22.4.7 ALTER_SHARDED_QUEUE Procedure 22-15
   22.4.8 ALTER_SUBSCRIBER Procedure 22-15
   22.4.9 CREATE_AQ_AGENT Procedure 22-16
   22.4.10 CREATE_NP_QUEUE Procedure 22-17
   22.4.11 CREATE_QUEUE Procedure 22-18
   22.4.12 CREATE_QUEUE_TABLE Procedure 22-20
   22.4.13 CREATE_SHARDED_QUEUE Procedure 22-23
   22.4.14 CREATE_EXCEPTION_QUEUE Procedure 22-24
   22.4.15 DEL_ALIAS_FROM_LDAP Procedure 22-24
   22.4.16 DISABLE_DB_ACCESS Procedure 22-25
   22.4.17 DISABLE_PROPAGATION_SCHEDULE Procedure 22-25
   22.4.18 DROP_AQ_AGENT Procedure 22-26
   22.4.19 DROP_QUEUE Procedure 22-26
   22.4.20 DROP_QUEUE_TABLE Procedure 22-27
   22.4.21 DROP_SHARDED_QUEUE Procedure 22-28
   22.4.22 ENABLE_DB_ACCESS Procedure 22-28
   22.4.23 ENABLE_JMS_TYPES Procedure 22-29
   22.4.24 ENABLE_PROPAGATION_SCHEDULE Procedure 22-29
   22.4.25 GET_MAX_STREAMS_POOL Procedure 22-30
   22.4.26 GET_MIN_STREAMS_POOL Procedure 22-30
   22.4.27 GET_QUEUE_PARAMETER Procedure 22-30
   22.4.28 GET_WATERMARK Procedure 22-31
   22.4.29 GRANT_QUEUE_PRIVILEGE Procedure 22-31
   22.4.30 GRANT_SYSTEM_PRIVILEGE Procedure 22-32
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.4.31</td>
<td>MIGRATE_QUEUE_TABLE Procedure</td>
</tr>
<tr>
<td>22.4.32</td>
<td>PURGE_QUEUE_TABLE Procedure</td>
</tr>
<tr>
<td>22.4.33</td>
<td>QUEUE_SUBSCRIBERS Function</td>
</tr>
<tr>
<td>22.4.34</td>
<td>REMOVE_SUBSCRIBER Procedure</td>
</tr>
<tr>
<td>22.4.35</td>
<td>REVOKE_QUEUE_PRIVILEGE Procedure</td>
</tr>
<tr>
<td>22.4.36</td>
<td>REVOKE_SYSTEM_PRIVILEGE Procedure</td>
</tr>
<tr>
<td>22.4.37</td>
<td>SCHEDULE_PROPAGATION Procedure</td>
</tr>
<tr>
<td>22.4.38</td>
<td>SET_WATERMARK Procedure</td>
</tr>
<tr>
<td>22.4.39</td>
<td>SET_MAX_STREAMS_POOL Procedure</td>
</tr>
<tr>
<td>22.4.40</td>
<td>SET_MIN_STREAMS_POOL Procedure</td>
</tr>
<tr>
<td>22.4.41</td>
<td>SET_QUEUE_PARAMETER Procedure</td>
</tr>
<tr>
<td>22.4.42</td>
<td>START_QUEUE Procedure</td>
</tr>
<tr>
<td>22.4.43</td>
<td>STOP_QUEUE Procedure</td>
</tr>
<tr>
<td>22.4.44</td>
<td>UNSCHEDULE_PROPAGATION Procedure</td>
</tr>
<tr>
<td>22.4.45</td>
<td>UNSET_QUEUE_PARAMETER Procedure</td>
</tr>
<tr>
<td>22.4.46</td>
<td>VERIFY_QUEUE_TYPES Procedure</td>
</tr>
</tbody>
</table>

23 DBMS_AQELM

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.1</td>
<td>DBMS_AQELM Security Model</td>
</tr>
<tr>
<td>23.2</td>
<td>Summary of DBMS_AQELM Subprograms</td>
</tr>
<tr>
<td>23.2.1</td>
<td>SET_MAILHOST Procedure</td>
</tr>
<tr>
<td>23.2.2</td>
<td>SET_MAILPORT Procedure</td>
</tr>
<tr>
<td>23.2.3</td>
<td>SET_SENDFROM Procedure</td>
</tr>
</tbody>
</table>

24 DBMS_AQIN

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.1</td>
<td>DBMS_AQIN Security Model</td>
</tr>
</tbody>
</table>

25 DBMS_ASSERT

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.1</td>
<td>DBMS_ASSERT Operational Notes</td>
</tr>
<tr>
<td>25.2</td>
<td>Summary of DBMS_ASSERT Subprograms</td>
</tr>
<tr>
<td>25.2.1</td>
<td>ENQUOTE_LITERAL Function</td>
</tr>
<tr>
<td>25.2.2</td>
<td>ENQUOTE_NAME Function</td>
</tr>
<tr>
<td>25.2.3</td>
<td>NOOP Functions</td>
</tr>
<tr>
<td>25.2.4</td>
<td>QUALIFIED_SQL_NAME Function</td>
</tr>
<tr>
<td>25.2.5</td>
<td>SCHEMA_NAME Function</td>
</tr>
<tr>
<td>25.2.6</td>
<td>SIMPLE_SQL_NAME Function</td>
</tr>
<tr>
<td>25.2.7</td>
<td>SQL_OBJECT_NAME Function</td>
</tr>
</tbody>
</table>
# DBMS_AUDIT_MGMT

26.1 DBMS_AUDIT_MGMT Overview 26-1
26.2 DBMS_AUDIT_MGMT Deprecated Subprograms 26-2
26.3 DBMS_AUDIT_MGMT Security Model 26-2
26.4 DBMS_AUDIT_MGMT Constants 26-3
26.5 DBMS_AUDIT_MGMT Views 26-4
26.6 DBMS_AUDIT_MGMT Subprogram Groups
   26.6.1 DBMS_AUDIT_MGMT Audit Trail Management Subprograms 26-5
   26.6.2 DBMS_AUDIT_MGMT Audit Trail Cleanup Subprograms 26-6
26.7 Summary of DBMS_AUDIT_MGMT Subprograms 26-7
   26.7.1 ALTER_PARTITION_INTERVAL Procedure 26-8
   26.7.2 CLEAN_AUDIT_TRAIL Procedure 26-9
   26.7.3 CLEAR_AUDIT_TRAIL_PROPERTY Procedure 26-11
   26.7.4 CLEAR_LAST_ARCHIVE_TIMESTAMP Procedure 26-12
   26.7.5 CREATE_PURGE_JOB Procedure 26-14
   26.7.6 DEINIT_CLEANUP Procedure 26-15
   26.7.7 DROP_OLD_UNIFIED_AUDIT_TABLES Procedure 26-16
   26.7.8 DROP_PURGE_JOB Procedure 26-16
   26.7.9 FLUSH_UNIFIED_AUDIT_TRAIL Procedure 26-17
   26.7.10 GET_AUDIT_COMMIT_DELAY Function 26-18
   26.7.11 GET_AUDIT_TRAIL_PROPERTY_VALUE Function 26-18
   26.7.12 GET_LAST_ARCHIVE_TIMESTAMP Function 26-19
   26.7.13 INIT_CLEANUP Procedure 26-20
   26.7.14 IS_CLEANUP_INITIALIZED Function 26-22
   26.7.15 LOAD_UNIFIED_AUDIT_FILES Procedure 26-23
   26.7.16 SET_AUDIT_TRAIL_LOCATION Procedure 26-24
   26.7.17 SET_AUDIT_TRAIL_PROPERTY Procedure 26-25
   26.7.18 SET_LAST_ARCHIVE_TIMESTAMP Procedure 26-29
   26.7.19 SET_PURGE_JOB_INTERVAL Procedure 26-30
   26.7.20 SET_PURGE_JOB_STATUS Procedure 26-31
   26.7.21 TRANSFER_UNIFIED_AUDIT_RECORDS Procedure 26-32

# DBMS_AUDIT_UTIL

27.1 DBMS_AUDIT_UTIL Overview 27-1
27.2 DBMS_AUDIT_UTIL Security Model 27-1
27.3 DBMS_AUDIT_UTIL Views 27-2
27.4 Summary of DBMS_AUDIT_UTIL Subprograms 27-2
   27.4.1 DECODE_RLS_INFO_ATRAIL_FGA Function 27-2
   27.4.2 DECODE_RLS_INFO_ATRAIL_STD Function 27-3
   27.4.3 DECODE_RLS_INFO_ATRAIL_UNI Function 27-4
32 DBMS_CAPTURE_ADM

32.1 DBMS_CAPTURE_ADM Overview
32.2 DBMS_CAPTURE_ADM Security Model
32.3 Summary of DBMS_CAPTURE_ADM Subprograms
  32.3.1 ABORT_GLOBAL_INSTANTIATION Procedure
  32.3.2 ABORT_SCHEMA_INSTANTIATION Procedure
  32.3.3 ABORT_SYNC_INSTANTIATION Procedure
  32.3.4 ABORT_TABLE_INSTANTIATION Procedure
  32.3.5 ALTER_CAPTURE Procedure
  32.3.6 ALTER_SYNC_CAPTURE Procedure
  32.3.7 BUILD Procedure
  32.3.8 CREATE_CAPTURE Procedure
  32.3.9 CREATE_SYNC_CAPTURE Procedure
  32.3.10 DROP_CAPTURE Procedure
  32.3.11 INCLUDE_EXTRA_ATTRIBUTE Procedure
  32.3.12 PREPARE_GLOBAL_INSTANTIATION Procedure
  32.3.13 PREPARE_SCHEMA_INSTANTIATION Procedure
  32.3.14 PREPARE_SYNC_INSTANTIATION Procedure
  32.3.15 PREPARE_TABLE_INSTANTIATION Procedure
  32.3.16 SET_PARAMETER Procedure
  32.3.17 START_CAPTURE Procedure
  32.3.18 STOP_CAPTURE Procedure

33 DBMS_COMPARISON

33.1 DBMS_COMPARISON Overview
33.2 DBMS_COMPARISON Security Model
33.3 DBMS_COMPARISON Constants
33.4 DBMS_COMPARISON Views
33.5 DBMS_COMPARISON Operational Notes
33.6 DBMS_COMPARISON Data Structures
  33.6.1 COMPARISON_TYPE Record Type
33.7 Summary of DBMS_COMPARISON Subprograms
  33.7.1 COMPARE Function
  33.7.2 CONVERGE Procedure
  33.7.3 CREATE_COMPARISON Procedure
  33.7.4 DROP_COMPARISON Procedure
  33.7.5 PURGE_COMPARISON Procedure
## 34 DBMS_COMPRESSION

- **34.1** DBMS_COMPRESSION Overview
- **34.2** DBMS_COMPRESSION Security Model
- **34.3** DBMS_COMPRESSION Constants
- **34.4** DBMS_COMPRESSION Data Structures  
  - **34.4.1** COMPREC Record Type
  - **34.4.2** COMPRECLIST Table Type
- **34.5** Summary of DBMS_COMPRESSION Subprograms
  - **34.5.1** GET_COMPRESSION_RATIO Procedure
  - **34.5.2** GET_COMPRESSION_TYPE Function

## 35 DBMS_CONNECTION_POOL

- **35.1** Summary of DBMS_CONNECTION_POOL Subprograms
  - **35.1.1** ALTER_PARAM Procedure
  - **35.1.2** CONFIGURE_POOL Procedure
  - **35.1.3** START_POOL Procedure
  - **35.1.4** STOP_POOL Procedure
  - **35.1.5** RESTORE_DEFAULTS Procedure

## 36 DBMS_CQ_NOTIFICATION

- **36.1** DBMS_CQ_NOTIFICATION Overview
- **36.2** DBMS_CQ_NOTIFICATION Security Model
- **36.3** DBMS_CQ_NOTIFICATION Constants
- **36.4** DBMS_CQ_NOTIFICATION Operational Notes
- **36.5** DBMS_CQ_NOTIFICATION Examples
- **36.6** DBMS_CQ_NOTIFICATION Data Structures
  - **36.6.1** CQ_NOTIFICATION$_DESCRIPTOR Object Type
  - **36.6.2** CQ_NOTIFICATION$_QUERY Object Type
  - **36.6.3** CQ_NOTIFICATION$_QUERY_ARRAY Object (Array) Type
  - **36.6.4** CQ_NOTIFICATION$_TABLE Object Type
  - **36.6.5** CQ_NOTIFICATION$_TABLE_ARRAY Object (Array) Type
  - **36.6.6** CQ_NOTIFICATION$_ROW Object Type
  - **36.6.7** CQ_NOTIFICATION$_ROW_ARRAY Object (Array) Type
  - **36.6.8** CQ_NOTIFICATION$_REG_INFO Object Type
- **36.7** Summary of DBMS_CQ_NOTIFICATION Subprograms
  - **36.7.1** CQ_NOTIFICATION_QUERYID Function
  - **36.7.2** DEREGISTER Procedure
### DBMS_CREDENTIAL

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.1</td>
<td>DBMS_CREDENTIAL Overview</td>
<td>37-1</td>
</tr>
<tr>
<td>37.2</td>
<td>DBMS_CREDENTIAL Security Model</td>
<td>37-1</td>
</tr>
<tr>
<td>37.3</td>
<td>DBMS_CREDENTIAL Operational Notes</td>
<td>37-2</td>
</tr>
<tr>
<td>37.4</td>
<td>Summary of DBMS_CREDENTIAL Subprograms</td>
<td>37-2</td>
</tr>
<tr>
<td>37.4.1</td>
<td>CREATE_CREDENTIAL Procedure</td>
<td>37-3</td>
</tr>
<tr>
<td>37.4.2</td>
<td>DISABLE_CREDENTIAL Procedure</td>
<td>37-5</td>
</tr>
<tr>
<td>37.4.3</td>
<td>DROP_CREDENTIAL Procedure</td>
<td>37-5</td>
</tr>
<tr>
<td>37.4.4</td>
<td>ENABLE_CREDENTIAL Procedure</td>
<td>37-6</td>
</tr>
<tr>
<td>37.4.5</td>
<td>UPDATE_CREDENTIAL Procedure</td>
<td>37-7</td>
</tr>
</tbody>
</table>

### DBMS_CRYPTO

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>38.1</td>
<td>DBMS_CRYPTO Overview</td>
<td>38-1</td>
</tr>
<tr>
<td>38.2</td>
<td>DBMS_CRYPTO Security Model</td>
<td>38-2</td>
</tr>
<tr>
<td>38.3</td>
<td>DBMS_CRYPTO Datatypes</td>
<td>38-2</td>
</tr>
<tr>
<td>38.4</td>
<td>DBMS_CRYPTO Algorithms</td>
<td>38-3</td>
</tr>
<tr>
<td>38.5</td>
<td>DBMS_CRYPTO Restrictions</td>
<td>38-4</td>
</tr>
<tr>
<td>38.6</td>
<td>DBMS_CRYPTO Exceptions</td>
<td>38-5</td>
</tr>
<tr>
<td>38.7</td>
<td>DBMS_CRYPTO Operational Notes</td>
<td>38-5</td>
</tr>
<tr>
<td>38.8</td>
<td>DBMS_CRYPTO Examples</td>
<td>38-7</td>
</tr>
<tr>
<td>38.9</td>
<td>Summary of DBMS_CRYPTO Subprograms</td>
<td>38-8</td>
</tr>
<tr>
<td>38.9.1</td>
<td>DECRYPT Function</td>
<td>38-8</td>
</tr>
<tr>
<td>38.9.2</td>
<td>DECRYPT Procedures</td>
<td>38-9</td>
</tr>
<tr>
<td>38.9.3</td>
<td>ENCRYPT Function</td>
<td>38-10</td>
</tr>
<tr>
<td>38.9.4</td>
<td>ENCRYPT Procedures</td>
<td>38-11</td>
</tr>
<tr>
<td>38.9.5</td>
<td>HASH Function</td>
<td>38-12</td>
</tr>
<tr>
<td>38.9.6</td>
<td>MAC Function</td>
<td>38-13</td>
</tr>
<tr>
<td>38.9.7</td>
<td>RANDOMBYTES Function</td>
<td>38-14</td>
</tr>
<tr>
<td>38.9.8</td>
<td>RANDOMINTEGER Function</td>
<td>38-14</td>
</tr>
<tr>
<td>38.9.9</td>
<td>RANDOMNUMBER Function</td>
<td>38-15</td>
</tr>
</tbody>
</table>

### DBMS_CSX_ADMIN

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>39.1</td>
<td>DBMS_CSX_ADMIN Overview</td>
<td>39-1</td>
</tr>
<tr>
<td>39.2</td>
<td>DBMS_CSX_ADMIN Security Model</td>
<td>39-1</td>
</tr>
</tbody>
</table>
39.3 DBMS_CSX_ADMIN Constants
39.4 Summary of DBMS_CSX_ADMIN Subprograms
  39.4.1 GETTOKENTABLEINFO Procedure & Function
  39.4.2 GETTOKENTABLEINFOBYTABLESPACE Procedure
  39.4.3 NAMESPACEIDTABLE Function
  39.4.4 PATHIDTABLE Function
  39.4.5 QNAMEIDTABLE Function

40 DBMS_CUBE

40.1 Using DBMS_CUBE
  40.1.1 DBMS_CUBE Security Model
40.2 Using SQL Aggregation Management
  40.2.1 Subprograms in SQL Aggregation Management
  40.2.2 Requirements for the Relational Materialized View
  40.2.3 Permissions for Managing and Querying Cube Materialized Views
  40.2.4 Example of SQL Aggregation Management
40.3 Upgrading 10g Analytic Workspaces
40.4 Summary of DBMS_CUBE Subprograms
  40.4.1 BUILD Procedure
  40.4.2 CREATE_EXPORT_OPTIONS Procedure
  40.4.3 CREATE_IMPORT_OPTIONS Procedure
  40.4.4 CREATE_MVIEW Function
  40.4.5 DERIVE_FROM_MVIEW Function
  40.4.6 DROP_MVIEW Procedure
  40.4.7 EXPORT_XML Procedure
  40.4.8 EXPORT_XML_TO_FILE Procedure
  40.4.9 IMPORT_XML Procedure
  40.4.10 INITIALIZE_CUBE_UPGRADE Procedure
  40.4.11 REFRESH_MVIEW Procedure
  40.4.12 UPGRADE_AW Procedure
  40.4.13 VALIDATE_XML Procedure

41 DBMS_CUBE_ADVISE

41.1 DBMS_CUBE_ADVISE Security Model
41.2 Summary of DBMS_CUBE_ADVISE Subprograms
  41.2.1 MV_CUBE_ADVICE Function
  41.2.2 SET_CNS_EXCEPTION_LOG Procedure
  41.2.3 TRACE Procedure
## 42 DBMS_CUBE_LOG

### 42.1 Using DBMS_CUBE_LOG

- 42.1.1 DBMS_CUBE_LOG — Logging Types
- 42.1.2 DBMS_CUBE_LOG — Logging Targets
- 42.1.3 DBMS_CUBE_LOG — Verbosity Levels
- 42.1.4 DBMS_CUBE_LOG Security Model
- 42.1.5 DBMS_CUBE_LOG — Creating Cube Logs
- 42.1.6 DBMS_CUBE_LOG — Cube Build Log
- 42.1.7 DBMS_CUBE_LOG — Cube Dimension Compile Log
- 42.1.8 DBMS_CUBE_LOG — Cube Operations Log
- 42.1.9 DBMS_CUBE_LOG — Cube Rejected Records Log

### 42.2 Summary of DBMS_CUBE_LOG Subprograms

- 42.2.1 DEFAULT_NAME Function
- 42.2.2 DISABLE Procedure
- 42.2.3 ENABLE Procedure
- 42.2.4 FLUSH Procedure
- 42.2.5 GET_LOG Procedure
- 42.2.6 GET_LOG_SPEC Function
- 42.2.7 GET_PARAMETER Function
- 42.2.8 LEVEL_HIGH Function
- 42.2.9 LEVEL_HIGHEST Function
- 42.2.10 LEVEL_LOW Function
- 42.2.11 LEVEL_LOWEST Function
- 42.2.12 LEVEL_MEDIUM Function
- 42.2.13 SET_LOG_SPEC Procedure
- 42.2.14 SET_PARAMETER Procedure
- 42.2.15 TABLE_CREATE Procedure
- 42.2.16 TARGET_FILE Function
- 42.2.17 TARGET_LOB Function
- 42.2.18 TARGET_TABLE Function
- 42.2.19 TARGET_TRACE Function
- 42.2.20 TYPE_BUILD Function
- 42.2.21 TYPE_DIMENSION_COMPILE Function
- 42.2.22 TYPE_OPERATIONS Function
- 42.2.23 TYPE_REJECTED_RECORDS Function
- 42.2.24 VERSION Function

## 43 DBMS_DATA_MINING

### 43.1 DBMS_DATA_MINING Overview

### 43.2 DBMS_DATA_MINING Security Model
43.7.13 Create Model Using Registration Information 43-97
43.7.14 DROP_ALGORITHM Procedure 43-97
43.7.15 DROP_PARTITION Procedure 43-98
43.7.16 DROP_MODEL Procedure 43-98
43.7.17 EXPORT_MODEL Procedure 43-99
43.7.18 EXPORT_SERMODEL Procedure 43-102
43.7.19 FETCH_JSON_SCHEMA Procedure 43-103
43.7.20 GET_ASSOCIATION_RULES Function 43-103
43.7.21 GET_FREQUENT_ITEMSETS Function 43-108
43.7.22 GET_MODEL_COST_MATRIX Function 43-110
43.7.23 GET_MODEL_DETAILS_AI Function 43-112
43.7.24 GET_MODEL_DETAILS_EM Function 43-113
43.7.25 GET_MODEL_DETAILS_EM_COMP Function 43-115
43.7.26 GET_MODEL_DETAILS_EM_PROJ Function 43-117
43.7.27 GET_MODEL_DETAILS_GLM Function 43-118
43.7.28 GET_MODEL_DETAILS_GLOBAL Function 43-121
43.7.29 GET_MODEL_DETAILS_KM Function 43-123
43.7.30 GET_MODELDETAILS_NB Function 43-125
43.7.31 GET_MODEL_DETAILS_NMF Function 43-127
43.7.32 GET_MODEL_DETAILS_OC Function 43-128
43.7.33 GET_MODEL_SETTINGS Function 43-130
43.7.34 GET_MODEL_SIGNATURE Function 43-131
43.7.35 GET_MODEL_DETAILS_SVD Function 43-133
43.7.36 GET_MODEL_DETAILS_SVM Function 43-135
43.7.37 GET_MODEL_DETAILS_XML Function 43-137
43.7.38 GET_MODEL_TRANSFORMATIONS Function 43-140
43.7.39 GET_TRANSFORM_LIST Procedure 43-142
43.7.40 IMPORT_MODEL Procedure 43-145
43.7.41 IMPORT_SERMODEL Procedure 43-150
43.7.42 JSON Schema for R Extensible Algorithm 43-151
43.7.43 REGISTER_ALGORITHM Procedure 43-155
43.7.44 RANK APPLY Procedure 43-156
43.7.45 REMOVE COST MATRIX Procedure 43-159
43.7.46 RENAME MODEL Procedure 43-160

44 DBMS_DATA_MINING_TRANSFORM

44.1 DBMS_DATA_MINING_TRANSFORM Overview 44-1
44.2 DBMS_DATA_MINING_TRANSFORM Operational Notes 44-4
  44.2.1 DBMS_DATA_MINING_TRANSFORM — About Transformation Lists 44-6
  44.2.2 DBMS_DATA_MINING_TRANSFORM — About Stacking and Stack Procedures 44-8
### DBMS_DATAPUMP

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>45.1</td>
<td>DBMS_DATAPUMP Overview</td>
<td>45-1</td>
</tr>
<tr>
<td>45.2</td>
<td>DBMS_DATAPUMP Security Model</td>
<td>45-2</td>
</tr>
<tr>
<td>45.3</td>
<td>DBMS_DATAPUMP Constants</td>
<td>45-2</td>
</tr>
<tr>
<td>45.4</td>
<td>DBMS_DATAPUMP Data Structures</td>
<td>45-3</td>
</tr>
<tr>
<td>45.5</td>
<td>Summary of DBMS_DATAPUMP Subprograms</td>
<td>45-12</td>
</tr>
<tr>
<td>45.5.1</td>
<td>ADD_FILE Procedure</td>
<td>45-13</td>
</tr>
<tr>
<td>45.5.2</td>
<td>ATTACH Function</td>
<td>45-17</td>
</tr>
<tr>
<td>45.5.3</td>
<td>DATA_FILTER Procedures</td>
<td>45-18</td>
</tr>
<tr>
<td>45.5.4</td>
<td>DATA_REMAP Procedure</td>
<td>45-20</td>
</tr>
<tr>
<td>45.5.5</td>
<td>DETACH Procedure</td>
<td>45-22</td>
</tr>
<tr>
<td>45.5.6</td>
<td>GET_DUMPFILE_INFO Procedure</td>
<td>45-23</td>
</tr>
<tr>
<td>45.5.7</td>
<td>GET_STATUS Procedure</td>
<td>45-26</td>
</tr>
<tr>
<td>45.5.8</td>
<td>LOG_ENTRY Procedure</td>
<td>45-29</td>
</tr>
<tr>
<td>45.5.9</td>
<td>METADATA_FILTER Procedure</td>
<td>45-29</td>
</tr>
<tr>
<td>45.5.10</td>
<td>METADATA_REMAP Procedure</td>
<td>45-33</td>
</tr>
<tr>
<td>45.5.11</td>
<td>METADATA_TRANSFORM Procedure</td>
<td>45-35</td>
</tr>
<tr>
<td>45.5.12</td>
<td>OPEN Function</td>
<td>45-39</td>
</tr>
<tr>
<td>45.5.13</td>
<td>SET_PARALLEL Procedure</td>
<td>45-42</td>
</tr>
<tr>
<td>45.5.14</td>
<td>SET_PARAMETER Procedures</td>
<td>45-43</td>
</tr>
<tr>
<td>45.5.15</td>
<td>START_JOB Procedure</td>
<td>45-52</td>
</tr>
<tr>
<td>45.5.16</td>
<td>STOP_JOB Procedure</td>
<td>45-54</td>
</tr>
<tr>
<td>45.5.17</td>
<td>WAIT_FOR_JOB Procedure</td>
<td>45-55</td>
</tr>
</tbody>
</table>

### DBMS_DB_VERSION

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>46.1</td>
<td>DBMS_DB_VERSION Overview</td>
<td>46-1</td>
</tr>
<tr>
<td>46.2</td>
<td>DBMS_DB_VERSION Constants</td>
<td>46-2</td>
</tr>
<tr>
<td>46.3</td>
<td>DBMS_DB_VERSION Examples</td>
<td>46-3</td>
</tr>
</tbody>
</table>

### DBMS_DBCOMP

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.1</td>
<td>Using DBMS_DBCOMP</td>
<td>47-1</td>
</tr>
<tr>
<td>47.2</td>
<td>DBMS_DBCOMP Security Model</td>
<td>47-2</td>
</tr>
<tr>
<td>47.3</td>
<td>Summary of the DBMS_DBCOMP Subprogram</td>
<td>47-2</td>
</tr>
</tbody>
</table>
48 DBMS_DBFS_CONTENT

48.1 DBMS_DBFS_CONTENT Overview 48-1
48.2 DBMS_DBFS_CONTENT Security Model 48-1
48.3 DBMS_DBFS_CONTENT Constants 48-2
48.4 Exceptions 48-14
48.5 DBMS_DBFS_CONTENT Operational Notes 48-15
48.6 DBMS_DBFS_CONTENT Data Structures 48-16
  48.6.1 DBMS_DBFS_CONTENT FEATURE_T Record Type 48-17
  48.6.2 DBMS_DBFS_CONTENT MOUNT_T Record Type 48-17
  48.6.3 DBMS_DBFS_CONTENT PATH_ITEM_T Record Type 48-18
  48.6.4 DBMS_DBFS_CONTENT PROP_ITEM_T Record Type 48-20
  48.6.5 PROPERTY_T Record Type 48-21
  48.6.6 DBMS_DBFS_CONTENT STORE_T Record Type 48-21
  48.6.7 DBMS_DBFS_CONTENT FEATURES_T Table Type 48-22
  48.6.8 MOUNTS_T Table Type 48-22
  48.6.9 DBMS_DBFS_CONTENT PATH_ITEMS_T Table Type 48-22
  48.6.10 DBMS_DBFS_CONTENT PROP_ITEMS_T Table Type 48-23
  48.6.11 DBMS_DBFS_CONTENT PROPERTIES_T Table Type 48-23
  48.6.12 STORES_T Table Type 48-23
48.7 Summary of DBMS_DBFS_CONTENT Subprograms 48-23
  48.7.1 CHECKACCESS Function 48-26
  48.7.2 CHECKSPI Functions and Procedures 48-26
  48.7.3 CREATEDIRECTORY Procedures 48-27
  48.7.4 CREATEFILE Procedures 48-28
  48.7.5 CREATELINK Procedures 48-29
  48.7.6 CREATEREREFERENCE Procedures 48-30
  48.7.7 DECODEFEATURES Function 48-31
  48.7.8 DELETECONTENT Procedure 48-31
  48.7.9 DELETEDIRECTORY Procedure 48-32
  48.7.10 DELETEFILE Procedure 48-32
  48.7.11 FEATURENAME Function 48-33
  48.7.12 FLUSHSTATS Function 48-33
  48.7.13 GETDEFAULTACL Procedure 48-33
  48.7.14 GETDEFAULTASOF Procedure 48-34
  48.7.15 GETDEFAULTCONTEXT Procedure 48-34
  48.7.16 GETDEFAULTOWNER Procedure 48-35
  48.7.17 GETDEFAULTPRINCIPAL Procedure 48-35
  48.7.18 GETFEATURESBYMOUNT Function 48-36
DBMS_DBFS_CONTENT_SPI

49.1 ODBMS_DBFS_CONTENT_SPI Overview
49.2 DBMS_DBFS_CONTENT_SPI Security Model
49.3 DBMS_DBFS_CONTENT_SPI Operational Notes
49.4 Summary of DBMS_DBFS_CONTENT_SPI Subprograms
  49.4.1 CHECKACCESS Function
  49.4.2 CREATEDIRECTORY Procedure
  49.4.3 CREATEFILE Procedure
  49.4.4 CREATELINK Procedure
  49.4.5 CREATEROVERRIDE Procedure
  49.4.6 DELETECONTENT Procedure
  49.4.7 DELETEDIRECTORY Procedure
  49.4.8 DELETEREFERENCE Procedure
  49.4.9 DELETEFILE Procedure
  49.4.10 GETACCESS Function
  49.4.11 GETFILEBYSTOREID Function
  49.4.12 GETFILENOWAIT Procedure
  49.4.13 GETREFERENCE Function
  49.4.14 GETVERSION Function
  49.4.15 GETSTOREID Function
  49.4.16 IMPORT Function
  49.4.17 LOCKPATH Procedure
  49.4.18 PURGEALL Procedure
  49.4.19 PURGEPATH Procedure
  49.4.20 PURGEFILE Procedure
  49.4.21 RENAMEPATH Procedure
  49.4.22 RESTOREALL Procedure
  49.4.23 RESTOREPATH Procedure
  49.4.24 RESTOREFILE Procedure
  49.4.25 SEARCH Function
  49.4.26 SETVERSION Function
  49.4.27 SETPATH Procedure
  49.4.28 SPACEUSAGE Procedure
52.4.4 CREATE_WRAPPED Procedures 52-5
52.4.5 IS_TRIGGER_FIRE_ONCE Function 52-6
52.4.6 SET_TRIGGER_FIRING_PROPERTY Procedures 52-7
52.4.7 WRAP Functions 52-9

53 DBMS_DEBUG

53.1 DBMS_DEBUG Overview 53-1
53.2 DBMS_DEBUG Constants 53-2
53.3 DBMS_DEBUG Variables 53-2
53.4 DBMS_DEBUG Exceptions 53-2
53.5 DBMS_DEBUG Operational Notes 53-4
53.6 DBMS_DEBUG Data Structures 53-11
  53.6.1 BREAKPOINT_INFO Record Type 53-12
  53.6.2 PROGRAM_INFO Record Type 53-12
  53.6.3 RUNTIME_INFO Record Type 53-13
  53.6.4 BACKTRACE_TABLE Table Type 53-14
  53.6.5 BREAKPOINT_TABLE Table Type 53-14
  53.6.6 INDEX_TABLE Table Type 53-14
  53.6.7 VC2_TABLE Table Type 53-14
53.7 Summary of DBMS_DEBUG Subprograms 53-14
  53.7.1 ATTACH_SESSION Procedure 53-16
  53.7.2 CONTINUE Function 53-16
  53.7.3 DEBUG_OFF Procedure 53-17
  53.7.4 DEBUG_ON Procedure 53-17
  53.7.5 DELETE_BREAKPOINT Function 53-18
  53.7.6 DELETE_OER_BREAKPOINT Function 53-18
  53.7.7 DETACH_SESSION Procedure 53-19
  53.7.8 DISABLE_BREAKPOINT Function 53-19
  53.7.9 ENABLE_BREAKPOINT Function 53-19
  53.7.10 EXECUTE Procedure 53-20
  53.7.11 GET_INDEXES Function 53-22
  53.7.12 GET_MORE_SOURCE Procedure 53-23
  53.7.13 GET_LINE_MAP Function 53-23
  53.7.14 GET_RUNTIME_INFO Function 53-24
  53.7.15 GET_TIMEOUT_BEHAVIOUR Function 53-25
  53.7.16 GET_VALUE Function 53-26
  53.7.17 INITIALIZE Function 53-27
  53.7.18 PING Procedure 53-29
  53.7.19 PRINT_BACKTRACE Procedure 53-29
  53.7.20 PRINT_INSTANTIATIONS Procedure 53-30
53.7.21 PROBE_VERSION Procedure 53-31
53.7.22 SELF_CHECK Procedure 53-31
53.7.23 SET_BREAKPOINT Function 53-32
53.7.24 SET_OER_BREAKPOINT Function 53-33
53.7.25 SET_TIMEOUT Function 53-33
53.7.26 SET_TIMEOUT_BEHAVIOUR Procedure 53-34
53.7.27 SET_VALUE Function 53-35
53.7.28 SHOW_BREAKPOINTS Procedures 53-36
53.7.29 SHOW_FRAME_SOURCE Procedure 53-37
53.7.30 SHOW_SOURCE Procedures 53-37
53.7.31 SYNCHRONIZE Function 53-39
53.7.32 TARGET_PROGRAM_RUNNING Procedure 53-39

54 DBMS_DEBUG_JDWP

54.1 DBMS_DEBUG_JDWP Overview 54-1
54.2 DBMS_DEBUG_JDWP Security Model 54-1
54.3 Summary DBMS_DEBUG_JDWP Subprograms 54-2
  54.3.1 CONNECT_TCP Procedure 54-2
  54.3.2 CURRENT_SESSION_ID Function 54-4
  54.3.3 CURRENT_SESSION_SERIAL Function 54-4
  54.3.4 DISCONNECT Procedure 54-4
  54.3.5 GET_NLS_PARAMETER Function 54-5
  54.3.6 PROCESS_CONNECT_STRING Procedure 54-6
  54.3.7 SET_NLS_PARAMETER Procedure 54-7

55 DBMS_DEBUG_JDWP_CUSTOM

55.1 DBMS_DEBUG_JDWP_CUSTOM Overview 55-1
55.2 DBMS_DEBUG_JDWP_CUSTOM Security Model 55-1
55.3 Summary DBMS_DEBUG_JDWP_CUSTOM Subprograms 55-1
  55.3.1 CONNECT_DEBUGGER Procedure 55-2

56 DBMS_DESCRIBE

56.1 DBMS_DESCRIBE Overview 56-1
56.2 DBMS_DESCRIBE Security Model 56-1
56.3 DBMS_DESCRIBE Types 56-1
56.4 DBMS_DESCRIBE Exceptions 56-2
56.5 DBMS_DESCRIBE Examples 56-2
56.6 Summary of DBMS_DESCRIBE Subprograms 56-5
57 DBMS_DG

57.1 Using DBMS_DG
57.2 DBMS_DG Security Model
57.3 Summary of the DBMS_DG Subprogram
  57.3.1 INITIATE_FS_FAILOVER Procedure

58 DBMS_DIMENSION

58.1 DBMS_DIMENSION Security Model
58.2 Summary of DBMS_DIMENSION Subprograms
  58.2.1 DESCRIBE_DIMENSION Procedure
  58.2.2 VALIDATE_DIMENSION Procedure

59 DBMS_DISTRIBUTED_TRUST_ADMIN

59.1 DBMS_DISTRIBUTED_TRUST_ADMIN Overview
59.2 DBMS_DISTRIBUTED_TRUST_ADMIN Security Model
59.3 DBMS_DISTRIBUTED_TRUST_ADMIN Examples
59.4 Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms
  59.4.1 ALLOW_ALL Procedure
  59.4.2 ALLOW_SERVER Procedure
  59.4.3 DENY_ALL Procedure
  59.4.4 DENY_SERVER Procedure

60 DBMS_DNFS

60.1 DBMS_DNFS Security Model
60.2 Summary of DBMS_DNFS Subprograms
  60.2.1 CLONEDB_RENAMEFILE Procedure
  60.2.2 RESTORE_DATAFILE_PERMISSIONS Procedure
  60.2.3 UNMOUNTVOLUME Procedure

61 DBMS_DST

61.1 DBMS_DST Overview
61.2 DBMS_DST Security Model
61.3 DBMS_DST Views
61.4 Summary of DBMS_DST Subprograms
  61.4.1 BEGIN_PREPARE Procedure
62.1 DBMS_EDITIONS_UTILITIES Overview 62-1
62.2 DBMS_EDITIONS_UTILITIES Security Model 62-1
62.3 DBMS_EDITIONS_UTILITIES Exceptions 62-1
62.4 Summary of DBMS_EDITIONS_UTILITIES Subprograms 62-1
   62.4.1 CLEAN_UNUSABLE_EDITIONS Procedure 62-2
   62.4.2 SET_EDITIONING_VIEWS_READ_ONLY Procedure 62-2
   62.4.3 SET_NULL_COLUMN_VALUES_TO_EXPR Procedure 62-3

63.1 DBMS_EPG Overview 63-1
63.2 DBMS_EPG Security Model 63-1
63.3 DBMS_EPG Exceptions 63-1
63.4 DBMS_EPG Data Structures 63-2
63.5 DBMS_EPG Subprogram Groups 63-2
   63.5.1 DBMS_EPG Configuration Subprograms 63-2
   63.5.2 DBMS_EPG Authorization Subprograms 63-3
63.6 Summary of DBMS_EPG Subprograms 63-3
   63.6.1 AUTHORIZE_DAD Procedure 63-4
   63.6.2 CREATE_DAD Procedure 63-5
   63.6.3 DEAUTHORIZE_DAD Procedure 63-5
   63.6.4 DELETE_DAD_ATTRIBUTE Procedure 63-6
   63.6.5 DELETE_GLOBAL_ATTRIBUTE Procedure 63-7
   63.6.6 DROP_DAD Procedure 63-7
   63.6.7 GET_ALL_DAD_ATTRIBUTES Procedure 63-8
   63.6.8 GET_ALL_DAD_MAPPINGS Procedure 63-8
   63.6.9 GET_ALL_GLOBAL_ATTRIBUTES Procedure 63-9
   63.6.10 GET_DAD_ATTRIBUTE Function 63-10
63.6.11 GET_DAD_LIST Procedure 63-10
63.6.12 GET_GLOBAL_ATTRIBUTE Function 63-11
63.6.13 MAP_DAD Procedure 63-11
63.6.14 SET_DAD_ATTRIBUTE Procedure 63-12
63.6.15 SET_GLOBAL_ATTRIBUTE Procedure 63-14
63.6.16 UNMAP_DAD Procedure 63-15

64 DBMS_ERRLOG

64.1 DBMS_DIMENSION Security Model 64-1
64.2 Summary of DBMS_ERRLOG Subprograms 64-1
  64.2.1 CREATE_ERROR_LOG Procedure 64-1

65 DBMS_FGA

65.1 DBMS_FGA Security Model 65-1
65.2 DBMS_FGA Operational Notes 65-1
65.3 Summary of DBMS_FGA Subprograms 65-1
  65.3.1 ADD_POLICY Procedure 65-2
  65.3.2 DISABLE_POLICY Procedure 65-6
  65.3.3 DROP_POLICY Procedure 65-7
  65.3.4 ENABLE_POLICY Procedure 65-8

66 DBMS_FILE_GROUP

66.1 DBMS_FILE_GROUP Overview 66-1
66.2 DBMS_FILE_GROUP Security Model 66-2
66.3 DBMS_FILE_GROUP Constants 66-2
66.4 DBMS_FILE_GROUP Examples 66-3
66.5 Summary of DBMS_FILE_GROUP Subprograms 66-3
  66.5.1 ADD_FILE Procedure 66-4
  66.5.2 ALTER_FILE Procedure 66-6
  66.5.3 ALTER_FILE_GROUP Procedure 66-8
  66.5.4 ALTER_VERSION Procedure 66-10
  66.5.5 CREATE_FILE_GROUP Procedure 66-11
  66.5.6 CREATE_VERSION Procedure 66-12
  66.5.7 DROP_FILE_GROUP Procedure 66-14
  66.5.8 DROP_VERSION Procedure 66-14
  66.5.9 GRANT_OBJECT_PRIVILEGE Procedure 66-15
  66.5.10 GRANT_SYSTEM_PRIVILEGE Procedure 66-16
  66.5.11 PURGE_FILE_GROUP Procedure 66-17
  66.5.12 REMOVE_FILE Procedure 66-17
67 DBMS_FILE_TRANSFER

67.1 DBMS_FILE_TRANSFER Overview
67.2 DBMS_FILE_TRANSFER Security Model
67.3 DBMS_FILE_TRANSFER Operational Notes
67.4 Summary of DBMS_FILE_TRANSFER Subprograms
  67.4.1 COPY_FILE Procedure
  67.4.2 GET_FILE Procedure
  67.4.3 PUT_FILE Procedure

68 DBMS_FLASHBACK

68.1 DBMS_FLASHBACK Overview
68.2 DBMS_FLASHBACK Security Model
68.3 DBMS_FLASHBACK Types
68.4 DBMS_FLASHBACK Exceptions
68.5 DBMS_FLASHBACK Operational Notes
68.6 DBMS_FLASHBACK Examples
68.7 Summary of DBMS_FLASHBACK Subprograms
  68.7.1 DISABLE Procedure
  68.7.2 ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure
  68.7.3 ENABLE_AT_TIME Procedure
  68.7.4 GET_SYSTEM_CHANGE_NUMBER Function
  68.7.5 TRANSACTION_BACKOUT Procedures

69 DBMS_FLASHBACK_ARCHIVE

69.1 DBMS_FLASHBACK_ARCHIVE Overview
69.2 DBMS_FLASHBACK_ARCHIVE Security Model
69.3 DBMS_FLASHBACK_ARCHIVE Constants
69.4 Summary of DBMS_FLASHBACK_ARCHIVE Subprograms
  69.4.1 ADD_TABLE_TO_APPLICATION Procedure
  69.4.2 CREATE_TEMP_HISTORY_TABLE Procedure
  69.4.3 DISABLE_APPLICATION Procedure
  69.4.4 DISABLE_ASOF_VALID_TIME Procedure
  69.4.5 DISASSOCIATE_FBA Procedure
  69.4.6 DROP_APPLICATION Procedure
  69.4.7 ENABLE_APPLICATION Procedure
  69.4.8 ENABLE_AT_VALID_TIME Procedure
69.4.9  EXTEND_MAPPINGS Procedure 69-7
69.4.10  GET_SYS_CONTEXT Function 69-7
69.4.11  IMPORT_HISTORY Procedure 69-8
69.4.12  LOCK_DOWN_APPLICATION Procedure 69-9
69.4.13  PURGE_CONTEXT Procedure 69-9
69.4.14  REASSOCIATE_FBA Procedure 69-9
69.4.15  REGISTER_APPLICATION Procedure 69-10
69.4.16  REMOVE_TABLE_FROM_APPLICATION Procedure 69-11
69.4.17  SET_CONTEXT_LEVEL Procedure 69-11

70  DBMS_FREQUENT_ITEMSET

70.1  Summary of DBMS_FREQUENT_ITEMSET Subprograms 70-1
   70.1.1  FI_HORIZONTAL Function 70-1
   70.1.2  FI_TRANSACTIONAL Function 70-3

71  DBMS_FS

71.1  DBMS_FS Overview 71-1
71.2  DBMS_FS Security Model 71-1
71.3  Summary of DBMS_FS Subprograms 71-1
   71.3.1  DESTROY_ORACLE_FS Procedure 71-2
   71.3.2  MAKE_ORACLE_FS Procedure 71-3
   71.3.3  MOUNT_ORACLE_FS Procedure 71-4
   71.3.4  UNMOUNT_ORACLE_FS Procedure 71-6

72  DBMS_GOLDENGATEADM

72.1  Using DBMS_GOLDENGATEADM 72-1
   72.1.1  DBMS_GOLDENGATEADM Overview 72-1
   72.1.2  DBMS_GOLDENGATEADM Security Model 72-2
72.2  Summary of DBMS_GOLDENGATEADM Subprograms 72-2
   72.2.1  ADD_AUTO_CDR Procedure 72-3
   72.2.2  ADD_AUTO_CDR_COLUMN_GROUP Procedure 72-4
   72.2.3  ADD_AUTO_CDR_DELTA_RES Procedure 72-5
   72.2.4  ALTER_AUTO_CDR Procedure 72-6
   72.2.5  ALTER_AUTO_CDR_COLUMN_GROUP Procedure 72-7
   72.2.6  DELETE_PROCREP_EXCLUSION_OBJ Procedure 72-8
   72.2.7  GG_PROCEDURE_REPLICATION_ON Function 72-9
   72.2.8  INSERT_PROCREP_EXCLUSION_OBJ Procedure 72-9
   72.2.9  PURGE_TOMBSTONES Procedure 72-10
   72.2.10 REMOVE_AUTO_CDR Procedure 72-10
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>72.2.11</td>
<td>REMOVE_AUTO_CDR_COLUMN_GROUP Procedure</td>
<td>72-10</td>
</tr>
<tr>
<td>72.2.12</td>
<td>REMOVE_AUTO_CDR_DELTA_RES Procedure</td>
<td>72-11</td>
</tr>
<tr>
<td>73</td>
<td>DBMS_GOLDENGATE_AUTH</td>
<td></td>
</tr>
<tr>
<td>73.1</td>
<td>DBMS_GOLDENGATE_AUTH Overview</td>
<td>73-1</td>
</tr>
<tr>
<td>73.2</td>
<td>DBMS_GOLDENGATE_AUTH Security Model</td>
<td>73-1</td>
</tr>
<tr>
<td>73.3</td>
<td>Summary of DBMS_GOLDENGATE_AUTH Subprograms</td>
<td>73-2</td>
</tr>
<tr>
<td>73.3.1</td>
<td>GRANT_ADMIN_PRIVILEGE Procedure</td>
<td>73-2</td>
</tr>
<tr>
<td>73.3.2</td>
<td>REVOKE_ADMIN_PRIVILEGE Procedure</td>
<td>73-6</td>
</tr>
<tr>
<td>74</td>
<td>DBMS_HADOOP</td>
<td></td>
</tr>
<tr>
<td>74.1</td>
<td>DBMS_HADOOP Overview</td>
<td>74-1</td>
</tr>
<tr>
<td>74.2</td>
<td>DBMS_HADOOP Security Model</td>
<td>74-1</td>
</tr>
<tr>
<td>74.3</td>
<td>Summary of DBMS_HADOOP Subprograms</td>
<td>74-1</td>
</tr>
<tr>
<td>74.3.1</td>
<td>CREATE_EXTDDL_FOR_HIVE Procedure</td>
<td>74-2</td>
</tr>
<tr>
<td>74.3.2</td>
<td>SYNCHRONIZE_PARTITION FOR_HIVE Procedure</td>
<td>74-2</td>
</tr>
<tr>
<td>75</td>
<td>DBMS_HANG_MANAGER</td>
<td></td>
</tr>
<tr>
<td>75.1</td>
<td>DBMS_HANG_MANAGER Overview</td>
<td>75-1</td>
</tr>
<tr>
<td>75.2</td>
<td>DBMS_HANG_MANAGER Security Model</td>
<td>75-1</td>
</tr>
<tr>
<td>75.3</td>
<td>DBMS_HANG_MANAGER Constants</td>
<td>75-1</td>
</tr>
<tr>
<td>75.4</td>
<td>DBMS_HANG_MANAGER Exceptions</td>
<td>75-3</td>
</tr>
<tr>
<td>75.5</td>
<td>Summary of DBMS_HANG_MANAGER Subprograms</td>
<td>75-3</td>
</tr>
<tr>
<td>75.5.1</td>
<td>SET Procedure</td>
<td>75-3</td>
</tr>
<tr>
<td>76</td>
<td>DBMS_HEAT_MAP</td>
<td></td>
</tr>
<tr>
<td>76.1</td>
<td>DBMS_HEAT_MAP Overview</td>
<td>76-1</td>
</tr>
<tr>
<td>76.2</td>
<td>DBMS_HEAT_MAP Security Model</td>
<td>76-1</td>
</tr>
<tr>
<td>76.3</td>
<td>Summary of DBMS_HEAT_MAP Subprograms</td>
<td>76-1</td>
</tr>
<tr>
<td>76.3.1</td>
<td>BLOCK_HEAT_MAP Function</td>
<td>76-2</td>
</tr>
<tr>
<td>76.3.2</td>
<td>EXTENT_HEAT_MAP Function</td>
<td>76-3</td>
</tr>
<tr>
<td>76.3.3</td>
<td>OBJECT_HEAT_MAP Function</td>
<td>76-4</td>
</tr>
<tr>
<td>76.3.4</td>
<td>SEGMENT_HEAT_MAP Procedure</td>
<td>76-5</td>
</tr>
<tr>
<td>76.3.5</td>
<td>TABLESPACE_HEAT_MAP Function</td>
<td>76-6</td>
</tr>
</tbody>
</table>
81.3.4  BIND_OUT_VARIABLE_RAW Procedure 81-5
81.3.5  BIND_VARIABLE Procedure 81-6
81.3.6  BIND_VARIABLE_RAW Procedure 81-7
81.3.7  CLOSE_CURSOR Procedure 81-8
81.3.8  EXECUTE_IMMEDIATE Procedure 81-9
81.3.9  EXECUTE_NON_QUERY Function 81-10
81.3.10 FETCH_ROW Function 81-10
81.3.11 GET_VALUE Procedure 81-11
81.3.12 GET_VALUE_RAW Procedure 81-12
81.3.13 OPEN_CURSOR Function 81-13
81.3.14 PARSE Procedure 81-14

82  DBMS_ILM

82.1  DBMS_ILM Overview 82-1
82.2  DBMS_ILM Security Model 82-2
82.3  DBMS_ILM Constants 82-2
82.4  DBMS_ILM Exceptions 82-2
82.5  Summary of DBMS_ILM Subprograms 82-3
   82.5.1  ADD_TO_IlM Procedure 82-3
   82.5.2  ARCHIVESTATENAME Function 82-4
   82.5.3  EXECUTE_ILM Procedure 82-4
   82.5.4  EXECUTE_ILM_TASK Procedure 82-5
   82.5.5  PREVIEW_ILM Procedure 82-6
   82.5.6  REMOVE_FROM_ILM Procedure 82-6
   82.5.7  STOP_ILM Procedure 82-7

83  DBMS_ILM_ADMIN

83.1  DBMS_ILM_ADMIN Overview 83-1
83.2  DBMS_ILM_ADMIN Security Model 83-1
83.3  DBMS_ILM_ADMIN Constants 83-1
83.4  Summary of DBMS_ILM_ADMIN Subprograms 83-5
   83.4.1  CLEAR_HEAT_MAP_ALL Procedure 83-5
   83.4.2  CLEAR_HEAT_MAP_TABLE Procedure 83-5
   83.4.3  CUSTOMIZE_ILM Procedure 83-6
   83.4.4  DISABLE_ILM Procedure 83-6
   83.4.5  ENABLE_ILM Procedure 83-6
   83.4.6  SET_HEAT_MAP_ALL Procedure 83-7
   83.4.7  SET_HEAT_MAP_START Procedure 83-7
84 DBMS_INMEMORY

84.1 DBMS_INMEMORY Overview 84-1
84.2 DBMS_INMEMORY Security Model 84-2
84.3 Summary of DBMS_INMEMORY Subprograms 84-2
  84.3.1 IME_DROP_EXPRESSIONS Procedure 84-2
  84.3.2 POPULATE Procedure 84-3
  84.3.3 REPOPULATE Procedure 84-4
  84.3.4 SEGMENT_DEALLOCATE_VERSIONS Procedure 84-4

85 DBMS_INMEMORY_ADMIN

85.1 DBMS_INMEMORY_ADMIN Overview 85-1
85.2 DBMS_INMEMORY_ADMIN Security Model 85-2
85.3 DBMS_INMEMORY_ADMIN Operational Notes 85-3
85.4 Summary of DBMS_INMEMORY_ADMIN Subprograms 85-3
  85.4.1 AIM_GET_PARAMETER Procedure 85-4
  85.4.2 AIM_SET_PARAMETER Procedure 85-5
  85.4.3 FASTSTART_DISABLE Procedure 85-6
  85.4.4 FASTSTART_ENABLE Procedure 85-7
  85.4.5 FASTSTART_MIGRATE_STORAGE Procedure 85-9
  85.4.6 GET_FASTSTART_TABLESPACE Function 85-10
  85.4.7 IME_CAPTURE_EXPRESSIONS Procedure 85-11
  85.4.8 IME_CLOSE_CAPTURE_WINDOW Procedure 85-13
  85.4.9 IME_DROP_ALL_EXPRESSIONS Procedure 85-14
  85.4.10 IME_GET_CAPTURE_STATE Procedure 85-14
  85.4.11 IME_OPEN_CAPTURE_WINDOW Procedure 85-15
  85.4.12 IME_POPULATE_EXPRESSIONS Procedure 85-16

86 DBMS_IOT

86.1 Summary of DBMS_IOT Subprograms 86-1
  86.1.1 BUILD_CHAIN_ROWS_TABLE Procedure 86-1
  86.1.2 BUILD_EXCEPTIONS_TABLE Procedure 86-2

87 DBMS_JAVA
DBMS_JOB

88.1 DBMS_JOB Security Model 88-1
88.2 DBMS_JOB Operational Notes 88-1
88.3 Summary of DBMS_JOB Subprograms 88-3
  88.3.1 BROKEN Procedure 88-3
  88.3.2 CHANGE Procedure 88-4
  88.3.3 INSTANCE Procedure 88-5
  88.3.4 INTERVAL Procedure 88-6
  88.3.5 NEXT_DATE Procedure 88-7
  88.3.6 REMOVE Procedure 88-7
  88.3.7 RUN Procedure 88-8
  88.3.8 SUBMIT Procedure 88-8
  88.3.9 USER_EXPORT Procedures 88-10
  88.3.10 WHAT Procedure 88-10

DBMS_JSON

89.1 DBMS_JSON Overview 89-1
89.2 DBMS_JSON Security Model 89-1
89.3 DBMS_JSON Constants 89-1
89.4 Summary of DBMS_JSON Subprograms 89-2
  89.4.1 ADDVIRTUAL_COLUMNS Procedure 89-2
  89.4.2 CREATE_VIEW Procedure 89-4
  89.4.3 CREATE_VIEW_ON_PATH Procedure 89-4
  89.4.4 DROPVIRTUAL_COLUMNS Procedure 89-5
  89.4.5 GET_INDEX_DATAGUIDE Function 89-6
  89.4.6 RENAME_COLUMN Procedure 89-7

DBMS_LDAP

DBMS_LDAP_UTL

DBMS_LIBCACHE

92.1 DBMS_LIBCACHE Overview 92-1
92.2 DBMS_LIBCACHE Security Model 92-1
92.3 Summary of DBMS_LIBCACHE Subprograms 92-1
### DBMS_LOB

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>93.1</td>
<td>DBMS_LOB Overview</td>
<td>93-1</td>
</tr>
<tr>
<td>93.2</td>
<td>DBMS_LOB Deprecated Subprograms</td>
<td>93-1</td>
</tr>
<tr>
<td>93.3</td>
<td>DBMS_LOB Security Model</td>
<td>93-2</td>
</tr>
<tr>
<td>93.4</td>
<td>DBMS_LOB Constants</td>
<td>93-2</td>
</tr>
<tr>
<td>93.5</td>
<td>DBMS_LOB Datatypes</td>
<td>93-4</td>
</tr>
<tr>
<td>93.6</td>
<td>DBMS_LOB Operational Notes</td>
<td>93-5</td>
</tr>
<tr>
<td>93.7</td>
<td>DBMS_LOB Rules and Limits</td>
<td>93-8</td>
</tr>
<tr>
<td>93.8</td>
<td>DBMS_LOB Exceptions</td>
<td>93-11</td>
</tr>
<tr>
<td>93.9</td>
<td>Summary of DBMS_LOB Subprograms</td>
<td>93-12</td>
</tr>
<tr>
<td>93.9.1</td>
<td>APPEND Procedures</td>
<td>93-14</td>
</tr>
<tr>
<td>93.9.2</td>
<td>CLOB2FILE Procedure</td>
<td>93-16</td>
</tr>
<tr>
<td>93.9.3</td>
<td>CLOSE Procedure</td>
<td>93-16</td>
</tr>
<tr>
<td>93.9.4</td>
<td>COMPARE Functions</td>
<td>93-17</td>
</tr>
<tr>
<td>93.9.5</td>
<td>CONVERTTOBLOB Procedure</td>
<td>93-19</td>
</tr>
<tr>
<td>93.9.6</td>
<td>CONVERTTOCLOB Procedure</td>
<td>93-22</td>
</tr>
<tr>
<td>93.9.7</td>
<td>COPY Procedures</td>
<td>93-24</td>
</tr>
<tr>
<td>93.9.8</td>
<td>COPY_DBFS_LINK Procedures</td>
<td>93-26</td>
</tr>
<tr>
<td>93.9.9</td>
<td>COPY_FROM_DBFS_LINK</td>
<td>93-27</td>
</tr>
<tr>
<td>93.9.10</td>
<td>CREATETEMPORARY Procedures</td>
<td>93-28</td>
</tr>
<tr>
<td>93.9.11</td>
<td>DBFS_LINK_GENERATE_PATH Functions</td>
<td>93-28</td>
</tr>
<tr>
<td>93.9.12</td>
<td>ERASE Procedures</td>
<td>93-29</td>
</tr>
<tr>
<td>93.9.13</td>
<td>FILECLOSE Procedure</td>
<td>93-31</td>
</tr>
<tr>
<td>93.9.14</td>
<td>FILECLOSEALL Procedure</td>
<td>93-32</td>
</tr>
<tr>
<td>93.9.15</td>
<td>FILEEXISTS Function</td>
<td>93-32</td>
</tr>
<tr>
<td>93.9.16</td>
<td>FILEGETNAME Procedure</td>
<td>93-33</td>
</tr>
<tr>
<td>93.9.17</td>
<td>FILEISOPEN Function</td>
<td>93-34</td>
</tr>
<tr>
<td>93.9.18</td>
<td>FILEOPEN Procedure</td>
<td>93-35</td>
</tr>
<tr>
<td>93.9.19</td>
<td>FRAGMENT_DELETE Procedure</td>
<td>93-36</td>
</tr>
<tr>
<td>93.9.20</td>
<td>FRAGMENT_INSERT Procedures</td>
<td>93-37</td>
</tr>
<tr>
<td>93.9.21</td>
<td>FRAGMENT_MOVE Procedure</td>
<td>93-38</td>
</tr>
<tr>
<td>93.9.22</td>
<td>FRAGMENT_REPLACE Procedures</td>
<td>93-39</td>
</tr>
<tr>
<td>93.9.23</td>
<td>FREETEMPORARY Procedures</td>
<td>93-40</td>
</tr>
<tr>
<td>93.9.24</td>
<td>GET_DBFS_LINK Functions</td>
<td>93-41</td>
</tr>
<tr>
<td>93.9.25</td>
<td>GET_DBFS_LINK_STATE Procedures</td>
<td>93-42</td>
</tr>
<tr>
<td>93.9.26</td>
<td>GETCONTENTTYPE Functions</td>
<td>93-43</td>
</tr>
<tr>
<td>93.9.27</td>
<td>GET_STORAGE_LIMIT Function</td>
<td>93-43</td>
</tr>
<tr>
<td>93.9.28</td>
<td>GETCHUNKSIZE Functions</td>
<td>93-44</td>
</tr>
<tr>
<td>Page</td>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>93.9.29</td>
<td>GETLENGTH Functions</td>
<td>93-45</td>
</tr>
<tr>
<td>93.9.30</td>
<td>GETOPTIONS Functions</td>
<td>93-46</td>
</tr>
<tr>
<td>93.9.31</td>
<td>INSTR Functions</td>
<td>93-47</td>
</tr>
<tr>
<td>93.9.32</td>
<td>ISOPEN Functions</td>
<td>93-49</td>
</tr>
<tr>
<td>93.9.33</td>
<td>ISREMOTE Function</td>
<td>93-50</td>
</tr>
<tr>
<td>93.9.34</td>
<td>ISSECUREFILE Function</td>
<td>93-51</td>
</tr>
<tr>
<td>93.9.35</td>
<td>ITEMPORARY Functions</td>
<td>93-52</td>
</tr>
<tr>
<td>93.9.36</td>
<td>LOADBLOBFROMFILE Procedure</td>
<td>93-52</td>
</tr>
<tr>
<td>93.9.37</td>
<td>LOADCLOBFROMFILE Procedure</td>
<td>93-54</td>
</tr>
<tr>
<td>93.9.38</td>
<td>LOADFROMFILE Procedure</td>
<td>93-58</td>
</tr>
<tr>
<td>93.9.39</td>
<td>MOVE_TO,DBFS_LINK Procedures</td>
<td>93-60</td>
</tr>
<tr>
<td>93.9.40</td>
<td>OPEN Procedures</td>
<td>93-61</td>
</tr>
<tr>
<td>93.9.41</td>
<td>READ Procedures</td>
<td>93-62</td>
</tr>
<tr>
<td>93.9.42</td>
<td>SET,DBFS_LINK Procedures</td>
<td>93-64</td>
</tr>
<tr>
<td>93.9.43</td>
<td>SETCONTENTTYPE Procedure</td>
<td>93-64</td>
</tr>
<tr>
<td>93.9.44</td>
<td>SETOPTIONS Procedures</td>
<td>93-65</td>
</tr>
<tr>
<td>93.9.45</td>
<td>SUBSTR Functions</td>
<td>93-66</td>
</tr>
<tr>
<td>93.9.46</td>
<td>TRIM Procedures</td>
<td>93-68</td>
</tr>
<tr>
<td>93.9.47</td>
<td>WRITE Procedures</td>
<td>93-70</td>
</tr>
<tr>
<td>93.9.48</td>
<td>WRITEAPPEND Procedures</td>
<td>93-72</td>
</tr>
</tbody>
</table>

### DBMS_LOCK

| 94 | DBMS_LOCK Overview | 94-1 |
| 94.1 | DBMS_LOCK Security Model | 94-1 |
| 94.2 | DBMS_LOCK Constants | 94-2 |
| 94.3 | DBMS_LOCK, Rules and Limits | 94-2 |
| 94.4 | DBMS_LOCK Operational Notes | 94-3 |
| 94.5 | Summary of DBMS_LOCK Subprograms | 94-3 |
| 94.6.1 | ALLOCATE_UNIQUE Procedure | 94-3 |
| 94.6.2 | ALLOCATE_UNIQUE_AUTONOMOUS Procedure | 94-5 |
| 94.6.3 | CONVERT Function | 94-6 |
| 94.6.4 | RELEASE Function | 94-7 |
| 94.6.5 | REQUEST Function | 94-8 |

### DBMS_LOGMNR

| 95 | DBMS_LOGMNR Overview | 95-1 |
| 95.1 | DBMS_LOGMNR Security Model | 95-2 |
| 95.2 | DBMS_LOGMNR Constants | 95-2 |
| 95.3 | DBMS_LOGMNR Views | 95-4 |
95.5  DBMS_LOGMNR Operational Notes 95-4
95.6  Summary of DBMS_LOGMNR Subprograms 95-5
  95.6.1  ADD_LOGFILE Procedure 95-5
  95.6.2  COLUMN_PRESENT Function 95-7
  95.6.3  END_LOGMNR Procedure 95-8
  95.6.4  MINE_VALUE Function 95-9
  95.6.5  REMOVE_LOGFILE Procedure 95-10
  95.6.6  START_LOGMNR Procedure 95-11

96  DBMS_LOGMNR_D
  96.1  DBMS_LOGMNR_D Overview 96-1
  96.2  DBMS_LOGMNR_D Security Model 96-2
  96.3  Summary of DBMS_LOGMNR_D Subprograms 96-2
    96.3.1  BUILD Procedure 96-2
    96.3.2  SET_TABLESPACE Procedure 96-5

97  DBMS_LOGSTDBY
  97.1  DBMS_LOGSTDBY Overview 97-1
  97.2  DBMS_LOGSTDBY Security Model 97-1
  97.3  DBMS_LOGSTDBY Constants 97-2
  97.4  Summary of DBMS_LOGSTDBY Subprograms 97-3
    97.4.1  APPLY_SET Procedure 97-5
    97.4.2  APPLY_UNSET Procedure 97-8
    97.4.3  BUILD Procedure 97-8
    97.4.4  EDS_ADD_TABLE 97-10
    97.4.5  EDS_EVOLVE_AUTOMATIC 97-11
    97.4.6  EDS_EVOLVE_MANUAL 97-13
    97.4.7  EDS_REMOVE_TABLE 97-14
    97.4.8  INSTANTIATE_TABLE Procedure 97-15
    97.4.9  IS_APPLY_SERVER Function 97-16
    97.4.10 MAP_PRIMARY_SCN Function 97-17
    97.4.11 PREPARE_FOR_NEW_PRIMARY Procedure 97-17
    97.4.12 PURGE_SESSION Procedure 97-18
    97.4.13 REBUILD Procedure 97-20
    97.4.14 SET_TABLESPACE Procedure 97-20
    97.4.15 SKIP Procedure 97-21
    97.4.16 SKIP_ERROR Procedure 97-30
    97.4.17 SKIP_TRANSACTION Procedure 97-34
    97.4.18 UNSKIP Procedure 97-35
97.4.19 UNSKIP_ERROR Procedure 97-36
97.4.20 UNSKIP_TRANSACTION Procedure 97-38

98 DBMS_LOGSTDBY_CONTEXT

98.1 DBMS_LOGSTDBY_CONTEXT Overview 98-1
98.2 DBMS_LOGSTDBY_CONTEXT Security Model 98-2
98.3 Summary of DBMS_LOGSTDBY_CONTEXT Subprograms 98-2
  98.3.1 CLEAR_ALL_CONTEXT Procedure 98-3
  98.3.2 CLEARCONTEXT Procedure 98-3
  98.3.3 GET_CONTEXT Procedure 98-3
  98.3.4 SET_CONTEXT Procedure 98-4

99 DBMS_MEMOPTIMIZE

99.1 DBMS_MEMOPTIMIZE Overview 99-1
99.2 Summary of DBMS_MEMOPTIMIZE Subprograms 99-1
  99.2.1 DROP_OBJECT Procedure 99-1
  99.2.2 POPULATE Procedure 99-2

100 DBMS_METADATA

100.1 DBMS_METADATA Overview 100-1
100.2 DBMS_METADATA Security Model 100-2
100.3 Rules and Limits 100-3
100.4 DBMS_METADATA Data Structures - Object and Table Types 100-3
100.5 DBMS_METADATA Subprogram Groupings 100-4
  100.5.1 DBMS_METADATA Subprograms for Retrieving Multiple Objects From the Database 100-5
  100.5.2 DBMS_METADATA Subprograms for Submitting XML to the Database 100-5
100.6 Summary of All DBMS_METADATA Subprograms 100-6
  100.6.1 ADD_TRANSFORM Function 100-6
  100.6.2 CLOSE Procedure 100-10
  100.6.3 CONVERT Functions and Procedures 100-11
  100.6.4 FETCH_xxx Functions and Procedures 100-13
  100.6.5 GET_xxx Functions 100-16
  100.6.6 GET_QUERY Function 100-19
  100.6.7 OPEN Function 100-20
  100.6.8 OPENW Function 100-27
  100.6.9 PUT Function 100-28
  100.6.10 SET_COUNT Procedure 100-29
  100.6.11 SET_FILTER Procedure 100-30
101 DBMS_METADATA_DIFF

101.1 DBMS_METADATA_DIFF Overview
101.2 DBMS_METADATA_DIFF Security Model
101.3 DBMS_METADATA_DIFF — Browsing APIs for Fetching and Comparing Objects
101.4 Summary of DBMS_METADATA_DIFF Subprograms
   101.4.1 OPENC Function
   101.4.2 ADD_DOCUMENT Procedure
   101.4.3 FETCH_CLOB Functions and Procedures
   101.4.4 CLOSE Procedure

102 DBMS_MGD_ID_UTL

102.1 DBMS_MGD_ID_UTL Security Model
102.2 DBMS_MGD_ID_UTL Constants
102.3 DBMS_MGD_ID_UTL Exceptions
102.4 Summary of DBMS_MGD_ID_UTL Subprograms
   102.4.1 ADD_SCHEME Procedure
   102.4.2 CREATE_CATEGORY Function
   102.4.3 EPC_TO_ORACLE_SCHEME Function
   102.4.4 GET_CATEGORY_ID Function
   102.4.5 GET_COMPONENTS Function
   102.4.6 GET_ENCODINGS Function
   102.4.7 GET_JAVA_LOGGING_LEVEL Function
   102.4.8 GET_PLSQL_LOGGING_LEVEL Function
   102.4.9 GET_SCHEME_NAMES Function
   102.4.10 GET_TDT_XML Function
   102.4.11 GET_VALIDATOR Function
   102.4.12 REFRESH_CATEGORY Function
   102.4.13 REMOVE_CATEGORY Procedure
   102.4.14 REMOVE_PROXY Procedure
   102.4.15 REMOVE_SCHEME Procedure
   102.4.16 SET_JAVA_LOGGING_LEVEL Procedure
   102.4.17 SET_PLSQL_LOGGING_LEVEL Procedure
   102.4.18 SET_PROXY Procedure
   102.4.19 VALIDATE_SCHEME Function
## DBMS_MGWADM

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>103.1</td>
<td>DBMS_MGWADM Security Model</td>
<td>103-1</td>
</tr>
<tr>
<td>103.2</td>
<td>DBMS_MGWADM Deprecated Subprograms</td>
<td>103-1</td>
</tr>
<tr>
<td>103.3</td>
<td>DBMS_MGWADM Constants</td>
<td>103-2</td>
</tr>
<tr>
<td>103.4</td>
<td>DBMS_MGWADM Data Structures</td>
<td>103-5</td>
</tr>
<tr>
<td>103.4.1</td>
<td>DBMS_MGWADM SYS.MGW_MQSERIES_PROPERTIES Object Type</td>
<td>103-5</td>
</tr>
<tr>
<td>103.4.2</td>
<td>DBMS_MGWADM SYS.MGW_PROPERTIES Object Type</td>
<td>103-7</td>
</tr>
<tr>
<td>103.4.3</td>
<td>DBMS_MGWADM SYS.MGW_PROPERTY Object Type</td>
<td>103-8</td>
</tr>
<tr>
<td>103.4.4</td>
<td>SYS.MGW_TIBRV_PROPERTIES Object Type</td>
<td>103-9</td>
</tr>
<tr>
<td>103.5</td>
<td>Summary of DBMS_MGWADM Subprograms</td>
<td>103-10</td>
</tr>
<tr>
<td>103.5.1</td>
<td>ADD_SUBSCRIBER Procedure</td>
<td>103-12</td>
</tr>
<tr>
<td>103.5.2</td>
<td>ALTER_AGENT Procedures</td>
<td>103-15</td>
</tr>
<tr>
<td>103.5.3</td>
<td>ALTER_JOB Procedure</td>
<td>103-17</td>
</tr>
<tr>
<td>103.5.4</td>
<td>ALTER_MSGSYSTEM_LINK Procedure for TIB/Rendezvous</td>
<td>103-19</td>
</tr>
<tr>
<td>103.5.5</td>
<td>ALTER_MSGSYSTEM_LINK Procedure for WebSphere MQ</td>
<td>103-20</td>
</tr>
<tr>
<td>103.5.6</td>
<td>ALTER_PROPAGATION_SCHEDULE Procedure</td>
<td>103-21</td>
</tr>
<tr>
<td>103.5.7</td>
<td>ALTER_SUBSCRIBER Procedure</td>
<td>103-22</td>
</tr>
<tr>
<td>103.5.8</td>
<td>CLEANUP_GATEWAY Procedures</td>
<td>103-24</td>
</tr>
<tr>
<td>103.5.9</td>
<td>CREATE_AGENT Procedure</td>
<td>103-27</td>
</tr>
<tr>
<td>103.5.10</td>
<td>CREATE_JOB Procedure</td>
<td>103-28</td>
</tr>
<tr>
<td>103.5.11</td>
<td>CREATE_MSGSYSTEM_LINK Procedures for TIB/Rendezvous</td>
<td>103-31</td>
</tr>
<tr>
<td>103.5.12</td>
<td>CREATE_MSGSYSTEM_LINK Procedures for WebSphere MQ</td>
<td>103-32</td>
</tr>
<tr>
<td>103.5.13</td>
<td>DB_CONNECT_INFO Procedure</td>
<td>103-33</td>
</tr>
<tr>
<td>103.5.14</td>
<td>DISABLE_JOB Procedure</td>
<td>103-34</td>
</tr>
<tr>
<td>103.5.15</td>
<td>DISABLE_PROPAGATION_SCHEDULE Procedure</td>
<td>103-35</td>
</tr>
<tr>
<td>103.5.16</td>
<td>ENABLE_JOB Procedure</td>
<td>103-35</td>
</tr>
<tr>
<td>103.5.17</td>
<td>ENABLE_PROPAGATION_SCHEDULE Procedure</td>
<td>103-36</td>
</tr>
<tr>
<td>103.5.18</td>
<td>REGISTER_FOREIGN_QUEUE Procedure</td>
<td>103-36</td>
</tr>
<tr>
<td>103.5.19</td>
<td>REMOVE_AGENT Procedure</td>
<td>103-37</td>
</tr>
<tr>
<td>103.5.20</td>
<td>REMOVE_JOB Procedure</td>
<td>103-38</td>
</tr>
<tr>
<td>103.5.21</td>
<td>REMOVE_MSGSYSTEM_LINK Procedure</td>
<td>103-38</td>
</tr>
<tr>
<td>103.5.22</td>
<td>REMOVE_OPTION Procedure</td>
<td>103-39</td>
</tr>
<tr>
<td>103.5.23</td>
<td>REMOVE_SUBSCRIBER Procedure</td>
<td>103-40</td>
</tr>
<tr>
<td>103.5.24</td>
<td>RESET_JOB Procedure</td>
<td>103-41</td>
</tr>
<tr>
<td>103.5.25</td>
<td>RESET_SUBSCRIBER Procedure</td>
<td>103-42</td>
</tr>
<tr>
<td>103.5.26</td>
<td>SCHEDULE_PROPAGATION Procedure</td>
<td>103-42</td>
</tr>
<tr>
<td>103.5.27</td>
<td>SET_LOG_LEVEL Procedures</td>
<td>103-44</td>
</tr>
<tr>
<td>103.5.28</td>
<td>SET_OPTION Procedure</td>
<td>103-44</td>
</tr>
<tr>
<td>103.5.29</td>
<td>SHUTDOWN Procedures</td>
<td>103-46</td>
</tr>
<tr>
<td>103.5.30</td>
<td>STARTUP Procedures</td>
<td>103-46</td>
</tr>
</tbody>
</table>
104 DBMS_MGWMSG

104.1 DBMS_MGWMSG Security Model 104-1
104.2 DBMS_MGWMSG Constants 104-1
104.3 DBMS_MGWMSG Types 104-3
  104.3.1 DBMS_MGWMSG SYS.MGW_NAME_VALUE_T Type 104-3
  104.3.2 DBMS_MGWMSG SYS.MGW_NAME_TYPE_ARRAY_T Type 104-6
  104.3.3 DBMS_MGWMSG SYS.MGW_TEXT_VALUE_T Type 104-6
  104.3.4 DBMS_MGWMSG SYS.MGW_RAW_VALUE_T Type 104-7
  104.3.5 DBMS_MGWMSG SYS.MGW_BASIC_MSG_T Type 104-7
  104.3.6 DBMS_MGWMSG SYS.MGW_NUMBER_ARRAY_T Type 104-8
  104.3.7 DBMS_MGWMSG SYS.MGW_TIBRV_FIELD_T Type 104-8
  104.3.8 DBMS_MGWMSG SYS.MGW_TIBRV_MSG_T Type 104-10
104.4 Summary of DBMS_MGWMSG Subprograms 104-18
  104.4.1 LCR_TO_XML Function 104-19
  104.4.2 NVARRAY_ADD Procedure 104-20
  104.4.3 NVARRAY_FIND_NAME Function 104-20
  104.4.4 NVARRAY_FIND_NAME_TYPE Function 104-21
  104.4.5 NVARRAY_GET Function 104-21
  104.4.6 NVARRAY_GET_BOOLEAN Function 104-22
  104.4.7 NVARRAY_GET_BYTE Function 104-22
  104.4.8 NVARRAY_GET_DATE Function 104-23
  104.4.9 NVARRAY_GET_DOUBLE Function 104-23
  104.4.10 NVARRAY_GET_FLOAT Function 104-24
  104.4.11 NVARRAY_GET_INTEGER Function 104-25
  104.4.12 NVARRAY_GET_LONG Function 104-25
  104.4.13 NVARRAY_GET_RAW Function 104-26
  104.4.14 NVARRAY_GET_SHORT Function 104-26
  104.4.15 NVARRAY_GET_TEXT Function 104-27
  104.4.16 XML_TO_LCR Function 104-27

105 DBMS_MONITOR

105.1 Summary of DBMS_MONITOR Subprograms 105-1
  105.1.1 CLIENT_ID_STAT_DISABLE Procedure 105-2
  105.1.2 CLIENT_ID_STAT_ENABLE Procedure 105-2
  105.1.3 CLIENT_ID_TRACE_DISABLE Procedure 105-3
  105.1.4 CLIENT_ID_TRACE_ENABLE Procedure 105-3
105.1.5 DATABASE_TRACE_DISABLE Procedure 105-4
105.1.6 DATABASE_TRACE_ENABLE Procedure 105-4
105.1.7 SERV_MOD_ACT_STAT_DISABLE Procedure 105-5
105.1.8 SERV_MOD_ACT_STAT_ENABLE Procedure 105-5
105.1.9 SERV_MOD_ACT_TRACE_DISABLE Procedure 105-7
105.1.10 SERV_MOD_ACT_TRACE_ENABLE Procedure 105-8
105.1.11 SESSION_TRACE_DISABLE Procedure 105-9
105.1.12 SESSION_TRACE_ENABLE Procedure 105-10

106 DBMS_MVIEW

106.1 DBMS_MVIEW Operational Notes 106-1
106.2 DBMS_MVIEW Security Model 106-1
106.3 DBMS_MVIEW Rules and Limits 106-2
106.4 Summary of DBMS_MVIEW Subprograms 106-2
   106.4.1 BEGIN_TABLE_REORGANIZATION Procedure 106-3
   106.4.2 END_TABLE_REORGANIZATION Procedure 106-3
   106.4.3 ESTIMATE_MVIEW_SIZE Procedure 106-3
   106.4.4 EXPLAIN_MVIEW Procedure 106-4
   106.4.5 EXPLAIN_REWRITE Procedure 106-5
   106.4.6 I_AM_A_REFRESH Function 106-6
   106.4.7 PMARKER Function 106-7
   106.4.8 PURGE_DIRECT_LOAD_LOG Procedure 106-7
   106.4.9 PURGE_LOG Procedure 106-7
   106.4.10 PURGE_MVIEW_FROM_LOG Procedure 106-8
   106.4.11 REFRESH Procedures 106-9
   106.4.12 REFRESH_ALL_MVIEWS Procedure 106-11
   106.4.13 REFRESH_DEPENDENT Procedures 106-12
   106.4.14 REGISTER_MVIEW Procedure 106-15
   106.4.15 UNREGISTER_MVIEW Procedure 106-16

107 DBMS_MVIEW_STATS

107.1 DBMS_MVIEW_STATS Overview 107-1
107.2 DBMS_MVIEW_STATS Security Model 107-1
107.3 Summary of DBMS_MVIEW_STATS Subprograms 107-1
   107.3.1 PURGE_REFRESH_STATS Procedure 107-2
   107.3.2 SET_MVREF_STATS_PARAMS Procedure 107-3
   107.3.3 SET_SYSTEM_DEFAULT Procedure 107-4
## DBMS_NETWORK_ACL_ADMIN

108.1 DBMS_NETWORK_ACL_ADMIN Overview 108-1
108.2 DBMS_NETWORK_ACL_ADMIN Deprecated Subprograms 108-1
108.3 DBMS_NETWORK_ACL_ADMIN Security Model 108-2
108.4 DBMS_NETWORK_ACL_ADMIN Constants 108-2
108.5 DBMS_NETWORK_ACL_ADMIN Exceptions 108-2
108.6 DBMS_NETWORK_ACL_ADMIN Examples 108-3
108.7 Summary of DBMS_NETWORK_ACL_ADMIN Subprograms 108-5
  108.7.1 ADD_PRIVILEGE Procedure 108-6
  108.7.2 APPEND_HOST_ACE Procedure 108-7
  108.7.3 APPEND_HOST_ACL Procedure 108-8
  108.7.4 APPEND_WALLET_ACE Procedure 108-9
  108.7.5 APPEND_WALLET_ACL Procedure 108-10
  108.7.6 ASSIGN_ACL Procedure 108-11
  108.7.7 ASSIGN_WALLET_ACL Procedure 108-12
  108.7.8 CHECK_PRIVILEGE Function 108-13
  108.7.9 CHECK_PRIVILEGE_ACLID Function 108-14
  108.7.10 CREATE_ACL Procedure 108-15
  108.7.11 DELETE_PRIVILEGE Procedure 108-16
  108.7.12 DROP_ACL Procedure 108-17
  108.7.13 REMOVE_HOST_ACE Procedure 108-18
  108.7.14 REMOVE_WALLET_ACE Procedure 108-18
  108.7.15 SET_HOST_ACL Procedure 108-19
  108.7.16 SET_WALLET_ACL Procedure 108-20
  108.7.17 UNASSIGN_ACL Procedure 108-20
  108.7.18 UNASSIGN_WALLET_ACL Procedure 108-21

## DBMS_NETWORK_ACL_UTILITY

109.1 DBMS_NETWORK_ACL_UTILITY Security Model 109-1
109.2 DBMS_NETWORK_ACL_UTILITY Examples 109-1
109.3 Summary of DBMS_NETWORK_ACL_UTILITY Subprograms 109-2
  109.3.1 CONTAINS_HOST Function 109-3
  109.3.2 DOMAIN_LEVEL Function 109-4
  109.3.3 DOMAINS Function 109-5
  109.3.4 EQUALS_HOST Function 109-5

## DBMS_ODCI

110.1 Summary of DBMS_ODCI Subprograms 110-1
## 111 DBMS_OUTLN

111.1 DBMS_OUTLN Overview
111.2 DBMS_OUTLN Security Model
111.3 Summary of DBMS_OUTLN Subprograms
  111.3.1 CLEAR_USED Procedure
  111.3.2 CREATE_OUTLINE Procedure
  111.3.3 DROP_BY_CAT Procedure
  111.3.4 DROP_UNUSED Procedure
  111.3.5 EXACT_TEXT_SIGNATURES Procedure
  111.3.6 UPDATE_BY_CAT Procedure
  111.3.7 UPDATE_SIGNATURES Procedure

## 112 DBMS_OUTPUT

112.1 DBMS_OUTPUT Overview
112.2 DBMS_OUTPUT Security Model
112.3 DBMS_OUTPUT Operational Notes
112.4 DBMS_OUTPUT Exceptions
112.5 DBMS_OUTPUT Rules and Limits
112.6 DBMS_OUTPUT Examples
112.7 DBMS_OUTPUT Data Structures
  112.7.1 CHARARR Table Type
  112.7.2 DBMS_OUTPUT DBMSOUTPUT_LINESARRAY Object Type
112.8 Summary of DBMS_OUTPUT Subprograms
  112.8.1 DISABLE Procedure
  112.8.2 ENABLE Procedure
  112.8.3 GET_LINE Procedure
  112.8.4 GET_LINES Procedure
  112.8.5 NEW_LINE Procedure
  112.8.6 PUT Procedure
  112.8.7 PUT_LINE Procedure

## 113 DBMS_PARALLEL_EXECUTE

113.1 DBMS_PARALLEL_EXECUTE Overview
113.2 DBMS_PARALLEL_EXECUTE Security Model
113.3 DBMS_PARALLEL_EXECUTE Constants
113.4 DBMS_PARALLEL_EXECUTE Views
113.5 DBMS_PARALLEL_EXECUTE Exceptions
113.6 DBMS_PARALLEL_EXECUTE Examples

113.7 Summary of DBMS_PARALLEL_EXECUTE Subprograms
   113.7.1 ADM_DROP_CHUNKS Procedure
   113.7.2 ADM_DROP_TASK Procedure
   113.7.3 ADM_TASK_STATUS Function
   113.7.4 ADM_STOP_TASK Procedure
   113.7.5 CREATE_TASK Procedure
   113.7.6 CREATE_CHUNKS_BY_NUMBER_COL Procedure
   113.7.7 CREATE_CHUNKS_BY_ROWID Procedure
   113.7.8 CREATE_CHUNKS_BY_SQL Procedure
   113.7.9 DROP_TASK Procedure
   113.7.10 DROP_CHUNKS Procedure
   113.7.11 GENERATE_TASK_NAME Function
   113.7.12 GET_NUMBER_COL_CHUNK Procedure
   113.7.13 GET_ROWID_CHUNK Procedure
   113.7.14 PURGE_PROCESSED_CHUNKS Procedure
   113.7.15 RESUME_TASK Procedures
   113.7.16 RUN_TASK Procedure
   113.7.17 SET_CHUNK_STATUS Procedure
   113.7.18 STOP_TASK Procedure
   113.7.19 TASK_STATUS Procedure

114 DBMS_PART
   114.1 DBMS_PART Security Model
   114.2 DBMS_PART Operational Notes
   114.3 Summary of DBMS_PART Subprograms
      114.3.1 CLEANUP_GIDX Procedure
      114.3.2 CLEANUP_GIDX_JOB Procedure
      114.3.3 CLEANUP_ONLINE_OP Procedure

115 DBMS_PCLXUTIL
   115.1 DBMS_PCLXUTIL Overview
   115.2 DBMS_PCLXUTIL Security Model
   115.3 DBMS_PCLXUTIL Operational Notes
   115.4 DBMS_PCLXUTIL Rules and Limits
   115.5 Summary of DBMS_PCLXUTIL Subprograms
      115.5.1 BUILD_PART_INDEX Procedure
## DBMS_PIPE

119.1 DBMS_PIPE Overview
119.2 DBMS_PIPE Security Model
119.3 DBMS_PIPE Constants
119.4 DBMS_PIPE Operational Notes
119.5 DBMS_PIPE Exceptions
119.6 DBMS_PIPE Examples
119.7 Summary of DBMS_PIPE Subprograms
119.7.1 CREATE_PIPE Function
119.7.2 NEXT_ITEM_TYPE Function
119.7.3 PACK_MESSAGE Procedures
119.7.4 PURGE Procedure
119.7.5 RECEIVE_MESSAGE Function
119.7.6 RESET_BUFFER Procedure
119.7.7 REMOVE_PIPE Function
119.7.8 SEND_MESSAGE Function
119.7.9 UNIQUE_SESSION_NAME Function
119.7.10 UNPACK_MESSAGE Procedures

## DBMS_PLSQL_CODE_COVERAGE

120.1 DBMS_PLSQL_CODE_COVERAGE Overview
120.2 DBMS_PLSQL_CODE_COVERAGE Security Model
120.3 Summary of DBMS_PLSQL_CODE_COVERAGE Subprograms
120.3.1 CREATE_COVERAGE_TABLES Procedure
120.3.2 START_COVERAGE Function
120.3.3 STOP_COVERAGE Procedure

## DBMS_PREDICTIVE_ANALYTICS

121.1 DBMS_PREDICTIVE_ANALYTICS Overview
121.2 DBMS_PREDICTIVE_ANALYTICS Security Model
121.3 Summary of DBMS_PREDICTIVE_ANALYTICS Subprograms
121.3.1 EXPLAIN Procedure
121.3.2 PREDICT Procedure
121.3.3 PROFILE Procedure

## DBMS_PREPROCESSOR

122.1 DBMS_PREPROCESSOR Overview
122.2 DBMS_PREPROCESSOR Operating Notes
### 126  DBMS_PROTECTION_ADM

126.1  DBMS_PROTECTION_ADM Overview  
126.2  DBMS_PROTECTION_ADM Security Model  
126.3  Summary of DBMS_PROTECTION_ADM Subprograms  
  
  126.3.1  ALTER_PROPAGATION Procedure  
  126.3.2  CREATE_PROPAGATION Procedure  
  126.3.3  DROP_PROPAGATION Procedure  
  126.3.4  START_PROPAGATION Procedure  
  126.3.5  STOP_PROPAGATION Procedure

### 127  DBMS_QOPATCH

127.1  DBMS_QOPATCH Overview  
127.2  DBMS_QOPATCH Security Model  
127.3  DBMS_QOPATCH Operational Notes  
127.4  DBMS_QOPATCH Exceptions  
127.5  Summary of DBMS_QOPATCH Subprograms  
  
  127.5.1  CHECK_PATCH_INSTALLED Function  
  127.5.2  GET_OPATCH_BUGS Function  
  127.5.3  GET_OPATCH_COUNT Function  
  127.5.4  GET_OPATCH_DATA Function  
  127.5.5  GET_OPATCH_FILES Function  
  127.5.6  GET_OPATCH_INSTALL_INFO Function  
  127.5.7  GET_OPATCH_LIST Function  
  127.5.8  GET_OPATCH_LSIENCY  
  127.5.9  GET_OPATCH_OLES Function  
  127.5.10  GET_OPATCH_PREFS Function  
  127.5.11  GET_OPATCH_XSLT  
  127.5.12  GET_PATCH_DETAILS Function  
  127.5.13  GET_SQLPATCH_STATUS Procedure  
  127.5.14  IS_PATCH_INSTALLED Function  
  127.5.15  OPATCH_COMPARE_CURRENT Function  
  127.5.16  OPATCH_COMPARE_NODES Function  
  127.5.17  SET_CURRENT_OPINST Procedure
130.7.2 ABORT_ROLLBACK Procedure 130-6
130.7.3 ABORT_UPDATE Procedure 130-6
130.7.4 CAN_REDEF_TABLE Procedure 130-7
130.7.5 COPY_TABLE_DEPENDENTS Procedure 130-7
130.7.6 EXECUTE_UPDATE Procedure 130-9
130.7.7 FINISH_REDEF_TABLE Procedure 130-9
130.7.8 REDEF_TABLE Procedure 130-10
130.7.9 REGISTER_DEPENDENT_OBJECT Procedure 130-12
130.7.10 ROLLBACK Procedure 130-13
130.7.11 SET_PARAM Procedure 130-14
130.7.12 START_REDEF_TABLE Procedure 130-14
130.7.13 SYNC_INTERIM_TABLE Procedure 130-16
130.7.14 UNREGISTER_DEPENDENT_OBJECT Procedure 130-17

131 DBMS_REFRESH

131.1 DBMS_REFRESH Overview 131-1
131.2 DBMS_REFRESH Security Model 131-1
131.3 Summary of DBMS_REFRESH Subprograms 131-1
  131.3.1 ADD Procedure 131-2
  131.3.2 CHANGE Procedure 131-3
  131.3.3 DESTROY Procedure 131-4
  131.3.4 MAKE Procedure 131-4
  131.3.5 REFRESH Procedure 131-6
  131.3.6 SUBTRACT Procedure 131-7

132 DBMS_REPAIR

132.1 DBMS_REPAIR Overview 132-1
132.2 DBMS_REPAIR Security Model 132-1
132.3 DBMS_REPAIR Constants 132-1
132.4 DBMS_REPAIR Operating Notes 132-2
132.5 DBMS_REPAIR Exceptions 132-3
132.6 DBMS_REPAIR Examples 132-4
132.7 Summary of DBMS_REPAIR Subprograms 132-4
  132.7.1 ADMIN_TABLES Procedure 132-5
  132.7.2 CHECK_OBJECT Procedure 132-6
  132.7.3 DUMP_ORPHAN_KEYS Procedure 132-7
  132.7.4 FIX_CORRUPT_BLOCKS Procedure 132-8
  132.7.5 ONLINE_INDEX_CLEAN Function 132-9
  132.7.6 REBUILD_FREELISTS Procedure 132-9
133 DBMS_RESCONFIG

133.1 DBMS_RESCONFIG Overview

133.2 Summary of DBMS_RESCONFIG Subprograms

133.2.1 ADDREPOSITORYRESCONFIG Procedure

133.2.2 ADDRESCONFIG Procedure

133.2.3 APPENDRESCONFIG Procedure

133.2.4 DELETEREPOSITORYRESCONFIG Procedure

133.2.5 DELETERESCONFIG Procedures

133.2.6 GETLISTENERS Function

133.2.7 GETREPOSITORYRESCONFIG Function

133.2.8 GETREPOSITORYRESCONFIGPATHS Function

133.2.9 GETRESCONFIG Function

133.2.10 GETRESCONFIGPATHS Function

133.2.11 PATCHREPOSITORYRESCONFIGLIST Procedure

134 DBMS_RESOURCE_MANAGER

134.1 DBMS_RESOURCE_MANAGER Deprecated Subprograms

134.2 DBMS_RESOURCE_MANAGER Security Model

134.3 DBMS_RESOURCE_MANAGER Constants

134.4 Summary of DBMS_RESOURCE_MANAGER Subprograms

134.4.1 BEGIN_SQL_BLOCK Procedure

134.4.2 CALIBRATE_IO Procedure

134.4.3 CLEAR_PENDING_AREA Procedure

134.4.4 CREATE_CATEGORY Procedure

134.4.5 CREATE_CDB_PLAN Procedure

134.4.6 CREATE_CDB_PLAN_DIRECTIVE Procedure

134.4.7 CREATE_CDB_PROFILE_DIRECTIVE Procedure

134.4.8 CREATE_CONSUMER_GROUP Procedure

134.4.9 CREATE_PENDING_AREA Procedure

134.4.10 CREATE_PLAN Procedure

134.4.11 CREATE_PLAN_DIRECTIVE Procedure

134.4.12 CREATE_SIMPLE_PLAN Procedure

134.4.13 DELETE_CATEGORY Procedure

134.4.14 DELETE_CDB_PLAN Procedure

134.4.15 DELETE_CDB_PLAN_DIRECTIVE Procedure

134.4.16 DELETE_CDB_PROFILE_DIRECTIVE Procedure
134.4.17  DELETE_CONSUMER_GROUP Procedure 134-23
134.4.18  DELETE_PLAN Procedure 134-23
134.4.19  DELETE_PLAN.Cascade Procedure 134-24
134.4.20  DELETE_PLAN_DIRECTIVE Procedure 134-24
134.4.21  DEQUEUE_PARALLEL_STATEMENT Procedure 134-24
134.4.22  END_SQL_BLOCK Procedure 134-25
134.4.23  SET_CONSUMER_GROUP_MAPPING Procedure 134-25
134.4.24  SET_CONSUMER_GROUP_MAPPING_PRI Procedure 134-26
134.4.25  SET_INITIAL_CONSUMER_GROUP Procedure 134-27
134.4.26  SUBMIT_PENDING_AREA Procedure 134-29
134.4.27  SWITCH_CONSUMER_GROUP_FOR_SESS Procedure 134-29
134.4.28  SWITCH_CONSUMER_GROUP_FOR_USER Procedure 134-29
134.4.29  SWITCH_PLAN Procedure 134-30
134.4.30  UPDATE_CATEGORY Procedure 134-31
134.4.31  UPDATE_CDB_AUTO_TASK_DIRECTIVE Procedure 134-34
134.4.32  UPDATE_CDB_DEFAULT_DIRECTIVE Procedure 134-32
134.4.33  UPDATE_CDB_PLAN Procedure 134-33
134.4.34  UPDATE_CDB_PLAN_DIRECTIVE Procedure 134-34
134.4.35  UPDATE_CDB_PROFILE_DIRECTIVE Procedure 134-35
134.4.36  UPDATE_CONSUMER_GROUP Procedure 134-36
134.4.37  UPDATE_PLAN Procedure 134-36
134.4.38  UPDATE PLAN DIRECTIVE Procedure 134-37
134.4.39  VALIDATE_PENDING_AREA Procedure 134-42

135  DBMS_RESOURCE_MANAGER_PRIVS

135.1  Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms 135-1
135.1.1  GRANT_SWITCH_CONSUMER_GROUP Procedure 135-1
135.1.2  GRANT_SYSTEM_PRIVILEGE Procedure 135-2
135.1.3  REVOKE_SWITCH_CONSUMER_GROUP Procedure 135-3
135.1.4  REVOKE_SYSTEM_PRIVILEGE Procedure 135-4

136  DBMS_RESULT_CACHE

136.1  DBMS_RESULT_CACHE Security Model 136-1
136.2  DBMS_RESULT_CACHE Constants 136-1
136.3  Summary of DBMS_RESULT_CACHE Subprograms 136-2
136.3.1  BLACK_LIST Function 136-2
136.3.2  BLACK_LIST_ADD Procedure 136-3
136.3.3  BLACK_LIST_CLEAR Procedure 136-3
136.3.4  BLACK_LIST_REMOVE Procedure 136-3
137 DBMS_RESUMABLE

137.1 DBMS_RESUMABLE Operational Notes 137-1
137.2 Summary of DBMS_RESUMABLE Subprograms 137-1
  137.2.1 ABORT Procedure 137-2
  137.2.2 GET_SESSION_TIMEOUT Function 137-2
  137.2.3 GET_TIMEOUT Function 137-3
  137.2.4 SET_SESSION_TIMEOUT Procedure 137-3
  137.2.5 SET_TIMEOUT Procedure 137-4
  137.2.6 SPACE_ERROR_INFO Function 137-4

138 DBMS_RLS

138.1 DBMS_RLS Overview 138-1
138.2 DBMS_RLS Security Model 138-2
138.3 DBMS_RLS Constants 138-2
138.4 DBMS_RLS Operational Notes 138-3
138.5 DBMS_RLS Rules and Limits 138-3
138.6 Summary of DBMS_RLS Subprograms 138-3
  138.6.1 ADD_GROUPED_POLICY Procedure 138-4
  138.6.2 ADD_POLICY Procedure 138-6
  138.6.3 ADD_POLICY_CONTEXT Procedure 138-10
  138.6.4 ALTER_POLICY Procedure 138-11
  138.6.5 ALTER_GROUPED_POLICY Procedure 138-12
  138.6.6 CREATE_POLICY_GROUP Procedure 138-14
  138.6.7 DELETE_POLICY_GROUP Procedure 138-14
  138.6.8 DISABLE_GROUPED_POLICY Procedure 138-15
  138.6.9 DROP_GROUPED_POLICY Procedure 138-15
  138.6.10 DROP_POLICY Procedure 138-16
  138.6.11 DROP_POLICY_CONTEXT Procedure 138-17
  138.6.12 ENABLE_GROUPED_POLICY Procedure 138-17
  138.6.13 ENABLE_POLICY Procedure 138-18
  138.6.14 REFRESH_GROUPED_POLICY Procedure 138-19
139 DBMS_ROLLING

139.1 DBMS_ROLLING Overview
139.2 DBMS_ROLLING Security Model
139.3 Summary of DBMS_ROLLING Subprograms
  139.3.1 INIT_PLAN Procedure
  139.3.2 DESTROY_PLAN Procedure
  139.3.3 BUILD_PLAN Procedure
  139.3.4 SET_PARAMETER Procedure
  139.3.5 START_PLAN Procedure
  139.3.6 SWITCHOVER Procedure
  139.3.7 FINISH_PLAN Procedure
  139.3.8 ROLLBACK_PLAN Procedure

140 DBMS_ROWID

140.1 DBMS_ROWID Security Model
140.2 DBMS_ROWID Types
140.3 DBMS_ROWID Exceptions
140.4 DBMS_ROWID Operational Notes
140.5 DBMS_ROWID Examples
140.6 Summary of DBMS_ROWID Subprograms
  140.6.1 ROWID_BLOCK_NUMBER Function
  140.6.2 ROWID_CREATE Function
  140.6.3 ROWID_INFO Procedure
  140.6.4 ROWID_OBJECT Function
  140.6.5 ROWID_RELATIVE_FNO Function
  140.6.6 ROWID_ROW_NUMBER Function
  140.6.7 ROWID_TO_ABSOLUTE_FNO Function
  140.6.8 ROWID_TO_EXTENDED Function
  140.6.9 ROWID_TO_RESTRICTED Function
  140.6.10 ROWID_TYPE Function
  140.6.11 ROWID_VERIFY Function

141 DBMS_RULE

141.1 DBMS_RULE Overview
141.2 DBMS_RULE Security Model
141.3 Summary of DBMS_RULE Subprograms
  141.3.1 CLOSE_ITERATOR Procedure
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>141.3.2</td>
<td>EVALUATE Procedure</td>
<td>141-3</td>
</tr>
<tr>
<td>141.3.3</td>
<td>EVALUATE_EXPRESSION Procedure</td>
<td>141-7</td>
</tr>
<tr>
<td>141.3.4</td>
<td>EVALUATE_EXPRESSION_ITERATOR Procedure</td>
<td>141-8</td>
</tr>
<tr>
<td>141.3.5</td>
<td>EVALUATE_RULE Procedure</td>
<td>141-8</td>
</tr>
<tr>
<td>141.3.6</td>
<td>EVALUATE_RULE_ITERATOR Procedure</td>
<td>141-9</td>
</tr>
<tr>
<td>141.3.7</td>
<td>GET_NEXT_HIT Function</td>
<td>141-10</td>
</tr>
<tr>
<td>141.3.8</td>
<td>GET_NEXT_RESULT Function</td>
<td>141-11</td>
</tr>
<tr>
<td>141.3.9</td>
<td>IS_FAST Procedure</td>
<td>141-11</td>
</tr>
</tbody>
</table>

### 142 DBMS_RULE_ADM

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>142.1</td>
<td>DBMS_RULE_ADM Overview</td>
<td>142-1</td>
</tr>
<tr>
<td>142.2</td>
<td>DBMS_RULE_ADM Security Model</td>
<td>142-1</td>
</tr>
<tr>
<td>142.3</td>
<td>Summary of DBMS_RULE_ADM Subprograms</td>
<td>142-1</td>
</tr>
<tr>
<td>142.3.1</td>
<td>ADD_RULE Procedure</td>
<td>142-2</td>
</tr>
<tr>
<td>142.3.2</td>
<td>ALTER_EVALUATION_CONTEXT Procedure</td>
<td>142-4</td>
</tr>
<tr>
<td>142.3.3</td>
<td>ALTER_RULE Procedure</td>
<td>142-6</td>
</tr>
<tr>
<td>142.3.4</td>
<td>CREATE_EVALUATION_CONTEXT Procedure</td>
<td>142-8</td>
</tr>
<tr>
<td>142.3.5</td>
<td>CREATE_RULE Procedure</td>
<td>142-10</td>
</tr>
<tr>
<td>142.3.6</td>
<td>CREATE_RULE_SET Procedure</td>
<td>142-11</td>
</tr>
<tr>
<td>142.3.7</td>
<td>DROP_EVALUATION_CONTEXT Procedure</td>
<td>142-12</td>
</tr>
<tr>
<td>142.3.8</td>
<td>DROP_RULE Procedure</td>
<td>142-13</td>
</tr>
<tr>
<td>142.3.9</td>
<td>DROP_RULE_SET Procedure</td>
<td>142-14</td>
</tr>
<tr>
<td>142.3.10</td>
<td>GRANT_OBJECT_PRIVILEGE Procedure</td>
<td>142-14</td>
</tr>
<tr>
<td>142.3.11</td>
<td>GRANT_SYSTEM_PRIVILEGE Procedure</td>
<td>142-16</td>
</tr>
<tr>
<td>142.3.12</td>
<td>REMOVE_RULE Procedure</td>
<td>142-18</td>
</tr>
<tr>
<td>142.3.13</td>
<td>REVOKE_OBJECT_PRIVILEGE Procedure</td>
<td>142-19</td>
</tr>
<tr>
<td>142.3.14</td>
<td>REVOKE_SYSTEM_PRIVILEGE Procedure</td>
<td>142-20</td>
</tr>
</tbody>
</table>

### 143 DBMS_SCHEDULER

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>143.1</td>
<td>DBMS_SCHEDULER Deprecated Subprograms</td>
<td>143-1</td>
</tr>
<tr>
<td>143.2</td>
<td>DBMS_SCHEDULER Security Model</td>
<td>143-1</td>
</tr>
<tr>
<td>143.3</td>
<td>DBMS_SCHEDULER Rules and Limits</td>
<td>143-1</td>
</tr>
<tr>
<td>143.4</td>
<td>DBMS_SCHEDULER Operational Notes</td>
<td>143-2</td>
</tr>
<tr>
<td>143.5</td>
<td>DBMS_SCHEDULER Data Structures</td>
<td>143-12</td>
</tr>
<tr>
<td>143.5.1</td>
<td>DBMS_SCHEDULER JOBARG Object Type</td>
<td>143-12</td>
</tr>
<tr>
<td>143.5.2</td>
<td>JOBARG_ARRAY Table Type</td>
<td>143-14</td>
</tr>
<tr>
<td>143.5.3</td>
<td>DBMS_SCHEDULER JOB_DEFINITION Object Type</td>
<td>143-14</td>
</tr>
<tr>
<td>143.5.4</td>
<td>JOB_DEFINITION_ARRAY Table Type</td>
<td>143-17</td>
</tr>
<tr>
<td>143.5.5</td>
<td>JOBATTR Object Type</td>
<td>143-18</td>
</tr>
</tbody>
</table>
143.5.6 JOBATTR_ARRAY Table Type
143-19
143.5.7 SCHEDULER$_STEP_TYPE Object Type
143-19
143.5.8 SCHEDULER$_STEP_TYPE_LIST Table Type
143-20
143.5.9 SCHEDULER$_EVENT_INFO Object Type
143-20
143.5.10 SCHEDULER_FILEWATCHER_RESULT Object Type
143-21
143.5.11 SCHEDULER_FILEWATCHER_REQUEST Object Type
143-22
143.6 Summary of DBMS_SCHEDULER Subprograms
143-23
143.6.1 ADD_EVENT_QUEUE_SUBSCRIBER Procedure
143-26
143.6.2 ADD_GROUP_MEMBER Procedure
143-27
143.6.3 ADD_JOB_EMAIL_NOTIFICATION Procedure
143-28
143.6.4 ADD_TO_INCOMPATIBILITY Procedure
143-30
143.6.5 ALTER_CHAIN Procedure
143-31
143.6.6 ALTER_RUNNING_CHAIN Procedure
143-33
143.6.7 CLOSE_WINDOW Procedure
143-35
143.6.8 COPY_JOB Procedure
143-36
143.6.9 CREATE_CHAIN Procedure
143-37
143.6.10 CREATE_CREDENTIAL Procedure
143-38
143.6.11 CREATE_DATABASE_DESTINATION Procedure
143-39
143.6.12 CREATE_EVENT_SCHEDULE Procedure
143-40
143.6.13 CREATE_FILE_WATCHER Procedure
143-41
143.6.14 CREATE_GROUP Procedure
143-42
143.6.15 CREATE_INCOMPATIBILITY Procedure
143-44
143.6.16 CREATE_JOB Procedure
143-45
143.6.17 CREATE_JOB_CLASS Procedure
143-53
143.6.18 CREATE_JOBS Procedure
143-55
143.6.19 CREATE_PROGRAM Procedure
143-56
143.6.20 CREATE_RESOURCE Procedure
143-60
143.6.21 CREATE_SCHEDULE Procedure
143-61
143.6.22 CREATE_WINDOW Procedure
143-62
143.6.23 DEFINE_ANYDATA_ARGUMENT Procedure
143-64
143.6.24 DEFINE_CHAIN_EVENT_STEP Procedure
143-65
143.6.25 DEFINE_CHAIN_RULE Procedure
143-66
143.6.26 DEFINE_CHAIN_STEP Procedure
143-69
143.6.27 DEFINE_METADATA_ARGUMENT Procedure
143-70
143.6.28 DEFINE_PROGRAM_ARGUMENT Procedure
143-71
143.6.29 DISABLE Procedure
143-73
143.6.30 DROP_AGENT_DESTINATION Procedure
143-75
143.6.31 DROP_CHAIN Procedure
143-76
143.6.32 DROP_CHAIN_RULE Procedure
143-77
143.6.33 DROP_CHAIN_STEP Procedure
143-78
143.6.34 DROP_CREDENTIAL Procedure
143-78
143.6.35 DROP_DATABASE_DESTINATION Procedure 143-79
143.6.36 DROP_FILE_WATCHER Procedure 143-80
143.6.37 DROP_GROUP Procedure 143-81
143.6.38 DROP_INCOMPATIBILITY Procedure 143-81
143.6.39 DROP_JOB Procedure 143-82
143.6.40 DROP_JOB_CLASS Procedure 143-83
143.6.41 DROP_PROGRAM Procedure 143-84
143.6.42 DROP_PROGRAM_ARGUMENT Procedure 143-84
143.6.43 DROP_RESOURCE Procedure 143-85
143.6.44 DROP_SCHEDULE Procedure 143-86
143.6.45 DROP_WINDOW Procedure 143-86
143.6.46 ENABLE Procedure 143-87
143.6.47 END_DETACHED_JOB_RUN Procedure 143-88
143.6.48 EVALUATECALENDARSTRING Procedure 143-89
143.6.49 EVALUATERUNNINGCHAIN Procedure 143-91
143.6.50 GENERATE_JOB_NAME Function 143-92
143.6.51 GET_AGENT_INFO Function 143-92
143.6.52 GET_AGENT_VERSION Function 143-93
143.6.53 GET_ATTRIBUTE Procedure 143-94
143.6.54 GET_FILE Procedure 143-94
143.6.55 GET_SCHEDULER_ATTRIBUTE Procedure 143-96
143.6.56 OPEN_WINDOW Procedure 143-97
143.6.57 PURGE_LOG Procedure 143-98
143.6.58 PUT_FILE Procedure 143-99
143.6.59 REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure 143-100
143.6.60 REMOVE_FROM_INCOMPATIBILITY Procedure 143-101
143.6.61 REMOVE_GROUP_MEMBER Procedure 143-101
143.6.62 REMOVE_JOB_EMAIL_NOTIFICATION Procedure 143-102
143.6.63 RESET_JOB_ARGUMENT_VALUE Procedure 143-103
143.6.64 RUN_CHAIN Procedure 143-104
143.6.65 RUN_JOB Procedure 143-105
143.6.66 SET_AGENT_REGISTRATION_PASS Procedure 143-107
143.6.67 SET_ATTRIBUTE Procedure 143-108
143.6.68 SET_ATTRIBUTE_NULL Procedure 143-124
143.6.69 SET_JOB_ANYDATA_VALUE Procedure 143-124
143.6.70 SET_JOB_ARGUMENT_VALUE Procedure 143-125
143.6.71 SET_JOB_ATTRIBUTES Procedure 143-127
143.6.72 SET_RESOURCE_CONSTRAINT Procedure 143-127
143.6.73 SET_SCHEDULER_ATTRIBUTE Procedure 143-128
143.6.74 STOP_JOB Procedure 143-130
## DBMS_SERVER_ALERT

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>144.1</td>
<td>DBMS_SERVER_ALERT Security Model</td>
<td>144-1</td>
</tr>
<tr>
<td>144.2</td>
<td>DBMS_SERVER_ALERT Object Types</td>
<td>144-1</td>
</tr>
<tr>
<td>144.3</td>
<td>DBMS_SERVER_ALERT Relational Operators</td>
<td>144-2</td>
</tr>
<tr>
<td>144.4</td>
<td>DBMS_SERVER_ALERT Supported Metrics</td>
<td>144-2</td>
</tr>
<tr>
<td>144.5</td>
<td>Summary of DBMS_SERVER_ALERT Subprograms</td>
<td>144-6</td>
</tr>
<tr>
<td>144.5.1</td>
<td>EXPAND_MESSAGE Function</td>
<td>144-7</td>
</tr>
<tr>
<td>144.5.2</td>
<td>GET_THRESHOLD Procedure</td>
<td>144-7</td>
</tr>
<tr>
<td>144.5.3</td>
<td>SET_THRESHOLD Procedure</td>
<td>144-8</td>
</tr>
</tbody>
</table>

## DBMS_SERVICE

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>145.1</td>
<td>DBMS_SERVICE Overview</td>
<td>145-1</td>
</tr>
<tr>
<td>145.2</td>
<td>DBMS_SERVICE Security Model</td>
<td>145-2</td>
</tr>
<tr>
<td>145.3</td>
<td>DBMS_SERVICE Constants</td>
<td>145-2</td>
</tr>
<tr>
<td>145.4</td>
<td>DBMS_SERVICE Operating Procedures</td>
<td>145-4</td>
</tr>
<tr>
<td>145.5</td>
<td>DBMS_SERVICE Exceptions</td>
<td>145-5</td>
</tr>
<tr>
<td>145.6</td>
<td>Summary of DBMS_SERVICE Subprograms</td>
<td>145-6</td>
</tr>
<tr>
<td>145.6.1</td>
<td>CREATE_SERVICE Procedure</td>
<td>145-6</td>
</tr>
<tr>
<td>145.6.2</td>
<td>DELETE_SERVICE Procedure</td>
<td>145-10</td>
</tr>
<tr>
<td>145.6.3</td>
<td>DISCONNECT_SESSION Procedure</td>
<td>145-10</td>
</tr>
<tr>
<td>145.6.4</td>
<td>MODIFY_SERVICE Procedure</td>
<td>145-11</td>
</tr>
<tr>
<td>145.6.5</td>
<td>START_SERVICE Procedure</td>
<td>145-15</td>
</tr>
<tr>
<td>145.6.6</td>
<td>STOP_SERVICE Procedure</td>
<td>145-15</td>
</tr>
</tbody>
</table>

## DBMS_SESSION

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>146.1</td>
<td>DBMS_SESSION Security Model</td>
<td>146-1</td>
</tr>
<tr>
<td>146.2</td>
<td>DBMS_SESSION Operational Notes</td>
<td>146-1</td>
</tr>
<tr>
<td>146.3</td>
<td>DBMS_SESSION Data Structures</td>
<td>146-1</td>
</tr>
<tr>
<td>146.3.1</td>
<td>DBMS_SERVICE INTEGER_ARRAY Table Type</td>
<td>146-1</td>
</tr>
<tr>
<td>146.3.2</td>
<td>DBMS_SERVICE LNAME_ARRAY Table Type</td>
<td>146-1</td>
</tr>
<tr>
<td>146.4</td>
<td>Summary of DBMS_SESSION Subprograms</td>
<td>146-2</td>
</tr>
<tr>
<td>146.4.1</td>
<td>ADD_SQL_CONNECTION_TEST Procedure</td>
<td>146-2</td>
</tr>
<tr>
<td>146.4.2</td>
<td>CLEAR_ALL_CONTEXT Procedure</td>
<td>146-3</td>
</tr>
<tr>
<td>146.4.3</td>
<td>CLEAR_CONTEXT Procedure</td>
<td>146-4</td>
</tr>
<tr>
<td>146.4.4</td>
<td>CLEAR_IDENTIFIER Procedure</td>
<td>146-4</td>
</tr>
<tr>
<td>146.4.5</td>
<td>CLOSE_DATABASE_LINK Procedure</td>
<td>146-5</td>
</tr>
<tr>
<td>146.4.6</td>
<td>DELETE_SQL_CONNECTION_TEST Procedure</td>
<td>146-5</td>
</tr>
<tr>
<td>146.4.7</td>
<td>DISABLE_CONNECTION_TEST Procedure</td>
<td>146-6</td>
</tr>
<tr>
<td>146.4.8</td>
<td>ENABLE_CONNECTION_TEST Procedure</td>
<td>146-6</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>146.4.9</td>
<td>FREE_UNUSED_USER_MEMORY Procedure</td>
<td>146-7</td>
</tr>
<tr>
<td>146.4.10</td>
<td>GET_PACKAGE_MEMORY_UTILIZATION Procedure</td>
<td>146-9</td>
</tr>
<tr>
<td>146.4.11</td>
<td>IS_ROLE_ENABLED Function</td>
<td>146-10</td>
</tr>
<tr>
<td>146.4.12</td>
<td>IS_SESSION_ALIVE Function</td>
<td>146-10</td>
</tr>
<tr>
<td>146.4.13</td>
<td>LIST_CONTEXT Procedures</td>
<td>146-11</td>
</tr>
<tr>
<td>146.4.14</td>
<td>MODIFY_PACKAGE_STATE Procedure</td>
<td>146-12</td>
</tr>
<tr>
<td>146.4.15</td>
<td>SESSION_TRACE_DISABLE Procedure</td>
<td>146-16</td>
</tr>
<tr>
<td>146.4.16</td>
<td>SESSION_TRACE_ENABLE Procedure</td>
<td>146-16</td>
</tr>
<tr>
<td>146.4.17</td>
<td>RESET_PACKAGE Procedure</td>
<td>146-16</td>
</tr>
<tr>
<td>146.4.18</td>
<td>SET_CONTEXT Procedure</td>
<td>146-18</td>
</tr>
<tr>
<td>146.4.19</td>
<td>SET_EDITION_DEFERRED Procedure</td>
<td>146-19</td>
</tr>
<tr>
<td>146.4.20</td>
<td>SET_IDENTIFIER Procedure</td>
<td>146-19</td>
</tr>
<tr>
<td>146.4.21</td>
<td>SET_NLS Procedure</td>
<td>146-20</td>
</tr>
<tr>
<td>146.4.22</td>
<td>SET_ROLE Procedure</td>
<td>146-20</td>
</tr>
<tr>
<td>146.4.23</td>
<td>SET_SQL_TRACE Procedure</td>
<td>146-21</td>
</tr>
<tr>
<td>146.4.24</td>
<td>SLEEP Procedure</td>
<td>146-21</td>
</tr>
<tr>
<td>146.4.25</td>
<td>SWITCH_CURRENT_CONSUMER_GROUP Procedure</td>
<td>146-22</td>
</tr>
<tr>
<td>146.4.26</td>
<td>UNIQUE_SESSION_ID Function</td>
<td>146-24</td>
</tr>
</tbody>
</table>

### 147 DBMS_SFW_ACL_ADMIN

- DBMS_SFW_ACL_ADMIN Security Model (147-1)
- DBMS_SFW_ACL_ADMIN Operational Notes (147-1)
- DBMS_SFW_ACL_ADMIN Examples (147-2)
- Summary of DBMS_SFW_ACL_ADMIN Subprograms (147-3)
  - COMMIT_ACL Procedure (147-3)
  - IP_ADD_ACE Procedure (147-3)
  - IP_ADD_PDB_ACE Procedure (147-4)
  - IP_REMOVE_ACE Procedure (147-4)
  - IP_REMOVE_ACL Procedure (147-5)
  - IP_REMOVE_PDB_ACE Procedure (147-5)
  - IP_REMOVE_PDB_ACL Procedure (147-6)

### 148 DBMS_SODA

- DBMS_SODA Security Model (148-1)
- Summary of DBMS_SODA Subprograms (148-1)
  - CREATE_COLLECTION Function (148-2)
  - DROP_COLLECTION Function (148-3)
  - LIST_COLLECTION_NAMES Function (148-3)
149 DBMS_SHARED_POOL

149.1 DBMS_SHARED_POOL Overview 149-1
149.2 DBMS_SHARED_POOL Operational Notes 149-1
149.3 Summary of DBMS_SHARED_POOL Subprograms 149-1
  149.3.1 ABORTED_REQUEST_THRESHOLD Procedure 149-2
  149.3.2 KEEP Procedure 149-2
  149.3.3 MARKHOT Procedure 149-4
  149.3.4 PURGE Procedure 149-5
  149.3.5 SIZES Procedure 149-6
  149.3.6 UNKEEP Procedure 149-7
  149.3.7 UNMARKHOT Procedure 149-8

150 DBMS_SPACE

150.1 DBMS_SPACE Security Model 150-1
150.2 DBMS_SPACE Data Structures 150-1
  150.2.1 DBMS_SPACE CREATE_TABLE_COST_COLINFO Object Type 150-1
  150.2.2 DBMS_SPACE ASA_RECO_ROW Record Type 150-2
  150.2.3 DBMS_SPACE ASA_RECO_ROW_TB Table Type 150-3
150.3 Summary of DBMS_SPACE Subprograms 150-3
  150.3.1 DBMS_SPACE ASA_RECOMMENDATIONS Function 150-4
  150.3.2 CREATE_INDEX_COST Procedure 150-5
  150.3.3 CREATE_TABLE_COST Procedures 150-6
  150.3.4 FREE_BLOCKS Procedure 150-7
  150.3.5 ISDATAFILEDROPPABLE_NAME Procedure 150-9
  150.3.6 OBJECT_DEPENDENT_SEGMENTS Function 150-9
  150.3.7 OBJECT_GROWTH_TREND Function 150-10
  150.3.8 SPACE_USAGE Procedures 150-12
  150.3.9 UNUSED_SPACE Procedure 150-15

151 DBMS_SPACE_ADMIN

151.1 DBMS_SPACE_ADMIN Security Model 151-1
151.2 DBMS_SPACE_ADMIN Constants 151-1
151.3 DBMS_SPACE_ADMIN Operational Notes 151-3
151.4 Summary of DBMS_SPACE_ADMIN Subprograms 151-3
  151.4.1 ASSM_SEGMENT_VERIFY Procedure 151-4
  151.4.2 ASSM_TABLESPACE_VERIFY Procedure 151-6
  151.4.3 DROP_EMPTY_SEGMENTS Procedure 151-7
151.4.4 GET_SEGADV_ATTRIB Procedure
151.4.5 MATERIALIZE_DEFERRED_SEGMENTS Procedure
151.4.6 SEGMENT_CORRUPT Procedure
151.4.7 SEGMENT_DROP_CORRUPT Procedure
151.4.8 SEGMENT_DUMP Procedure
151.4.9 SEGMENT_VERIFY Procedure
151.4.10 SET_SEGADV_ATTRIB Procedure
151.4.11 TABLESPACE_FIX_BITMAPS Procedure
151.4.12 TABLESPACE_FIX_SEGMENT_STATES Procedure
151.4.13 TABLESPACE_MIGRATE_FROM_LOCAL Procedure
151.4.14 TABLESPACE_MIGRATE_TO_LOCAL Procedure
151.4.15 TABLESPACE_REBUILD_BITMAPS Procedure
151.4.16 TABLESPACE_REBUILD_QUOTAS Procedure
151.4.17 TABLESPACE_RELOCATE_BITMAPS Procedure
151.4.18 TABLESPACE_VERIFY Procedure

152 DBMS_SPD

152.1 DBMS_SPD Overview
152.2 DBMS_SPD Security Model
152.3 DBMS_SPD Views
152.4 Summary of DBMS_SPD Subprograms
  152.4.1 ALTER_SQL_PLAN_DIRECTIVE Procedure
  152.4.2 CREATE_STGTAB_DIRECTIVE Procedure
  152.4.3 DROP_SQL_PLAN_DIRECTIVE Procedure
  152.4.4 FLUSH_SQL_PLAN_DIRECTIVE Procedure
  152.4.5 GET_PREFS Function
  152.4.6 PACK_STGTAB_DIRECTIVE Function
  152.4.7 SET_PREFS Procedure
  152.4.8 UNPACK_STGTAB_DIRECTIVE Function

153 DBMS_SPM

153.1 DBMS_SPM Overview
153.2 DBMS_SPM Security Model
153.3 DBMS_SPM Constants
153.4 DBMS_SPM Examples
153.5 DBMS_SPM Data Structures
  153.5.1 DBMS_SPM NAMELIST Table Type
153.6 Summary of DBMS_SPM Subprograms
  153.6.1 ACCEPT_SQL_PLAN_BASELINE Procedure
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>153.6.2</td>
<td>ALTER_SQL_PLAN_BASELINE Function</td>
<td>153-5</td>
</tr>
<tr>
<td>153.6.3</td>
<td>CANCEL_EVOLVE_TASK Procedure</td>
<td>153-6</td>
</tr>
<tr>
<td>153.6.4</td>
<td>CONFIGURE Procedure</td>
<td>153-6</td>
</tr>
<tr>
<td>153.6.5</td>
<td>CREATE_EVOLVE_TASK Function</td>
<td>153-9</td>
</tr>
<tr>
<td>153.6.6</td>
<td>CREATE_STGTAB_BASELINE Procedure</td>
<td>153-10</td>
</tr>
<tr>
<td>153.6.7</td>
<td>DROP_EVOLVE_TASK Procedure</td>
<td>153-11</td>
</tr>
<tr>
<td>153.6.8</td>
<td>DROP_SQL_PLAN_BASELINE Function</td>
<td>153-11</td>
</tr>
<tr>
<td>153.6.9</td>
<td>EVOLVE_SQL_PLAN_BASELINE Function</td>
<td>153-12</td>
</tr>
<tr>
<td>153.6.10</td>
<td>EXECUTE_EVOLVE_TASK Function</td>
<td>153-14</td>
</tr>
<tr>
<td>153.6.11</td>
<td>IMPLEMENT_EVOLVE_TASK Function</td>
<td>153-14</td>
</tr>
<tr>
<td>153.6.12</td>
<td>INTERRUPT_EVOLVE_TASK Procedure</td>
<td>153-15</td>
</tr>
<tr>
<td>153.6.13</td>
<td>LOAD_PLANS_FROM_AWR Function</td>
<td>153-15</td>
</tr>
<tr>
<td>153.6.14</td>
<td>LOAD_PLANS_FROM_CURSOR_CACHE Functions</td>
<td>153-16</td>
</tr>
<tr>
<td>153.6.15</td>
<td>LOAD_PLANS_FROM_SQLSET Function</td>
<td>153-18</td>
</tr>
<tr>
<td>153.6.16</td>
<td>MIGRATE_STORED_OUTLINE Functions</td>
<td>153-19</td>
</tr>
<tr>
<td>153.6.17</td>
<td>PACK_STGTAB_BASELINE Function</td>
<td>153-21</td>
</tr>
<tr>
<td>153.6.18</td>
<td>RESET_EVOLVE_TASK Procedure</td>
<td>153-22</td>
</tr>
<tr>
<td>153.6.19</td>
<td>RESUME_EVOLVE_TASK Procedure</td>
<td>153-22</td>
</tr>
<tr>
<td>153.6.20</td>
<td>REPORT_AUTO_EVOLVE_TASK Function</td>
<td>153-22</td>
</tr>
<tr>
<td>153.6.21</td>
<td>REPORT_EVOLVE_TASK Function</td>
<td>153-23</td>
</tr>
<tr>
<td>153.6.22</td>
<td>SET_EVOLVE_TASK_PARAMETER Procedure</td>
<td>153-24</td>
</tr>
<tr>
<td>153.6.23</td>
<td>UNPACK_STGTAB_BASELINE Function</td>
<td>153-25</td>
</tr>
</tbody>
</table>

## 154 DBMS_SQL

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>154.1</td>
<td>DBMS_SQL Overview</td>
<td>154-1</td>
</tr>
<tr>
<td>154.2</td>
<td>DBMS_SQL Security Model</td>
<td>154-2</td>
</tr>
<tr>
<td>154.3</td>
<td>DBMS_SQL Constants</td>
<td>154-3</td>
</tr>
<tr>
<td>154.4</td>
<td>DBMS_SQL Operational Notes</td>
<td>154-4</td>
</tr>
<tr>
<td>154.5</td>
<td>DBMS_SQL Execution Flow</td>
<td>154-5</td>
</tr>
<tr>
<td>154.5.1</td>
<td>OPEN_CURSOR</td>
<td>154-5</td>
</tr>
<tr>
<td>154.5.2</td>
<td>PARSE</td>
<td>154-5</td>
</tr>
<tr>
<td>154.5.3</td>
<td>BIND_VARIABLE, BIND_VARIABLE_PKG or BIND_ARRAY</td>
<td>154-7</td>
</tr>
<tr>
<td>154.5.4</td>
<td>DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY</td>
<td>154-7</td>
</tr>
<tr>
<td>154.5.5</td>
<td>EXECUTE</td>
<td>154-7</td>
</tr>
<tr>
<td>154.5.6</td>
<td>FETCH_ROWS or EXECUTE_AND_FETCH</td>
<td>154-8</td>
</tr>
<tr>
<td>154.5.7</td>
<td>VARIABLE_VALUE, VARIABLE_VALUE_PKG, COLUMN_VALUE, or COLUMN_VALUE_LONG</td>
<td>154-8</td>
</tr>
<tr>
<td>154.5.8</td>
<td>CLOSE_CURSOR</td>
<td>154-8</td>
</tr>
<tr>
<td>154.6</td>
<td>DBMS_SQL Exceptions</td>
<td>154-9</td>
</tr>
<tr>
<td>154.7</td>
<td>DBMS_SQL Examples</td>
<td>154-9</td>
</tr>
<tr>
<td>154.8</td>
<td>DBMS_SQL Data Structures</td>
<td>154-15</td>
</tr>
<tr>
<td>154.8.1</td>
<td>DBMS_SQL DESC_REC Record Type</td>
<td>154-16</td>
</tr>
<tr>
<td>154.8.2</td>
<td>DBMS_SQL DESC_REC2 Record Type</td>
<td>154-17</td>
</tr>
<tr>
<td>154.8.3</td>
<td>DBMS_SQL DESC_REC3 Record Type</td>
<td>154-18</td>
</tr>
<tr>
<td>154.8.4</td>
<td>DBMS_SQL DESC_REC4 Record Type</td>
<td>154-19</td>
</tr>
<tr>
<td>154.8.5</td>
<td>DBMS_SQL BFILE_TABLE Table Type</td>
<td>154-20</td>
</tr>
<tr>
<td>154.8.6</td>
<td>DBMS_SQL BINARY_DOUBLE_TABLE Table Type</td>
<td>154-21</td>
</tr>
<tr>
<td>154.8.7</td>
<td>DBMS_SQL BINARY_FLOAT_TABLE Table Type</td>
<td>154-21</td>
</tr>
<tr>
<td>154.8.8</td>
<td>DBMS_SQL BLOB_TABLE Table Type</td>
<td>154-21</td>
</tr>
<tr>
<td>154.8.9</td>
<td>DBMS_SQL CLOB_TABLE Table Type</td>
<td>154-21</td>
</tr>
<tr>
<td>154.8.10</td>
<td>DBMS_SQL DATE_TABLE Table Type</td>
<td>154-21</td>
</tr>
<tr>
<td>154.8.11</td>
<td>DBMS_SQL DESC_TAB Table Type</td>
<td>154-21</td>
</tr>
<tr>
<td>154.8.12</td>
<td>DBMS_SQL DESC_TAB2 Table Type</td>
<td>154-22</td>
</tr>
<tr>
<td>154.8.13</td>
<td>DBMS_SQL DESC_TAB3 Table Type</td>
<td>154-22</td>
</tr>
<tr>
<td>154.8.14</td>
<td>DBMS_SQL DESC_TAB4 Table Type</td>
<td>154-22</td>
</tr>
<tr>
<td>154.8.15</td>
<td>DBMS_SQL INTERVAL_DAY_TO_SECOND_TABLE Table Type</td>
<td>154-22</td>
</tr>
<tr>
<td>154.8.16</td>
<td>DBMS_SQL INTERVAL_YEAR_TO_MONTH_TABLE Table Type</td>
<td>154-23</td>
</tr>
<tr>
<td>154.8.17</td>
<td>DBMS_SQL NUMBER_TABLE Table Type</td>
<td>154-23</td>
</tr>
<tr>
<td>154.8.18</td>
<td>DBMS_SQL TIME_TABLE Table Type</td>
<td>154-23</td>
</tr>
<tr>
<td>154.8.19</td>
<td>DBMS_SQL TIME_WITH_TIME_ZONE_TABLE Table Type</td>
<td>154-23</td>
</tr>
<tr>
<td>154.8.20</td>
<td>DBMS_SQL TIMESTAMP_TABLE Table Type</td>
<td>154-23</td>
</tr>
<tr>
<td>154.8.21</td>
<td>DBMS_SQL TIMESTAMP_WITH_LTZ_TABLE Table Type</td>
<td>154-23</td>
</tr>
<tr>
<td>154.8.22</td>
<td>DBMS_SQL TIMESTAMP_WITH_TIME_ZONE_TABLE Table Type</td>
<td>154-24</td>
</tr>
<tr>
<td>154.8.23</td>
<td>DBMS_SQL UROWID_TABLE Table Type</td>
<td>154-24</td>
</tr>
<tr>
<td>154.8.24</td>
<td>DBMS_SQL VARCHAR2_TABLE Table Type</td>
<td>154-24</td>
</tr>
<tr>
<td>154.8.25</td>
<td>DBMS_SQL VARCHAR2A Table Type</td>
<td>154-24</td>
</tr>
<tr>
<td>154.8.26</td>
<td>DBMS_SQL VARCHAR2S Table Type</td>
<td>154-24</td>
</tr>
</tbody>
</table>

154.9  Summary of DBMS_SQL Subprograms  

| 154.9.1 | BIND_ARRAY Procedures                  | 154-26 |
| 154.9.2 | BIND_VARIABLE Procedures               | 154-31 |
| 154.9.3 | BIND_VARIABLE_PKG Procedure            | 154-33 |
| 154.9.4 | CLOSE_CURSOR Procedure                 | 154-35 |
| 154.9.5 | COLUMN_VALUE Procedure                 | 154-35 |
| 154.9.6 | COLUMN_VALUE_LONG Procedure            | 154-38 |
| 154.9.7 | DEFINE_ARRAY Procedure                 | 154-38 |
| 154.9.8 | DEFINE_COLUMN Procedures               | 154-43 |
| 154.9.9 | DEFINE_COLUMN_CHAR Procedure           | 154-44 |
| 154.9.10| DEFINE_COLUMN_LONG Procedure           | 154-45 |
| 154.9.11| DEFINE_COLUMN_RAW Procedure            | 154-45 |
| 154.9.12| DEFINE_COLUMN_ROWID Procedure          | 154-46 |
| 154.9.13| DESCRIBE_COLUMNS Procedure             | 154-47 |
| 154.9.14| DESCRIBE_COLUMNS2 Procedure            | 154-48 |
156.6.3 **DEREGISTER_ERROR_TRANSLATION Procedure** 156-7
156.6.4 **DROP_PROFILE Procedure** 156-8
156.6.5 **ENABLE_ERROR_TRANSLATION Procedure** 156-9
156.6.6 **ENABLE_SQL_TRANSLATION Procedure** 156-9
156.6.7 **EXPORT_PROFILE Procedure** 156-10
156.6.8 **IMPORT_PROFILE Procedure** 156-12
156.6.9 **REGISTER_ERROR_TRANSLATION Procedure** 156-13
156.6.10 **REGISTER_SQL_TRANSLATION Procedure** 156-14
156.6.11 **SET_ATTRIBUTE Procedure** 156-15
156.6.12 **SQL_HASH Function** 156-16
156.6.13 **SQL_ID Function** 156-16
156.6.14 **TRANSLATE_ERROR Procedure** 156-17
156.6.15 **TRANSLATE_SQL Procedure** 156-18

157 **DBMS_SQLDIAG**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>157.1</td>
<td>DBMS_SQLDIAG Overview</td>
<td>157-1</td>
</tr>
<tr>
<td>157.2</td>
<td>DBMS_SQLDIAG Security Model</td>
<td>157-3</td>
</tr>
<tr>
<td>157.3</td>
<td>DBMS_SQLDIAG Constants</td>
<td>157-3</td>
</tr>
<tr>
<td>157.4</td>
<td>DBMS_SQLDIAG Examples</td>
<td>157-6</td>
</tr>
<tr>
<td>157.5</td>
<td>Summary of DBMS_SQLDIAG Subprograms</td>
<td>157-6</td>
</tr>
<tr>
<td>157.5.1</td>
<td>ACCEPT_SQL_PATCH Function &amp; Procedure</td>
<td>157-8</td>
</tr>
<tr>
<td>157.5.2</td>
<td>ALTER_SQL_PATCH Procedure</td>
<td>157-9</td>
</tr>
<tr>
<td>157.5.3</td>
<td>CANCEL_DIAGNOSIS_TASK Procedure</td>
<td>157-10</td>
</tr>
<tr>
<td>157.5.4</td>
<td>CREATE_DIAGNOSIS_TASK Functions</td>
<td>157-10</td>
</tr>
<tr>
<td>157.5.5</td>
<td>CREATE_SQL_PATCH Function</td>
<td>157-12</td>
</tr>
<tr>
<td>157.5.6</td>
<td>CREATE_STGTAB_SQLPATCH Procedure</td>
<td>157-13</td>
</tr>
<tr>
<td>157.5.7</td>
<td>DROP_DIAGNOSIS_TASK Procedure</td>
<td>157-14</td>
</tr>
<tr>
<td>157.5.8</td>
<td>DROP_SQL_PATCH Procedure</td>
<td>157-14</td>
</tr>
<tr>
<td>157.5.9</td>
<td>EXECUTE_DIAGNOSIS_TASK Procedure</td>
<td>157-14</td>
</tr>
<tr>
<td>157.5.10</td>
<td>EXPLAIN_SQL_TESTCASE Function</td>
<td>157-15</td>
</tr>
<tr>
<td>157.5.11</td>
<td>EXPORT_SQL_TESTCASE Procedures</td>
<td>157-15</td>
</tr>
<tr>
<td>157.5.12</td>
<td>EXPORT_SQL_TESTCASE_DIR_BY_INC Function</td>
<td>157-19</td>
</tr>
<tr>
<td>157.5.13</td>
<td>EXPORT_SQL_TESTCASE_DIR_BY_TXT Function</td>
<td>157-20</td>
</tr>
<tr>
<td>157.5.14</td>
<td>GET_FIX_CONTROL Function</td>
<td>157-21</td>
</tr>
<tr>
<td>157.5.15</td>
<td>GET_SQL Function</td>
<td>157-22</td>
</tr>
<tr>
<td>157.5.16</td>
<td>IMPORT_SQL_TESTCASE Procedures</td>
<td>157-22</td>
</tr>
<tr>
<td>157.5.17</td>
<td>INCIDENTID_2_SQL Procedure</td>
<td>157-24</td>
</tr>
<tr>
<td>157.5.18</td>
<td>INTERRUPT_DIAGNOSIS_TASK Procedure</td>
<td>157-25</td>
</tr>
<tr>
<td>157.5.19</td>
<td>LOAD_SQLSET_FROM_TCB Function</td>
<td>157-25</td>
</tr>
<tr>
<td>157.5.20</td>
<td>PACK_STGTAB_SQLPATCH Procedure</td>
<td>157-26</td>
</tr>
</tbody>
</table>
157.5.21  REPLAY_SQL_TESTCASE Function 157-26
157.5.22  REPORT_DIAGNOSIS_TASK Function 157-28
157.5.23  RESET_DIAGNOSIS_TASK Procedure 157-29
157.5.24  RESUME_DIAGNOSIS_TASK Procedure 157-29
157.5.25  SET_DIAGNOSIS_TASK_PARAMETER Procedure 157-30
157.5.26  UNPACK_STGTAB_SQLPATCH Procedure 157-30

158  DBMS_SQLPA

158.1  DBMS_SQLPA Overview 158-1
158.2  DBMS_SQLPA Security Model 158-1
158.3  Summary of DBMS_SQLPA Subprograms 158-1
  158.3.1  CANCEL_ANALYSIS_TASK Procedure 158-2
  158.3.2  CREATE_ANALYSIS_TASK Functions 158-2
  158.3.3  DROP_ANALYSIS_TASK Procedure 158-5
  158.3.4  EXECUTE_ANALYSIS_TASK Function & Procedure 158-5
  158.3.5  INTERRUPT_ANALYSIS_TASK Procedure 158-8
  158.3.6  REPORT_ANALYSIS_TASK Function 158-8
  158.3.7  RESET_ANALYSIS_TASK Procedure 158-10
  158.3.8  RESUME_ANALYSIS_TASK Procedure 158-10
  158.3.9  SET_ANALYSIS_TASK_PARAMETER Procedures 158-11
  158.3.10 SET_ANALYSIS_DEFAULT_PARAMETER Procedures 158-15

159  DBMS_SQLSET

159.1  DBMS_SQLSET Overview 159-1
159.2  DBMS_SQLSET Security Model 159-2
159.3  DBMS_SQLSET Data Structures 159-2
  159.3.1  SQLSET_ROW Object Type 159-2
159.4  Summary of DBMS_SQLSET Subprograms 159-4
  159.4.1  ADD_REFERENCE Function 159-6
  159.4.2  CAPTURE_CURSOR_CACHE Procedure 159-7
  159.4.3  CREATE_SQLSET Procedure and Function 159-9
  159.4.4  CREATE_STGTAB Procedure 159-10
  159.4.5  DELETE_SQLSET Procedure 159-11
  159.4.6  DROP_SQLSET Procedure 159-11
  159.4.7  LOAD_SQLSET Procedure 159-12
  159.4.8  PACK_STGTAB Procedure 159-16
  159.4.9  REMAP_STGTAB Procedure 159-18
  159.4.10 REMOVE_REFERENCE Procedure 159-19
  159.4.11 SELECT_CURSOR_CACHE Function 159-20
159.4.12 SELECT_SQL_TRACE Function 159-25
159.4.13 SELECT_SQLPA_TASK Function 159-28
159.4.14 SELECT_SQLSET Function 159-29
159.4.15 SELECT_WORKLOAD_REPOSITORY Function 159-32
159.4.16 UNPACK_STGTAB Procedure 159-34
159.4.17 UPDATE_SQLSET Procedures 159-35

160 DBMS_SQLTUNE

160.1 DBMS_SQLTUNE Overview 160-1
160.2 DBMS_SQLTUNE Security Model 160-4
160.3 DBMS_SQLTUNE Data Structures 160-5
   160.3.1 SQLSET_ROW Object Type 160-5
160.4 DBMS_SQLTUNE Subprogram Groups 160-7
   160.4.1 DBMS_SQLTUNE SQL Tuning Advisor Subprograms 160-7
   160.4.2 DBMS_SQLTUNE SQL Profile Subprograms 160-8
   160.4.3 DBMS_SQLTUNE SQL Tuning Set Subprograms 160-8
   160.4.4 DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms 160-9
   160.4.5 DBMS_SQLTUNE SQL Performance Reporting Subprograms 160-10
160.5 Summary of DBMS_SQLTUNE Subprograms 160-10
   160.5.1 ACCEPT_ALL_SQL_PROFILES Procedure 160-13
   160.5.2 ACCEPT_SQL_PROFILE Procedure and Function 160-15
   160.5.3 ADD_SQLSET_REFERENCE Function 160-20
   160.5.4 ALTER_SQL_PROFILE Procedure 160-21
   160.5.5 CANCEL_TUNING_TASK Procedure 160-22
   160.5.6 CAPTURE_CURSOR_CACHE_SQLSET Procedure 160-23
   160.5.7 CREATE_SQL_PLAN_BASELINE Procedure 160-25
   160.5.8 CREATE_SQLSET Procedure and Function 160-26
   160.5.9 CREATE_STGTAB_SQLPROF Procedure 160-27
   160.5.10 CREATE_STGTAB_SQLSET Procedure 160-28
   160.5.11 CREATE_TUNING_TASK Functions 160-29
   160.5.12 DELETE_SQLSET Procedure 160-35
   160.5.13 DROP_SQL_PROFILE Procedure 160-36
   160.5.14 DROP_SQLSET Procedure 160-37
   160.5.15 DROP_TUNING_TASK Procedure 160-37
   160.5.16 EXECUTE_TUNING_TASK Function and Procedure 160-38
   160.5.17 IMPLEMENT_TUNING_TASK Procedure 160-39
   160.5.18 INTERRUPT_TUNING_TASK Procedure 160-41
   160.5.19 LOAD_SQLSET Procedure 160-41
   160.5.20 PACK_STGTAB_SQLPROF Procedure 160-45
   160.5.21 PACK_STGTAB_SQLSET Procedure 160-47
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>160.5.22</td>
<td>REMAP_STGTAB_SQLPROF Procedure</td>
</tr>
<tr>
<td>160.5.23</td>
<td>REMAP_STGTAB_SQLSET Procedure</td>
</tr>
<tr>
<td>160.5.24</td>
<td>REMOVE_SQLSET_REFERENCE Procedure</td>
</tr>
<tr>
<td>160.5.25</td>
<td>REPORT_AUTO_TUNING_TASK Function</td>
</tr>
<tr>
<td>160.5.26</td>
<td>REPORT_SQL_DETAIL Function</td>
</tr>
<tr>
<td>160.5.27</td>
<td>REPORT_SQL_MONITOR Function</td>
</tr>
<tr>
<td>160.5.28</td>
<td>REPORT_SQL_MONITOR_LIST Function</td>
</tr>
<tr>
<td>160.5.29</td>
<td>REPORT_TUNING_TASK Function</td>
</tr>
<tr>
<td>160.5.30</td>
<td>REPORT_TUNING_TASK_XML Function</td>
</tr>
<tr>
<td>160.5.31</td>
<td>RESET_TUNING_TASK Procedure</td>
</tr>
<tr>
<td>160.5.32</td>
<td>RESUME_TUNING_TASK Procedure</td>
</tr>
<tr>
<td>160.5.33</td>
<td>SCHEDULE_TUNING_TASK Function</td>
</tr>
<tr>
<td>160.5.34</td>
<td>SCRIPT_TUNING_TASK Function</td>
</tr>
<tr>
<td>160.5.35</td>
<td>SELECT_CURSOR_CACHE Function</td>
</tr>
<tr>
<td>160.5.36</td>
<td>SELECT_SQL_TRACE Function</td>
</tr>
<tr>
<td>160.5.37</td>
<td>SELECT_SQLPA_TASK Function</td>
</tr>
<tr>
<td>160.5.38</td>
<td>SELECT_SQLSET Function</td>
</tr>
<tr>
<td>160.5.39</td>
<td>SELECT_WORKLOAD_REPOSITORY Function</td>
</tr>
<tr>
<td>160.5.40</td>
<td>SET_TUNING_TASK_PARAMETER Procedures</td>
</tr>
<tr>
<td>160.5.41</td>
<td>SQLTEXT_TO_SIGNATURE Function</td>
</tr>
<tr>
<td>160.5.42</td>
<td>UNPACK_STGTAB_SQLPROF Procedure</td>
</tr>
<tr>
<td>160.5.43</td>
<td>UNPACK_STGTAB_SQLSET Procedure</td>
</tr>
<tr>
<td>160.5.44</td>
<td>UPDATE_SQLSET Procedures</td>
</tr>
</tbody>
</table>

### DBMS_STAT_FUNCS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>161.1.1</td>
<td>EXPONENTIAL_DIST_FIT Procedure</td>
</tr>
<tr>
<td>161.1.2</td>
<td>NORMAL_DIST_FIT Procedure</td>
</tr>
<tr>
<td>161.1.3</td>
<td>POISSON_DIST_FIT Procedure</td>
</tr>
<tr>
<td>161.1.4</td>
<td>SUMMARY Procedure</td>
</tr>
<tr>
<td>161.1.5</td>
<td>UNIFORM_DIST_FIT Procedure</td>
</tr>
<tr>
<td>161.1.6</td>
<td>WEIBULL_DIST_FIT Procedure</td>
</tr>
</tbody>
</table>

### DBMS_STATS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>162.1</td>
<td>DBMS_STATS Overview</td>
</tr>
<tr>
<td>162.2</td>
<td>DBMS_STATS Deprecated Subprograms</td>
</tr>
<tr>
<td>162.3</td>
<td>DBMS_STATS Types</td>
</tr>
<tr>
<td>162.4</td>
<td>DBMS_STATS Constants</td>
</tr>
<tr>
<td>162.5</td>
<td>DBMS_STATS Operational Notes</td>
</tr>
</tbody>
</table>
162.6 DBMS_STATS Data Structures

162.6.1 DBMS_STATS STAT_REC Record Type

162.7 Summary of DBMS_STATS Subprograms

162.7.1 ALTER_STATS_HISTORY_RETENTION Procedure
162.7.2 CANCEL_ADVISOR_TASK Procedure
162.7.3 CONFIGURE_ADVISOR_FILTER Function
162.7.4 CONFIGURE_ADVISOR_OBJ_FILTER Function
162.7.5 CONFIGURE_ADVISOR_OPR_FILTER Functions
162.7.6 CONFIGURE_ADVISOR_RULE_FILTER Function
162.7.7 CONVERT_RAW_VALUE Procedures
162.7.8 CONVERT_RAW_VALUE_NVARCHAR Procedure
162.7.9 CONVERT_RAW_VALUE_ROWID Procedure
162.7.10 COPY_TABLE_STATS Procedure
162.7.11 CREATE_ADVISOR_TASK Function
162.7.12 CREATE_EXTENDED_STATS Function
162.7.13 CREATE_STAT_TABLE Procedure
162.7.14 DELETE_COLUMN_STATS Procedure
162.7.15 DELETE_DATABASE_PREFS Procedure
162.7.16 DELETE_DATABASE_STATS Procedure
162.7.17 DELETE_DICTIONARY_STATS Procedure
162.7.18 DELETE_FIXED_OBJECTS_STATS Procedure
162.7.19 DELETE_INDEX_STATS Procedure
162.7.20 DELETE_PENDING_STATS Procedure
162.7.21 DELETE_PROCESSING_RATE Procedure
162.7.22 DELETE_SCHEMA_PREFS Procedure
162.7.23 DELETE_SCHEMA_STATS Procedure
162.7.24 DELETE_SYSTEM_STATS Procedure
162.7.25 DELETE_TABLE_PREFS Procedure
162.7.26 DELETE_TABLE_STATS Procedure
162.7.27 DIFF_TABLE_STATS_IN_HISTORY Function
162.7.28 DIFF_TABLE_STATS_IN_PENDING Function
162.7.29 DIFF_TABLE_STATS_IN_STATTAB Function
162.7.30 DROP_ADVISOR_TASK Procedure
162.7.31 DROP_EXTENDED_STATS Procedure
162.7.32 DROP_STAT_TABLE Procedure
162.7.33 EXECUTE_ADVISOR_TASK Function
162.7.34 EXPORT_COLUMN_STATS Procedure
162.7.35 EXPORT_DATABASE_PREFS Procedure
162.7.36 EXPORT_DATABASE_STATS Procedure
162.7.37 EXPORT_DICTIONARY_STATS Procedure
162.7.38 EXPORT_FIXED_OBJECTS_STATS Procedure
<table>
<thead>
<tr>
<th>Procedure/Macro</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORT_INDEX_STATS Procedure</td>
<td>162-76</td>
</tr>
<tr>
<td>EXPORT_PENDING_STATS Procedure</td>
<td>162-77</td>
</tr>
<tr>
<td>EXPORT_SCHEMA_PREFS Procedure</td>
<td>162-78</td>
</tr>
<tr>
<td>EXPORT_SCHEMA_STATS Procedure</td>
<td>162-79</td>
</tr>
<tr>
<td>EXPORT_SYSTEM_STATS Procedure</td>
<td>162-80</td>
</tr>
<tr>
<td>EXPORT_TABLE_PREFS Procedure</td>
<td>162-81</td>
</tr>
<tr>
<td>EXPORT_TABLE_STATS Procedure</td>
<td>162-81</td>
</tr>
<tr>
<td>FLUSH_DATABASE_MONITORING_INFO Procedure</td>
<td>162-83</td>
</tr>
<tr>
<td>GATHER_DATABASE_STATS Procedures</td>
<td>162-83</td>
</tr>
<tr>
<td>GATHER_DICTIONARY_STATS Procedure</td>
<td>162-88</td>
</tr>
<tr>
<td>GATHER_FIXED_OBJECTS_STATS Procedure</td>
<td>162-92</td>
</tr>
<tr>
<td>GATHER_INDEX_STATS Procedure</td>
<td>162-93</td>
</tr>
<tr>
<td>GATHER_PROCESSING_RATE Procedure</td>
<td>162-95</td>
</tr>
<tr>
<td>GATHER_SCHEMA_STATS Procedures</td>
<td>162-95</td>
</tr>
<tr>
<td>GATHER_SYSTEM_STATS Procedure</td>
<td>162-100</td>
</tr>
<tr>
<td>GATHER_TABLE_STATS Procedure</td>
<td>162-103</td>
</tr>
<tr>
<td>GENERATE_STATS Procedure</td>
<td>162-107</td>
</tr>
<tr>
<td>GET_ADVISOR_OPR_FILTER Procedure</td>
<td>162-108</td>
</tr>
<tr>
<td>GET_ADVISOR_RECS Function</td>
<td>162-109</td>
</tr>
<tr>
<td>GET_COLUMN_STATS Procedures</td>
<td>162-110</td>
</tr>
<tr>
<td>GET_INDEX_STATS Procedures</td>
<td>162-112</td>
</tr>
<tr>
<td>GET_PARAM Function</td>
<td>162-114</td>
</tr>
<tr>
<td>GET_PREFS Function</td>
<td>162-115</td>
</tr>
<tr>
<td>GET_STATS_HISTORY_AVAILABILITY Function</td>
<td>162-124</td>
</tr>
<tr>
<td>GET_STATS_HISTORY_RETENTION Function</td>
<td>162-124</td>
</tr>
<tr>
<td>GET_SYSTEM_STATS Procedure</td>
<td>162-124</td>
</tr>
<tr>
<td>GET_TABLE_STATS Procedure</td>
<td>162-126</td>
</tr>
<tr>
<td>IMPLEMENT_ADVISOR_TASK Function</td>
<td>162-128</td>
</tr>
<tr>
<td>IMPORT_COLUMN_STATS Procedure</td>
<td>162-130</td>
</tr>
<tr>
<td>IMPORT_DATABASE_PREFS Procedure</td>
<td>162-131</td>
</tr>
<tr>
<td>IMPORT_DATABASE_STATS Procedure</td>
<td>162-132</td>
</tr>
<tr>
<td>IMPORT_DICTIONARY_STATS Procedure</td>
<td>162-133</td>
</tr>
<tr>
<td>IMPORT_FIXED_OBJECTS_STATS Procedure</td>
<td>162-135</td>
</tr>
<tr>
<td>IMPORT_INDEX_STATS Procedure</td>
<td>162-136</td>
</tr>
<tr>
<td>IMPORT_SCHEMA_PREFS Procedure</td>
<td>162-137</td>
</tr>
<tr>
<td>IMPORT_SCHEMA_STATS Procedure</td>
<td>162-138</td>
</tr>
<tr>
<td>IMPORT_SYSTEM_STATS Procedure</td>
<td>162-139</td>
</tr>
<tr>
<td>IMPORT_TABLE_PREFS Procedure</td>
<td>162-140</td>
</tr>
<tr>
<td>IMPORT_TABLE_STATS Procedure</td>
<td>162-140</td>
</tr>
<tr>
<td>INTERRUPT_ADVISOR_TASK Procedure</td>
<td>162-142</td>
</tr>
<tr>
<td>LOCK_PARTITION_STATS Procedure</td>
<td>162-143</td>
</tr>
<tr>
<td>162.7.80</td>
<td>LOCK_SCHEMA_STATS Procedure</td>
</tr>
<tr>
<td>162.7.81</td>
<td>LOCK_TABLE_STATS Procedure</td>
</tr>
<tr>
<td>162.7.82</td>
<td>MERGE_COL_USAGE Procedure</td>
</tr>
<tr>
<td>162.7.83</td>
<td>PREPARE_COLUMN_VALUES Procedures</td>
</tr>
<tr>
<td>162.7.84</td>
<td>PREPARE_COLUMN_VALUES_NVARCHAR Procedure</td>
</tr>
<tr>
<td>162.7.85</td>
<td>PREPARE_COLUMN_VALUES_ROWID Procedure</td>
</tr>
<tr>
<td>162.7.86</td>
<td>PUBLISH_PENDING_STATS Procedure</td>
</tr>
<tr>
<td>162.7.87</td>
<td>PURGE_STATS Procedure</td>
</tr>
<tr>
<td>162.7.88</td>
<td>REMAP_STAT_TABLE Procedure</td>
</tr>
<tr>
<td>162.7.89</td>
<td>REPORT_ADVISOR_TASK Function</td>
</tr>
<tr>
<td>162.7.90</td>
<td>REPORT_COL_USAGE Function</td>
</tr>
<tr>
<td>162.7.91</td>
<td>REPORT_GATHER_AUTO_STATS Function</td>
</tr>
<tr>
<td>162.7.92</td>
<td>REPORT_GATHER_DATABASE_STATS Functions</td>
</tr>
<tr>
<td>162.7.93</td>
<td>REPORT_GATHER_DICTIONARY_STATS Functions</td>
</tr>
<tr>
<td>162.7.94</td>
<td>REPORT_GATHER_FIXED_OBJ_STATS Function</td>
</tr>
<tr>
<td>162.7.95</td>
<td>REPORT_GATHER_SCHEMA_STATS Functions</td>
</tr>
<tr>
<td>162.7.96</td>
<td>REPORT_GATHER_TABLE_STATS Function</td>
</tr>
<tr>
<td>162.7.97</td>
<td>REPORT_SINGLE_STATS_OPERATION Function</td>
</tr>
<tr>
<td>162.7.98</td>
<td>REPORT_STATS_OPERATIONS Function</td>
</tr>
<tr>
<td>162.7.99</td>
<td>RESET_ADVISOR_TASK Procedure</td>
</tr>
<tr>
<td>162.7.100</td>
<td>RESET_COL_USAGE Procedure</td>
</tr>
<tr>
<td>162.7.101</td>
<td>RESET_GLOBAL_PREF_DEFAULTS Procedure</td>
</tr>
<tr>
<td>162.7.102</td>
<td>RESET_PARAM_DEFAULTS Procedure</td>
</tr>
<tr>
<td>162.7.103</td>
<td>RESTORE_DATABASE_STATS Procedure</td>
</tr>
<tr>
<td>162.7.104</td>
<td>RESTORE_DICTIONARY_STATS Procedure</td>
</tr>
<tr>
<td>162.7.105</td>
<td>RESTORE_FIXED_OBJECTS_STATS Procedure</td>
</tr>
<tr>
<td>162.7.106</td>
<td>RESTORE_SCHEMA_STATS Procedure</td>
</tr>
<tr>
<td>162.7.107</td>
<td>RESTORE_SYSTEM_STATS Procedure</td>
</tr>
<tr>
<td>162.7.108</td>
<td>RESTORE_TABLE_STATS Procedure</td>
</tr>
<tr>
<td>162.7.109</td>
<td>RESUME_ADVISOR_TASK Procedure</td>
</tr>
<tr>
<td>162.7.110</td>
<td>SCRIPT_ADVISOR_TASK Function</td>
</tr>
<tr>
<td>162.7.111</td>
<td>SEED_COL_USAGE Procedure</td>
</tr>
<tr>
<td>162.7.112</td>
<td>SET_ADVISOR_TASK_PARAMETER Procedure</td>
</tr>
<tr>
<td>162.7.113</td>
<td>SET_COLUMN_STATS Procedures</td>
</tr>
<tr>
<td>162.7.114</td>
<td>SET_DATABASE_PREFS Procedure</td>
</tr>
<tr>
<td>162.7.115</td>
<td>SET_GLOBAL_PREFS Procedure</td>
</tr>
<tr>
<td>162.7.116</td>
<td>SET_INDEX_STATS Procedures</td>
</tr>
<tr>
<td>162.7.117</td>
<td>SET_PARAM Procedure</td>
</tr>
<tr>
<td>162.7.118</td>
<td>SET_PROCESSING_RATE Procedure</td>
</tr>
<tr>
<td>162.7.119</td>
<td>SET_SCHEMA_PREFS Procedure</td>
</tr>
<tr>
<td>162.7.120</td>
<td>SET_SYSTEM_STATS Procedure</td>
</tr>
</tbody>
</table>
162.7.121  SET_TABLE_PREFS Procedure  
162.7.122  SET_TABLE_STATS Procedure  
162.7.123  SHOW_EXTENDED_STATS_NAME Function  
162.7.124  TRANSFER_STATS Procedure  
162.7.125  UNLOCK_PARTITION_STATS Procedure  
162.7.126  UNLOCK_SCHEMA_STATS Procedure  
162.7.127  UNLOCK_TABLE_STATS Procedure  
162.7.128  UPGRADE_STAT_TABLE Procedure

163  DBMS_STORAGE_MAP

163.1  DBMS_STORAGE_MAP Overview  
163.2  DBMS_STORAGE_MAP Operational Notes  
163.3  Summary of DBMS_STORAGE_MAP Subprograms  
   163.3.1  DROP_ALL Function  
   163.3.2  DROP_ELEMENT Function  
   163.3.3  DROP_FILE Function  
   163.3.4  LOCK_MAP Procedure  
   163.3.5  MAP_ALL Function  
   163.3.6  MAP_ELEMENT Function  
   163.3.7  MAP_FILE Function  
   163.3.8  MAP_OBJECT Function  
   163.3.9  RESTORE Function  
   163.3.10  SAVE Function  
   163.3.11  UNLOCK_MAP Procedure

164  DBMS_STREAMS

164.1  DBMS_STREAMS Overview  
164.2  DBMS_STREAMS Security Model  
164.3  Summary of DBMS_STREAMS Subprograms  
   164.3.1  COMPATIBLE_12_2 Function  
   164.3.2  COMPATIBLE_12_1 Function  
   164.3.3  COMPATIBLE_11_2 Function  
   164.3.4  COMPATIBLE_11_1 Function  
   164.3.5  COMPATIBLE_10_2 Function  
   164.3.6  COMPATIBLE_10_1 Function  
   164.3.7  COMPATIBLE_9_2 Function  
   164.3.8  CONVERT_ANYDATA_TO_LCRDDL Function  
   164.3.9  CONVERT_ANYDATA_TO_LCR_ROW Function  
   164.3.10 CONVERT_LCR_TO_XML Function
164.3.11 CONVERT_XML_TO_LCR Function
164.3.12 GET_INFORMATION Function
164.3.13 GET_STREAMS_NAME Function
164.3.14 GET_STREAMS_TYPE Function
164.3.15 GET_TAG Function
164.3.16 MAX_COMPATIBLE Function
164.3.17 SET_TAG Procedure

165 DBMS_STREAMS_ADM

165.1 DBMS_STREAMS_ADM Overview
165.2 DBMS_STREAMS_ADM Deprecated Subprograms
165.3 DBMS_STREAMS_ADM Security Model
165.4 Procedures That Create Rules for Oracle Streams Clients and XStream Clients
165.5 Procedures That Configure an Oracle Streams Environment
165.6 Summary of DBMS_STREAMS_ADM Subprograms
  165.6.1 ADD_COLUMN Procedure
  165.6.2 ADD_GLOBAL_PROPAGATION_RULES Procedure
  165.6.3 ADD_GLOBAL_RULES Procedure
  165.6.4 ADD_MESSAGE_PROPAGATION_RULE Procedure
  165.6.5 ADD_MESSAGE_RULE Procedure
  165.6.6 ADD_SCHEMA_PROPAGATION_RULES Procedure
  165.6.7 ADD_SCHEMA_RULES Procedure
  165.6.8 ADD_SUBSET_PROPAGATION_RULES Procedure
  165.6.9 ADD_SUBSET_RULES Procedure
  165.6.10 ADD_TABLE_PROPAGATION_RULES Procedure
  165.6.11 ADD_TABLE_RULES Procedure
  165.6.12 CLEANUP_INSTANTIATION_SETUP Procedure
  165.6.13 DELETE_COLUMN Procedure
  165.6.14 GET_MESSAGE_TRACKING Function
  165.6.15 GET_SCN_MAPPING Procedure
  165.6.16 GET_TAG Function
  165.6.17 KEEP_COLUMNS Procedure
  165.6.18 MAINTAIN_CHANGE_TABLE Procedure
  165.6.19 MAINTAIN_GLOBAL Procedure
  165.6.20 MAINTAIN_SCHEMAS Procedure
  165.6.21 MAINTAIN_SIMPLE_TABLESPACE Procedure
  165.6.22 MAINTAIN_SIMPLE_TTS Procedure
  165.6.23 MAINTAIN_TABLES Procedure
  165.6.24 MAINTAIN_TABLESPACES Procedure
  165.6.25 MAINTAIN_TTS Procedure
# DBMS_STREAMS_HANDLER_ADM

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>168.1</td>
<td>DBMS_STREAMS_HANDLER_ADM Overview</td>
<td>168-1</td>
</tr>
<tr>
<td>168.2</td>
<td>DBMS_STREAMS_HANDLER_ADM Security Model</td>
<td>168-2</td>
</tr>
<tr>
<td>168.3</td>
<td>DBMS_STREAMS_HANDLER_ADM Views</td>
<td>168-2</td>
</tr>
<tr>
<td>168.4</td>
<td>DBMS_STREAMS_HANDLER_ADM Operational Notes</td>
<td>168-3</td>
</tr>
<tr>
<td>168.5</td>
<td>Summary of DBMS_STREAMS_HANDLER_ADM Subprograms</td>
<td>168-6</td>
</tr>
<tr>
<td>168.5.1</td>
<td>ADD_STMT_TO_HANDLER Procedure</td>
<td>168-7</td>
</tr>
<tr>
<td>168.5.2</td>
<td>CREATE_STMT_HANDLER Procedure</td>
<td>168-7</td>
</tr>
<tr>
<td>168.5.3</td>
<td>DROP_STMT_HANDLER Procedure</td>
<td>168-8</td>
</tr>
<tr>
<td>168.5.4</td>
<td>REMOVE_STMT_FROM_HANDLER Procedure</td>
<td>168-8</td>
</tr>
</tbody>
</table>

# DBMS_STREAMS_MESSAGING

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>169.1</td>
<td>DBMS_STREAMS_MESSAGING Overview</td>
<td>169-1</td>
</tr>
<tr>
<td>169.2</td>
<td>DBMS_STREAMS_MESSAGING Security Model</td>
<td>169-1</td>
</tr>
<tr>
<td>169.3</td>
<td>Summary of DBMS_STREAMS_MESSAGING Subprograms</td>
<td>169-2</td>
</tr>
<tr>
<td>169.3.1</td>
<td>DEQUEUE Procedure</td>
<td>169-2</td>
</tr>
<tr>
<td>169.3.2</td>
<td>ENQUEUE Procedure</td>
<td>169-4</td>
</tr>
</tbody>
</table>

# DBMS_STREAMS_TABLESPACE_ADM

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>170.1</td>
<td>DBMS_STREAMS_TABLESPACE_ADM Overview</td>
<td>170-1</td>
</tr>
<tr>
<td>170.2</td>
<td>DBMS_STREAMS_TABLESPACE_ADM Security Model</td>
<td>170-2</td>
</tr>
<tr>
<td>170.3</td>
<td>DBMS_STREAMS_TABLESPACE_ADM Data Structures</td>
<td>170-2</td>
</tr>
<tr>
<td>170.3.1</td>
<td>DIRECTORY_OBJECT_SET Table Type</td>
<td>170-3</td>
</tr>
<tr>
<td>170.3.2</td>
<td>FILE Record Type</td>
<td>170-3</td>
</tr>
<tr>
<td>170.3.3</td>
<td>FILE_SET Table Type</td>
<td>170-3</td>
</tr>
<tr>
<td>170.3.4</td>
<td>TABLESPACE_SET Table Type</td>
<td>170-3</td>
</tr>
<tr>
<td>170.4</td>
<td>Summary of DBMS_STREAMS_TABLESPACE_ADM Subprograms</td>
<td>170-3</td>
</tr>
<tr>
<td>170.4.1</td>
<td>ATTACH_SIMPLE_TABLESPACE Procedure</td>
<td>170-4</td>
</tr>
<tr>
<td>170.4.2</td>
<td>ATTACH_TABLESPACES Procedure</td>
<td>170-6</td>
</tr>
<tr>
<td>170.4.3</td>
<td>CLONE_SIMPLE_TABLESPACE Procedure</td>
<td>170-10</td>
</tr>
<tr>
<td>170.4.4</td>
<td>CLONE_TABLESPACES Procedure</td>
<td>170-10</td>
</tr>
<tr>
<td>170.4.5</td>
<td>DETACH_SIMPLE_TABLESPACE Procedure</td>
<td>170-16</td>
</tr>
<tr>
<td>170.4.6</td>
<td>DETACH_TABLESPACES Procedure</td>
<td>170-18</td>
</tr>
<tr>
<td>170.4.7</td>
<td>PULL_SIMPLE_TABLESPACE Procedure</td>
<td>170-21</td>
</tr>
<tr>
<td>170.4.8</td>
<td>PULL_TABLESPACES Procedure</td>
<td>170-24</td>
</tr>
</tbody>
</table>
171  DBMS_SYNC_REFRESH

171.1  DBMS_SYNC_REFRESH Overview 171-1
171.2  DBMS_SYNC_REFRESH Security Model 171-1
171.3  Summary of DBMS_SYNC_REFRESH Subprograms 171-2
    171.3.1  ABORT_REFRESH Procedure 171-2
    171.3.2  ALTER_REFRESH_STATS_RETENTION Procedure 171-3
    171.3.3  CAN_SYNCREF_TABLE Procedure 171-4
    171.3.4  EXECUTE_REFRESH Procedure 171-6
    171.3.5  GET_ALL_GROUP_IDS Function 171-6
    171.3.6  GET_GROUP_ID Function 171-7
    171.3.7  GET_GROUP_ID_LIST Function 171-7
    171.3.8  PREPARE_REFRESH Procedure 171-8
    171.3.9  PREPARE_STAGING_LOG Procedure 171-9
    171.3.10 PURGE_REFRESH_STATS Procedure 171-10
    171.3.11 REGISTER_MVIEWS 171-11
    171.3.12 REGISTER_PARTITION_OPERATION Procedure 171-12
    171.3.13 UNREGISTER_MVIEWS 171-13
    171.3.14 UNREGISTER_PARTITION_OPERATION Procedure 171-13

172  DBMS_TDB

172.1  DBMS_TDB Overview 172-1
172.2  DBMS_TDB Security Model 172-2
172.3  DBMS_TDB Constants 172-2
172.4  DBMS_TDB Views 172-2
172.5  DBMS_TDB Operational Notes 172-2
172.6  Summary of DBMS_TDB Subprograms 172-2
    172.6.1  CHECK_DB Function 172-3
    172.6.2  CHECK_EXTERNAL Function 172-5

173  DBMS_TF

173.1  DBMS_TF Overview 173-1
173.2  DBMS_TF Security Model 173-2
173.3  DBMS_TF Constants 173-2
173.4  DBMS_TF Operational Notes 173-3
    173.4.1  PTF Client Interface 173-3
        173.4.1.1  DESCRIBE Only Polymorphic Table Function 173-5
    173.4.2  PTF Server Side Interface 173-5
    173.4.3  Read Columns 173-6
    173.4.4  Pass-Through Columns 173-7
173.4.5 State Management
   173.4.5.1 CSTORE Compilation State Management
   173.4.5.2 XSTORE Execution State Management
173.4.6 Method Name Overrides
173.4.7 Using the COLUMNS Pseudo-Operator
173.4.8 Query Transformations
173.4.9 Parallel Execution
173.5 DBMS_TF Execution Flow
173.6 DBMS_TF Restrictions
173.7 DBMS_TF Examples
   173.7.1 Echo Polymorphic Table Function Example
173.8 DBMS_TF Data Structures
   173.8.1 Supported Types Collections
   173.8.2 COLUMN_DATA_T Record Type
   173.8.3 COLUMN_METADATA_T Record Type
   173.8.4 COLUMN_T Record Type
   173.8.5 DESCRIBE_T Record Type
   173.8.6 ENV_T Record Type
   173.8.7 PARALLEL_ENV_T Record Type
   173.8.8 TABLE_T Record Type
   173.8.9 COLUMNS_NEW_T Table Type
   173.8.10 COLUMNS_T Table Type
   173.8.11 COLUMNS_WITH_TYPE_T Table Type
   173.8.12 TABLE_COLUMNS_T Table Type
   173.8.13 ROW_SET_T Table Type
   173.8.14 XID_T Subtype
173.9 Summary of DBMS_TF Subprograms
   173.9.1 COLUMN_TYPE_NAME Function
   173.9.2 COL_TO_CHAR Function
   173.9.3 CSTORE_EXISTS Function
   173.9.4 CSTORE_GET Procedure
   173.9.5 GET_COL Procedure
   173.9.6 GET_ENV Function
   173.9.7 GET_ROW_SET Procedure
      173.9.7.1 Stack Polymorphic Table Function Example
   173.9.8 GET_XID Function
   173.9.9 PUT_COL Procedure
      173.9.9.1 Rand_col Polymorphic Table Function Example
   173.9.10 PUT_ROW_SET Procedure
      173.9.10.1 Split Polymorphic Table Function Example
   173.9.11 ROW_REPLICATION Procedure
176.2.9  PURGE_MIXED Procedure 176-6
176.2.10 READ_ONLY Procedure 176-6
176.2.11 READ_WRITE Procedure 176-7
176.2.12 ROLLBACK Procedure 176-7
176.2.13 ROLLBACK_FORCE Procedure 176-7
176.2.14 ROLLBACK_SAVEPOINT Procedure 176-7
176.2.15 SAVEPOINT Procedure 176-8
176.2.16 STEP_ID Function 176-8
176.2.17 USE.Rollback SEGMENT Procedure 176-8

177  DBMS_TRANSFORM

177.1  Summary of DBMS_TRANSFORM Subprograms 177-1
177.1.1 CREATE_TRANSFORMATION Procedure 177-1
177.1.2 DROP_TRANSFORMATION Procedure 177-2
177.1.3 MODIFY_TRANSFORMATION Procedure 177-3

178  DBMS_TSDP_MANAGE

178.1  DBMS_TSDP_MANAGE Overview 178-1
178.2  DBMS_TSDP_MANAGE Security Model 178-2
178.3  Summary of DBMS_TSDP_MANAGE Subprograms 178-2
178.3.1 ADD_SENSITIVE_COLUMN Procedure 178-2
178.3.2 ALTER_SENSITIVE_COLUMN Procedure 178-3
178.3.3 ADD_SENSITIVE_TYPE Procedure 178-4
178.3.4 DROP_SENSITIVE_COLUMN Procedure 178-4
178.3.5 DROP_SENSITIVE_TYPE Procedure 178-5
178.3.6 DROP_SENSITIVE_TYPE_SOURCE Procedure 178-5
178.3.7 IMPORT_DISCOVERY_RESULT Procedure 178-6
178.3.8 IMPORT_SENSITIVE_TYPES Procedure 178-7
178.3.9 REMOVE_DISCOVERY_RESULT Procedure 178-8

179  DBMS_TSDP_PROTECT

179.1  DBMS_TSDP_PROTECT Overview 179-1
179.2  DBMS_TSDP_PROTECT Security Model 179-1
179.3  DBMS_TSDP_PROTECT Constants 179-2
179.4  DBMS_TSDP_PROTECT Data Structures 179-2
179.4.1 FEATURE_OPTIONS Table Type 179-2
179.4.2 POLICY_CONDITIONS Table Type 179-2
179.5  Summary of DBMS_TSDP_PROTECT Subprograms 179-2
179.5.1 ADD_POLICY Procedure 179-3
DBMS_UTILITY

183.1 DBMS_UTILITY Deprecated Subprograms
183.2 DBMS_UTILITY Security Model
183.3 DBMS_UTILITY Constants
183.4 DBMS_UTILITY Exceptions
183.5 DBMS_UTILITY Data Structures
  183.5.1 DBMS_UTILITY INSTANCE_RECORD Record Type
  183.5.2 DBMS_UTILITY DBLINK_ARRAY TABLE Type
  183.5.3 DBMS_UTILITY INDEX_TABLE_TYPE Table Type
  183.5.4 DBMS_UTILITY INSTANCE_TABLE Table Type
  183.5.5 DBMS_UTILITY LNAME_ARRAY Table Type
  183.5.6 DBMS_UTILITY NAME_ARRAY Table Type
  183.5.7 DBMS_UTILITY NUMBER_ARRAY Table Type
  183.5.8 DBMS_UTILITY UNCL_ARRAY Table Type
183.6 Summary of DBMS_UTILITY Subprograms
  183.6.1 ACTIVE_INSTANCES Procedure
  183.6.2 ANALYZE_DATABASE Procedure
  183.6.3 ANALYZE_PART_OBJECT Procedure
  183.6.4 ANALYZE_SCHEMA Procedure
  183.6.5 CANONICALIZE Procedure
  183.6.6 COMMA_TO_TABLE Procedures
  183.6.7 COMPIL_...
183.6.24 GET_TIME Function 183-22
183.6.25 GET_TZ_TRANSITIONS Procedure 183-23
183.6.26 INVALIDATE Procedure 183-23
183.6.27 IS_BIT_SET Function 183-26
183.6.28 IS_CLUSTER_DATABASE Function 183-26
183.6.29 MAKE_DATA_BLOCK_ADDRESS Function 183-26
183.6.30 NAME_RESOLVE Procedure 183-27
183.6.31 NAME_TOKENIZE Procedure 183-28
183.6.32 OLD_CURRENT_SCHEMA Function 183-29
183.6.33 OLD_CURRENT_USER Function 183-29
183.6.34 PORT_STRING Function 183-29
183.6.35 SQLID_TO_SQLHASH Function 183-30
183.6.36 TABLE_TO_COMMA Procedures 183-30
183.6.37 VALIDATE Procedure 183-31
183.6.38 WAIT_ON_PENDING_DML Function 183-32

184 DBMS_WARNING

184.1 DBMS_WARNING Security Model 184-1
184.2 Summary of DBMS_WARNING Subprograms 184-1
  184.2.1 ADD_WARNING_SETTING_CAT Procedure 184-2
  184.2.2 ADD_WARNING_SETTING_NUM Procedure 184-2
  184.2.3 GET_CATEGORY Function 184-3
  184.2.4 GET_WARNING_SETTING_CAT Function 184-4
  184.2.5 GET_WARNING_SETTING_NUM Function 184-4
  184.2.6 GET_WARNING_SETTING_STRING Function 184-4
  184.2.7 SET_WARNING_SETTING_STRING Procedure 184-5

185 DBMS_WM

186 DBMS_WORKLOAD_CAPTURE

186.1 DBMS_WORKLOAD_CAPTURE Overview 186-1
186.2 DBMS_WORKLOAD_CAPTURE Security Model 186-1
186.3 Summary of DBMS_WORKLOAD_CAPTURE Subprograms 186-2
  186.3.1 ADD_FILTER Procedures 186-3
  186.3.2 DECRYPT_CAPTURE Procedure 186-4
  186.3.3 DELETE_CAPTURE_INFO Procedure 186-5
  186.3.4 DELETE_FILTER Procedure 186-5
  186.3.5 ENCRYPT_CAPTURE Procedure 186-6
187 DBMS_WORKLOAD_REPLAY

187.1 DBMS_WORKLOAD_REPLAY Overview
187.2 DBMS_WORKLOAD_REPLAY Security Model
187.3 Summary of DBMS_WORKLOAD_REPLAY Subprograms

187.3.1 ADD_CAPTURE Function
187.3.2 ADD_FILTER Procedure
187.3.3 ADD_SCHEDULE_ORDERING Function
187.3.4 ASSIGN_GROUP_TO_INSTANCE Procedure
187.3.5 BEGIN_REPLAY_SCHEDULE Procedure
187.3.6 CALIBRATE Function
187.3.7 CANCEL_REPLAY Procedure
187.3.8 COMPARE_PERIOD_REPORT Procedure
187.3.9 COMPARE_SQLSET_REPORT Function
187.3.10 CREATE_FILTER_SET Procedure
187.3.11 DELETE_FILTER Procedure
187.3.12 DELETE_REPLAY_INFO Procedure
187.3.13 END_REPLAY_SCHEDULE Procedure
187.3.14 EXPORT_AWR Procedure
187.3.15 GENERATE_CAPTURE_SUBSET Procedure
187.3.16 GET_DIVERGING_STATEMENT Function
187.3.17 GET_REPLAY_DIRECTORY Function
187.3.18 GET_REPLAY_INFO Function
187.3.19 GET_REPLAY_TIMEOUT Procedure
187.3.20 IMPORT_AWR Function
187.3.21 INITIALIZE_CONsolidated_REPLAY Procedure
187.3.22 INITIALIZE_REPLAY Procedure
187.3.23 IS_REPLAY_PAUSED Function
187.3.24 LOAD_LONG_SQLTEXT Procedure
187.3.25 PAUSE_REPLAY Procedure
187.3.26 POPULATE_DIVERGENCE Procedure
187.3.27 PREPARE_CONsolidated_REPLAY Procedure
187.3.28 PREPARE_REPLAY Procedure
187.3.29 PROCESS_CAPTURE Procedure
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>187.3.30</td>
<td>REMAP_CONNECTION Procedure</td>
</tr>
<tr>
<td>187.3.31</td>
<td>REMOVE_CAPTURE Procedure</td>
</tr>
<tr>
<td>187.3.32</td>
<td>REMOVE_SCHEDULE_ORDERING Procedure</td>
</tr>
<tr>
<td>187.3.33</td>
<td>REPORT Function</td>
</tr>
<tr>
<td>187.3.34</td>
<td>RESUME_REPLAY Procedure</td>
</tr>
<tr>
<td>187.3.35</td>
<td>REUSE_REPLAY_FILTER_SET Procedure</td>
</tr>
<tr>
<td>187.3.36</td>
<td>SET_ADVANCED_PARAMETER Procedure</td>
</tr>
<tr>
<td>187.3.37</td>
<td>SET_REPLAY_DIRECTORY Procedure</td>
</tr>
<tr>
<td>187.3.38</td>
<td>SET_REPLAY_TIMEOUT Procedure</td>
</tr>
<tr>
<td>187.3.39</td>
<td>SET_SQL_MAPPING Procedure</td>
</tr>
<tr>
<td>187.3.40</td>
<td>SET_USER_MAPPING Procedure</td>
</tr>
<tr>
<td>187.3.41</td>
<td>START_CONSOLIDATED_REPLAY Procedure</td>
</tr>
<tr>
<td>187.3.42</td>
<td>START_REPLAY Procedure</td>
</tr>
<tr>
<td>187.3.43</td>
<td>USE_FILTER_SET Procedure</td>
</tr>
</tbody>
</table>

188 DBMS_WORKLOAD_REPOSITORY

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>188.1</td>
<td>DBMS_WORKLOAD_REPOSITORY Examples</td>
</tr>
<tr>
<td>188.2</td>
<td>DBMS_WORKLOAD_REPOSITORY Data Structures</td>
</tr>
<tr>
<td>188.2.1</td>
<td>DBMS_WORKLOAD_REPOSITORY AWR_BASELINE_METRIC_TYPE Object Type</td>
</tr>
<tr>
<td>188.2.2</td>
<td>DBMS_WORKLOAD_REPOSITORY AWR_BASELINE_METRIC_TYPE_TABLE Table Type</td>
</tr>
<tr>
<td>188.2.3</td>
<td>DBMS_WORKLOAD_REPOSITORY AWRRPT_INSTANCE_LIST_TYPE Table Type</td>
</tr>
<tr>
<td>188.3</td>
<td>Summary of DBMS_WORKLOAD_REPOSITORY Subprograms</td>
</tr>
<tr>
<td>188.3.1</td>
<td>ADD_COLORED_SQL Procedure</td>
</tr>
<tr>
<td>188.3.2</td>
<td>ASH_GLOBAL_REPORT_HTML Function</td>
</tr>
<tr>
<td>188.3.3</td>
<td>ASH_GLOBAL_REPORT_TEXT Function</td>
</tr>
<tr>
<td>188.3.4</td>
<td>ASH_REPORT_ANALYTICS Function</td>
</tr>
<tr>
<td>188.3.5</td>
<td>ASH_REPORT_HTML Function</td>
</tr>
<tr>
<td>188.3.6</td>
<td>ASH_REPORT_TEXT Function</td>
</tr>
<tr>
<td>188.3.7</td>
<td>AWR_DIFF_REPORT_HTML Function</td>
</tr>
<tr>
<td>188.3.8</td>
<td>AWR_DIFF_REPORT_TEXT Function</td>
</tr>
<tr>
<td>188.3.9</td>
<td>AWR_GLOBAL_DIFF_REPORT_HTML Functions</td>
</tr>
<tr>
<td>188.3.10</td>
<td>AWR_GLOBAL_DIFF_REPORT_TEXT Functions</td>
</tr>
<tr>
<td>188.3.11</td>
<td>AWR_GLOBAL_REPORT_HTML Functions</td>
</tr>
<tr>
<td>188.3.12</td>
<td>AWR_GLOBAL_REPORT_TEXT Functions</td>
</tr>
<tr>
<td>188.3.13</td>
<td>AWR_REPORT_HTML Function</td>
</tr>
<tr>
<td>188.3.14</td>
<td>AWR_REPORT_TEXT Function</td>
</tr>
<tr>
<td>188.3.15</td>
<td>AWR_SET_REPORT_THRESHOLDS Procedure</td>
</tr>
<tr>
<td>188.3.16</td>
<td>AWR_SQL_REPORT_HTML Function</td>
</tr>
</tbody>
</table>
188.3.17  AWR_SQL_REPORT_TEXT Function 188-24
188.3.18  CONTROL_RESTRICTED_SNAPSHOT Procedure 188-25
188.3.19  CREATE_BASELINE Functions and Procedures 188-26
188.3.20  CREATE_BASELINE_TEMPLATE Procedures 188-27
188.3.21  CREATE.REMOTE_SNAPSHOT Function and Procedure 188-28
188.3.22  CREATE_SNAPSHOT Function and Procedure 188-29
188.3.23  DROP_BASELINE Procedure 188-30
188.3.24  DROP_BASELINETEMPLATE Procedure 188-31
188.3.25  DROP_SNAPSHOT_RANGE Procedure 188-31
188.3.26  LOCAL_AWR_DBID Function 188-32
188.3.27  MODIFY_BASELINE_WINDOW_SIZE Procedure 188-32
188.3.28  MODIFY_SNAPSHOT_SETTINGS Procedures 188-33
188.3.29  PURGE_SQL_DETAILS Procedure 188-34
188.3.30  REGISTER_REMOTE_DATABASE Procedures 188-35
188.3.31  REMOVE_COLORED_SQL Procedure 188-36
188.3.32  RENAME_BASELINE Procedure 188-36
188.3.33  SELECT_BASELINE_DETAILS Function 188-36
188.3.34  SELECT_BASELINE_METRIC Function 188-36
188.3.35  UNREGISTER_REMOTE_DATABASE Procedures 188-38
188.3.36  UPDATE_DATAFILE_INFO Procedure 188-38
188.3.37  UPDATE_OBJECT_INFO Procedure 188-39

189  DBMS_XA

189.1  DBMS_XA Overview 189-1
189.2  DBMS_XA Security Model 189-1
189.3  DBMS_XA Constants 189-2
189.4  DBMS_XA Operational Notes 189-3
189.5  DBMS_XA Data Structures 189-4
    189.5.1  DBMS_XA DBMS_XA_XID Object Type 189-4
    189.5.2  DBMS_XA DBMS_XA_XID_ARRAY Table Type 189-5
189.6  Summary of DBMS_XA Subprograms 189-5
    189.6.1  DIST_TXN_SYNC Procedure 189-6
    189.6.2  XA_COMMIT Function 189-6
    189.6.3  XA_END Function 189-7
    189.6.4  XA_FORGET Function 189-8
    189.6.5  XA_GETLASTOPER Function 189-8
    189.6.6  XA_PREPARE Function 189-8
    189.6.7  XA_RECOVER Function 189-9
    189.6.8  XA_ROLLBACK Function 189-9
    189.6.9  XA_SETTIMEOUT Function 189-10
190  DBMS_XDB

190.1  DBMS_XDB Overview
190.2  DBMS_XDB Security Model
190.3  DBMS_XDB Constants
190.4  Summary of DBMS_XDB Subprograms
   190.4.1  ACLCHECKPRIVILEGES Function
   190.4.2  APPENDRESOURCEMETADATA Procedure
   190.4.3  CHANGEOWNER Procedure
   190.4.4  CHANGEPRIVILEGES Function
   190.4.5  CHECKPRIVILEGES Function
   190.4.6  CREATEFOLDER Function
   190.4.7  CREATEOIDPATH Function
   190.4.8  CREATERSOURCE Functions
   190.4.9  DELETESOURCE Procedure
   190.4.10  DELETESOURCEMETADATA Procedures
   190.4.11  ENABLEDIGESTAUTHENTICATION Procedure
   190.4.12  EXISTSRESOURCE Function
   190.4.13  GETACLDOCUMENT Function
   190.4.14  GETCONTENTBLOB Function
   190.4.15  GETCONTENTCLOB Function
   190.4.16  GETCONTENTVARCHAR2 Function
   190.4.17  GETCONTENTXMLREF Function
   190.4.18  GETCONTENTXMLTYPE Function
   190.4.19  GETHTTPREQUESTHEADER Function
   190.4.20  GETLOCKTOKEN Procedure
   190.4.21  GETPRIVILEGES Function
   190.4.22  GETRESOID Function
   190.4.23  GETXDB_TABLESPACE Function
   190.4.24  HASBLOBCONTENT Function
   190.4.25  HASCHARCONTENT Function
   190.4.26  HASXMLCONTENT Function
   190.4.27  HASXMLREFERENCE Function
   190.4.28  ISFOLDER Function
   190.4.29  LINK Procedures
   190.4.30  LOCKRESOURCE Function
   190.4.31  PROCESSLINKS Procedure
   190.4.32  PURGERESOURCEMETADATA Procedure
   190.4.33  RENAMERSOURCE Procedure
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>190.4.34</td>
<td>SETACL Procedure</td>
<td>190-27</td>
</tr>
<tr>
<td>190.4.35</td>
<td>SPLITPATH Procedure</td>
<td>190-28</td>
</tr>
<tr>
<td>190.4.36</td>
<td>TOUCHRESOURCE Procedure</td>
<td>190-29</td>
</tr>
<tr>
<td>190.4.37</td>
<td>UNLOCKRESOURCE Function</td>
<td>190-29</td>
</tr>
<tr>
<td>190.4.38</td>
<td>UPDATERESOURCEMETADATA Procedures</td>
<td>190-30</td>
</tr>
<tr>
<td>191</td>
<td>DBMS_XDB_ADMIN</td>
<td></td>
</tr>
<tr>
<td>191.1</td>
<td>DBMS_XDB_ADMIN Security Model</td>
<td>191-1</td>
</tr>
<tr>
<td>191.2</td>
<td>Summary of DBMS_XDB_ADMIN Subprograms</td>
<td>191-1</td>
</tr>
<tr>
<td>191.2.1</td>
<td>CREATENONCEKEY Procedure</td>
<td>191-1</td>
</tr>
<tr>
<td>191.2.2</td>
<td>INSTALLDEFAULTWALLET Procedure</td>
<td>191-2</td>
</tr>
<tr>
<td>191.2.3</td>
<td>MOVEXDB_TABLESPACE Procedure</td>
<td>191-2</td>
</tr>
<tr>
<td>191.2.4</td>
<td>REBUILDHIERARCHICALINDEX Procedure</td>
<td>191-3</td>
</tr>
<tr>
<td>192</td>
<td>DBMS_XDB_CONFIG</td>
<td></td>
</tr>
<tr>
<td>192.1</td>
<td>DBMS_XDB_CONFIG Overview</td>
<td>192-1</td>
</tr>
<tr>
<td>192.2</td>
<td>DBMS_XDB_CONFIG Security Model</td>
<td>192-1</td>
</tr>
<tr>
<td>192.3</td>
<td>DBMS_XDB_CONFIG Constants</td>
<td>192-2</td>
</tr>
<tr>
<td>192.4</td>
<td>Summary of DBMS_XDB_CONFIG Subprograms</td>
<td>192-2</td>
</tr>
<tr>
<td>192.4.1</td>
<td>ADDHTTPEXPIREMAPPING Procedure</td>
<td>192-4</td>
</tr>
<tr>
<td>192.4.2</td>
<td>ADDMIMEMAPPING Procedure</td>
<td>192-4</td>
</tr>
<tr>
<td>192.4.3</td>
<td>ADDSCHEMALOCMAPPING Procedure</td>
<td>192-5</td>
</tr>
<tr>
<td>192.4.4</td>
<td>ADDSERVLET Procedure</td>
<td>192-5</td>
</tr>
<tr>
<td>192.4.5</td>
<td>ADDSERVLETMAPPING Procedure</td>
<td>192-6</td>
</tr>
<tr>
<td>192.4.6</td>
<td>ADDSERVLETSECROLE Procedure</td>
<td>192-7</td>
</tr>
<tr>
<td>192.4.7</td>
<td>ADDXMLEXTENSION Procedure</td>
<td>192-7</td>
</tr>
<tr>
<td>192.4.8</td>
<td>CFG_GET Function</td>
<td>192-8</td>
</tr>
<tr>
<td>192.4.9</td>
<td>CFG_REFRESH Procedure</td>
<td>192-8</td>
</tr>
<tr>
<td>192.4.10</td>
<td>CFG_UPDATE Procedure</td>
<td>192-8</td>
</tr>
<tr>
<td>192.4.11</td>
<td>DELETETTTPEXPIREMAPPING Procedure</td>
<td>192-9</td>
</tr>
<tr>
<td>192.4.12</td>
<td>DELETEMIMEMAPPING Procedure</td>
<td>192-9</td>
</tr>
<tr>
<td>192.4.13</td>
<td>DELETESCHEMALOCMAPPING Procedure</td>
<td>192-9</td>
</tr>
<tr>
<td>192.4.14</td>
<td>DELETESERVLET Procedure</td>
<td>192-10</td>
</tr>
<tr>
<td>192.4.15</td>
<td>DELETESERVLETMAPPING Procedure</td>
<td>192-10</td>
</tr>
<tr>
<td>192.4.16</td>
<td>DELETESERVLETSECROLE Procedure</td>
<td>192-10</td>
</tr>
<tr>
<td>192.4.17</td>
<td>DELETEXMLEXTENSION Procedure</td>
<td>192-11</td>
</tr>
<tr>
<td>192.4.18</td>
<td>ENABLEDIGESTAUTHENTICATION Procedure</td>
<td>192-11</td>
</tr>
<tr>
<td>192.4.19</td>
<td>GETFTPPORT Function</td>
<td>192-11</td>
</tr>
<tr>
<td>192.4.20</td>
<td>GETHTTPCONFIGREALM Function</td>
<td>192-11</td>
</tr>
</tbody>
</table>
192.4.21 GETHTTPPORT Function 192-11
192.4.22 GETHTTPSPORT Function 192-12
192.4.23 GETLISTENERENDPOINT Procedure 192-12
192.4.24 GETREMOTEHTTPPORT Function 192-12
192.4.25 GETREMOTEHTTPSPORT Function 192-13
192.4.26 ISGLOBALPORTENABLED Function 192-13
192.4.27 SETFTPPORT Procedure 192-13
192.4.28 SETGLOBALPORTENABLED Procedure 192-13
192.4.29 SETHTTPPORT Procedure 192-15
192.4.30 SETHTTPCONFIGREALM Procedure 192-15
192.4.31 SETHTTPSPORT Procedure 192-15
192.4.32 SETLISTENERENDPOINT Procedure 192-16
192.4.33 SETLISTENERLOCALACCESS Procedure 192-16
192.4.34 SETREMOTEHTTPPORT Procedure 192-17
192.4.35 SETREMOTEHTTPSPORT Procedure 192-17
192.4.36 USEDPORT Function 192-17

193 DBMS_XDB_CONSTANTS

<table>
<thead>
<tr>
<th>193.1</th>
<th>DBMS_XDB_CONSTANTS Security Model</th>
<th>193-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>193.2</td>
<td>Summary of DBMS_XDB_CONSTANTS Subprograms</td>
<td>193-1</td>
</tr>
<tr>
<td>193.2.1</td>
<td>ENCODING_DEFAULT Function</td>
<td>193-2</td>
</tr>
<tr>
<td>193.2.2</td>
<td>ENCODING_ISO LATIN1 Function</td>
<td>193-3</td>
</tr>
<tr>
<td>193.2.3</td>
<td>ENCODING_UTF8 Function</td>
<td>193-3</td>
</tr>
<tr>
<td>193.2.4</td>
<td>ENCODING_WIN1252 Function</td>
<td>193-3</td>
</tr>
<tr>
<td>193.2.5</td>
<td>NAMESPACE_ACL Function</td>
<td>193-3</td>
</tr>
<tr>
<td>193.2.6</td>
<td>NAMESPACE_METADATA Function</td>
<td>193-4</td>
</tr>
<tr>
<td>193.2.7</td>
<td>NAMESPACE_ORACLE Function</td>
<td>193-4</td>
</tr>
<tr>
<td>193.2.8</td>
<td>NAMESPACE_ORACLE_XDB Function</td>
<td>193-4</td>
</tr>
<tr>
<td>193.2.9</td>
<td>NAMESPACE_RESOURCE Function</td>
<td>193-4</td>
</tr>
<tr>
<td>193.2.10</td>
<td>NAMESPACE_RESOURCE_EVENT Function</td>
<td>193-5</td>
</tr>
<tr>
<td>193.2.11</td>
<td>NAMESPACE_RESOURCE_CONFIG Function</td>
<td>193-5</td>
</tr>
<tr>
<td>193.2.12</td>
<td>NAMESPACE_XDBSCHEMA Function</td>
<td>193-5</td>
</tr>
<tr>
<td>193.2.13</td>
<td>NAMESPACE_XMDIFF Function</td>
<td>193-5</td>
</tr>
<tr>
<td>193.2.14</td>
<td>NAMESPACE_XMLINSTANCE Function</td>
<td>193-6</td>
</tr>
<tr>
<td>193.2.15</td>
<td>NAMESPACE_XMLSHEMA Function</td>
<td>193-6</td>
</tr>
<tr>
<td>193.2.16</td>
<td>NSPREFIX_ACL_ACL Function</td>
<td>193-6</td>
</tr>
<tr>
<td>193.2.17</td>
<td>NSPREFIX_RESCONFIG_RC Function</td>
<td>193-6</td>
</tr>
<tr>
<td>193.2.18</td>
<td>NSPREFIX_RESOURCE_R Function</td>
<td>193-7</td>
</tr>
<tr>
<td>193.2.19</td>
<td>NSPREFIX_XDB_XDB Function</td>
<td>193-7</td>
</tr>
<tr>
<td>193.2.20</td>
<td>NSPREFIX_XMLINSTANCE_XSI Function</td>
<td>193-7</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>193.2.21</td>
<td>NSPREFIX_XMLEDIFF_XD Function</td>
<td></td>
</tr>
<tr>
<td>193.2.22</td>
<td>NSPREFIX_XMLSHEMA_XSD Function</td>
<td></td>
</tr>
<tr>
<td>193.2.23</td>
<td>SCHEMAURL_ACL Function</td>
<td></td>
</tr>
<tr>
<td>193.2.24</td>
<td>SCHEMALEM_RES_ACL Function</td>
<td></td>
</tr>
<tr>
<td>193.2.25</td>
<td>SCHEMALEM_RESCONTENT_BINARY Function</td>
<td></td>
</tr>
<tr>
<td>193.2.26</td>
<td>SCHEMALEM_RESCONTENT_TEXT Function</td>
<td></td>
</tr>
<tr>
<td>193.2.27</td>
<td>SCHEMAURL_RESOURCE Function</td>
<td></td>
</tr>
<tr>
<td>193.2.28</td>
<td>SCHEMAURL_XDBSCHEMA Function</td>
<td></td>
</tr>
<tr>
<td>193.2.29</td>
<td>XDBSCHEMA_PREFIXES Function</td>
<td></td>
</tr>
<tr>
<td>193.2.30</td>
<td>XSD_ATTRIBUTE Function</td>
<td></td>
</tr>
<tr>
<td>193.2.31</td>
<td>XSD_COMPLEX_TYPE Function</td>
<td></td>
</tr>
<tr>
<td>193.2.32</td>
<td>XSD_ELEMENT Function</td>
<td></td>
</tr>
<tr>
<td>193.2.33</td>
<td>XSD_GROUP Function</td>
<td></td>
</tr>
</tbody>
</table>

194  
**DBMS_XDB_REPOS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>194.1</td>
<td>DBMS_XDB_REPOS Overview</td>
</tr>
<tr>
<td>194.2</td>
<td>DBMS_XDB_REPOS Security Model</td>
</tr>
<tr>
<td>194.3</td>
<td>DBMS_XDB_REPOS Constants</td>
</tr>
<tr>
<td>194.4</td>
<td>Summary of DBMS_XDB_REPOS Subprograms</td>
</tr>
<tr>
<td>194.4.1</td>
<td>ACLCHECKPRIVILEGES Function</td>
</tr>
<tr>
<td>194.4.2</td>
<td>APPENDRESOURCESMETADATA Procedure</td>
</tr>
<tr>
<td>194.4.3</td>
<td>CHANGEOWNER Procedure</td>
</tr>
<tr>
<td>194.4.4</td>
<td>CHANGEPRIVILEGES Function</td>
</tr>
<tr>
<td>194.4.5</td>
<td>CHECKPRIVILEGES Function</td>
</tr>
<tr>
<td>194.4.6</td>
<td>CREATEFOLDER Function</td>
</tr>
<tr>
<td>194.4.7</td>
<td>CREATEOIDPATH Function</td>
</tr>
<tr>
<td>194.4.8</td>
<td>CREATERESOURCE Functions</td>
</tr>
<tr>
<td>194.4.9</td>
<td>DELETERESOURCE Procedure</td>
</tr>
<tr>
<td>194.4.10</td>
<td>DELETERESOURCESMETADATA Procedures</td>
</tr>
<tr>
<td>194.4.11</td>
<td>EXISTSRESOURCE Function</td>
</tr>
<tr>
<td>194.4.12</td>
<td>GETACLDOCUMENT Function</td>
</tr>
<tr>
<td>194.4.13</td>
<td>GETCONTENTBLOB Function</td>
</tr>
<tr>
<td>194.4.14</td>
<td>GETCONTENTCLOB Function</td>
</tr>
<tr>
<td>194.4.15</td>
<td>GETCONTENTVARCHAR2 Function</td>
</tr>
<tr>
<td>194.4.16</td>
<td>GETCONTENTXMLREF Function</td>
</tr>
<tr>
<td>194.4.17</td>
<td>GETCONTENTXMLTYPE Function</td>
</tr>
<tr>
<td>194.4.18</td>
<td>GETLOCKTOKEN Procedure</td>
</tr>
<tr>
<td>194.4.19</td>
<td>GETPRIVILEGES Function</td>
</tr>
<tr>
<td>194.4.20</td>
<td>GETRESOID Function</td>
</tr>
<tr>
<td>194.4.21</td>
<td>GETXDB_TABLESPACE Function</td>
</tr>
</tbody>
</table>
195 DBMS_XDB_VERSION

195.1 DBMS_XDB_VERSION Security Model
195.2 Summary of DBMS_XDB_VERSION Subprograms
  195.2.1 CHECKIN Function
  195.2.2 CHECKOUT Procedure
  195.2.3 GETCONTENTSBLOBBYRESID Function
  195.2.4 GETCONTENTSCLOBBYRESID Function
  195.2.5 GETCONTENTSTXSXMLBYRESID Function
  195.2.6 GETPREDECESSORS Function
  195.2.7 GETPREDSBYRESID Function
  195.2.8 GETRESOURCEBYRESID Function
  195.2.9 GETSUCCESSORS Function
  195.2.10 GETSUCCSBYRESID Function
  195.2.11 MAKEVERSIONED Function
  195.2.12 UNCHECKOUT Function

196 DBMS_XDBRESOURCE

196.1 DBMS_XDBRESOURCE Overview
196.2 DBMS_XDBRESOURCE Security Model
196.3 Summary of DBMS_XDBRESOURCE Subprograms
  196.3.1 FREERESOURCE Procedure
  196.3.2 GETACL Function
  196.3.3 GETACLDOCFROMRES Function
| 196.3.4 | GETAUTHOR Function | 196-5 |
| 196.3.5 | GETCHARACTERSET Function | 196-5 |
| 196.3.6 | GETCOMMENT Function | 196-5 |
| 196.3.7 | GETCONTENTBLOB Function | 196-6 |
| 196.3.8 | GETCONTENTCLOB Function | 196-6 |
| 196.3.9 | GETCONTENTREF Function | 196-6 |
| 196.3.10 | GETCONTENTTYPE Function | 196-7 |
| 196.3.11 | GETCONTENTXML Function | 196-7 |
| 196.3.12 | GETCONTENTVARCHAR2 Function | 196-7 |
| 196.3.13 | GETCREATIONDATE Function | 196-8 |
| 196.3.14 | GETCREATOR Function | 196-8 |
| 196.3.15 | GETCUSTOMMETADATA Function | 196-8 |
| 196.3.16 | GETDISPLAYNAME Function | 196-9 |
| 196.3.17 | GETLANGUAGE Function | 196-9 |
| 196.3.18 | GETLASTMODIFIER Function | 196-10 |
| 196.3.19 | GETMODIFICATIONDATE Function | 196-10 |
| 196.3.20 | GETOWNER Function | 196-10 |
| 196.3.21 | GETREFCOUNT Function | 196-11 |
| 196.3.22 | GETVERSIONID Function | 196-11 |
| 196.3.23 | HASACLCHANGED Function | 196-11 |
| 196.3.24 | HASAUTHORCHANGED Function | 196-12 |
| 196.3.25 | HASCHANGED Function | 196-12 |
| 196.3.26 | HASCHARACTERSETCHANGED Function | 196-12 |
| 196.3.27 | HASCOMMENTCHANGED Function | 196-13 |
| 196.3.28 | HASCONTENTCHANGED Function | 196-13 |
| 196.3.29 | HASCONTENTTYPECHANGED Function | 196-14 |
| 196.3.30 | HASCREATIONDATECHANGED Function | 196-14 |
| 196.3.31 | HASCUSTOMMETADATACHANGED Function | 196-15 |
| 196.3.32 | HASCUSTOMMETADATASETCHANGED Function | 196-15 |
| 196.3.33 | HASDISPLAYNAMECHANGED Function | 196-15 |
| 196.3.34 | HASLANGUAGECHANGED Function | 196-16 |
| 196.3.35 | HASTEXTCHANGED Function | 196-16 |
| 196.3.36 | HASMODIFICATIONDATECHANGED Function | 196-16 |
| 196.3.37 | HASONOWNERCHANGED Function | 196-17 |
| 196.3.38 | HASTEXTCHANGED Function | 196-17 |
| 196.3.39 | HASTEXTCHANGED Function | 196-18 |
| 196.3.40 | ISFOLDER Function | 196-18 |
| 196.3.41 | ISNULL Function | 196-18 |
| 196.3.42 | MAKEDOCUMENT Function | 196-19 |
| 196.3.43 | SAVE Procedure | 196-19 |
| 196.3.44 | SETACL Procedure | 196-20 |
### DBMS_XDBT

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>197.1</td>
<td>DBMS_XDBT Overview</td>
</tr>
<tr>
<td>197.2</td>
<td>DBMS_XDBT Security Model</td>
</tr>
<tr>
<td>197.3</td>
<td>DBMS_XDBT Operational Notes</td>
</tr>
<tr>
<td>197.4</td>
<td>Summary of DBMS_XDBT Subprograms</td>
</tr>
<tr>
<td>197.4.1</td>
<td>CONFIGUREAUTOSYNC Procedure</td>
</tr>
<tr>
<td>197.4.2</td>
<td>CREATEDATASTOREPREF Procedure</td>
</tr>
<tr>
<td>197.4.3</td>
<td>CREATEFILTERPREF Procedure</td>
</tr>
<tr>
<td>197.4.4</td>
<td>CREATEINDEX Procedure</td>
</tr>
<tr>
<td>197.4.5</td>
<td>CREATELEXERPREF Procedure</td>
</tr>
<tr>
<td>197.4.6</td>
<td>CREATEPREFERENCES Procedure</td>
</tr>
<tr>
<td>197.4.7</td>
<td>CREATESECTIONGROUPPREF Procedure</td>
</tr>
<tr>
<td>197.4.8</td>
<td>CREATESTOPLISTPREF Procedure</td>
</tr>
<tr>
<td>197.4.9</td>
<td>CREATESTORAGEPREF Procedure</td>
</tr>
<tr>
<td>197.4.10</td>
<td>CREATERWORLDLISTPREF Procedure</td>
</tr>
<tr>
<td>197.4.11</td>
<td>DROPPREFERENCES Procedure</td>
</tr>
</tbody>
</table>

### DBMS_XDBZ

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>198.1</td>
<td>DBMS_XDBZ Security Model</td>
</tr>
<tr>
<td>198.2</td>
<td>DBMS_XDBZ Constants</td>
</tr>
<tr>
<td>198.3</td>
<td>Summary of DBMS_XDBZ Subprograms</td>
</tr>
<tr>
<td>198.3.1</td>
<td>CREATERWORLDLISTPREF Procedure</td>
</tr>
<tr>
<td>198.3.2</td>
<td>DISABLE_HIERARCHY Procedure</td>
</tr>
<tr>
<td>198.3.3</td>
<td>ENABLE_HIERARCHY Procedure</td>
</tr>
<tr>
<td>198.3.4</td>
<td>GET_ACLOID Function</td>
</tr>
<tr>
<td>198.3.5</td>
<td>GET_USERID Function</td>
</tr>
<tr>
<td>198.3.6</td>
<td>IS_HIERARCHY_ENABLED Function</td>
</tr>
<tr>
<td>198.3.7</td>
<td>PURGELDAPCACHE Function</td>
</tr>
</tbody>
</table>
199.1 DBMS_XEVENT Security Model
199.2 DBMS_XEVENT Constants
199.3 Subprogram Groups
   199.3.1 DBMS_XEVENT XDBEvent Type Subprograms
   199.3.2 DBMS_XEVENT XDBRepositoryEvent Type Subprograms
   199.3.3 DBMS_XEVENT XDBHandlerList Type Subprograms
   199.3.4 DBMS_XEVENT XDBHandler Type Subprograms
   199.3.5 DBMS_XEVENT XDBPath Type Subprograms
   199.3.6 DBMS_XEVENT XDBLink Type Subprograms
199.4 Summary of DBMS_XEVENT Subprograms
   199.4.1 CLEAR Procedure
   199.4.2 GETAPPLICATIONDATA Function
   199.4.3 GETCHILDOID Function
   199.4.4 GETCURRENTUSER Function
   199.4.5 GETEVENT Function
   199.4.6 GETFIRST Function
   199.4.7 GETHANDLERLIST Function
   199.4.8 GETINTERFACE Function
   199.4.9 GETLANGUAGE Function
   199.4.10 GETLINK Function
   199.4.11 GETLINKNAME Function
   199.4.12 GETLOCK Function
   199.4.13 GETLANGUAGE Function
   199.4.14 GETNAME Function
   199.4.15 GETNEXT Function
   199.4.16 GETOLDRESOURCE Function
   199.4.17 GETOPENACCESSMODE Function
   199.4.18 GETOPENDENYMODE Function
   199.4.19 GETOUTPUTSTREAM Function
   199.4.20 GETPARAMETER Function
   199.4.21 GPARENT Function
   199.4.22 GEPARENTNAME Function
   199.4.23 GEPARENTOID Function
   199.4.24 GEPARENTPATH Function
   199.4.25 GEPATH Function
   199.4.26 GETHRESOURCES Function
   199.4.27 GETHSCHEMA Function
   199.4.28 GETHSOURCE Function
   199.4.29 GUPDATEBYTECOUNT Function
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>199.4.30</td>
<td>GETUPDATEBYTEOFFSET Function</td>
</tr>
<tr>
<td>199.4.31</td>
<td>GETXDBEVENT Function</td>
</tr>
<tr>
<td>199.4.32</td>
<td>ISNULL Functions</td>
</tr>
<tr>
<td>199.4.33</td>
<td>REMOVE Procedure</td>
</tr>
<tr>
<td>199.4.34</td>
<td>SETRENDERPATH Procedure</td>
</tr>
<tr>
<td>199.4.35</td>
<td>SETRENDERSTREAM Procedure</td>
</tr>
</tbody>
</table>

## DBMS_XMLDOM

### 200 DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.1</td>
<td>DBMS_XMLDOM Overview</td>
</tr>
<tr>
<td>200.2</td>
<td>DBMS_XMLDOM Security Model</td>
</tr>
<tr>
<td>200.3</td>
<td>DBMS_XMLDOM Constants</td>
</tr>
<tr>
<td>200.4</td>
<td>DBMS_XMLDOM Types</td>
</tr>
<tr>
<td>200.5</td>
<td>DBMS_XMLDOM Exceptions</td>
</tr>
<tr>
<td>200.6</td>
<td>DBMS_XMLDOM Subprogram Groups</td>
</tr>
<tr>
<td>200.6.1</td>
<td>DBMS_XMLDOM DOMNode Subprograms</td>
</tr>
<tr>
<td>200.6.2</td>
<td>DBMS_XMLDOM DOMAttr Subprograms</td>
</tr>
<tr>
<td>200.6.3</td>
<td>DBMS_XMLDOM DOMCDataSection Subprograms</td>
</tr>
<tr>
<td>200.6.4</td>
<td>DBMS_XMLDOM DOMCharacterData Subprograms</td>
</tr>
<tr>
<td>200.6.5</td>
<td>DBMS_XMLDOM DOMComment Subprograms</td>
</tr>
<tr>
<td>200.6.6</td>
<td>DBMS_XMLDOM DOMDocument Subprograms</td>
</tr>
<tr>
<td>200.6.7</td>
<td>DBMS_XMLDOM DOMDocumentFragment Subprograms</td>
</tr>
<tr>
<td>200.6.8</td>
<td>DBMS_XMLDOM DOMDocumentType Subprograms</td>
</tr>
<tr>
<td>200.6.9</td>
<td>DBMS_XMLDOM DOMElement Subprograms</td>
</tr>
<tr>
<td>200.6.10</td>
<td>DBMS_XMLDOM DOMEntity Subprograms</td>
</tr>
<tr>
<td>200.6.11</td>
<td>DBMS_XMLDOM DOMEntityReference Subprograms</td>
</tr>
<tr>
<td>200.6.12</td>
<td>DBMS_XMLDOM DOMImplementation Subprograms</td>
</tr>
<tr>
<td>200.6.13</td>
<td>DBMS_XMLDOM DOMNamedNodeMap Subprograms</td>
</tr>
<tr>
<td>200.6.14</td>
<td>DBMS_XMLDOMDOMNodeList Subprograms</td>
</tr>
<tr>
<td>200.6.15</td>
<td>DBMS_XMLDOM DOMNotation Subprograms</td>
</tr>
<tr>
<td>200.6.16</td>
<td>DBMS_XMLDOM DOMProcessingInstruction Subprograms</td>
</tr>
<tr>
<td>200.6.17</td>
<td>DBMS_XMLDOM DOMText Subprograms</td>
</tr>
<tr>
<td>200.7</td>
<td>Summary of DBMS_XMLDOM Subprograms</td>
</tr>
<tr>
<td>200.7.1</td>
<td>ADOPTNODE Function</td>
</tr>
<tr>
<td>200.7.2</td>
<td>APPENDCHILD Function</td>
</tr>
<tr>
<td>200.7.3</td>
<td>APPENDDATA Procedure</td>
</tr>
<tr>
<td>200.7.4</td>
<td>CLONENODE Function</td>
</tr>
<tr>
<td>200.7.5</td>
<td>CREATEATTRIBUTE Functions</td>
</tr>
<tr>
<td>200.7.6</td>
<td>CREATECDATASECTION Function</td>
</tr>
<tr>
<td>200.7.7</td>
<td>CREATECOMMENT Function</td>
</tr>
<tr>
<td>200.7.8</td>
<td>CREATEDOCUMENT Function</td>
</tr>
</tbody>
</table>
CREATEDOCUMENTFRAGMENT Function 200-26
CREATEELEMENT Functions 200-26
CREATEENTITYREFERENCE Function 200-27
CREATEPROCESSINGINSTRUCTION Function 200-27
CREATETEXTNODE Function 200-28
DELETEDATA Procedure 200-29
FINDENTITY Function 200-29
FINDNOTATION Function 200-30
FREEDOCFRAG Procedure 200-30
FREEDOCUMENT Procedure 200-31
FREEELEMENT Procedure 200-31
FREENODE Procedure 200-32
FREENODELIST Procedure 200-32
GETATTRIBUTE Functions 200-33
GETATTRIBUTENODE Functions 200-34
GETATTRIBUTES Function 200-34
GETATTRIBUTE Functions 200-34
GETCHARSET Function 200-35
GETCHILDNODES Function 200-35
GETCHILDRENBYTAGNAME Functions 200-36
GETDATA Functions 200-37
GETDOCTYPE Function 200-37
GETDOCUMENTELEMENT Function 200-38
GETELEMENTSBYTAGNAME Functions 200-38
GETENTITIES Function 200-39
GETEXPANDEDNAME Procedure and Functions 200-39
GETFIRSTCHILD Function 200-40
GETIMPLEMENTATION Function 200-41
GETLASTCHILD Function 200-41
GETLENGTH Functions 200-42
GETLOCALNAME Procedure and Functions 200-42
GETNAME Functions 200-43
GETNAMEDITEM Function 200-44
GETNAMESPACE Procedure and Functions 200-44
GETNEXTSIBLING Function 200-45
GETNODETYPE Function 200-46
GETNODENAME Function 200-46
GETNODEVALUE Function 200-47
GETNODEVALUEASBINARYSTREAM Function & Procedure 200-47
GETNODEVALUEASCHARACTERSTREAM Function & Procedure 200-48
GETNOTATIONNAME Function 200-49
GETNOTATIONS Function 200-49
200.7.50  GETTARGET Function                      200-50
200.7.51  GETOWNERDOCUMENT Function               200-50
200.7.52  GETOWNERELEMENT Function                 200-51
200.7.53  GETPARENTPNODE Function                  200-51
200.7.54  GETPREFIX Function                      200-52
200.7.55  GETPREVIOUSSibling Function             200-52
200.7.56  GETPUBLICID Functions                   200-52
200.7.57  GETQUALIFIEDNAME Functions              200-53
200.7.58  GETSCHEMANODE Function                   200-54
200.7.59  GETSPECIFIED Function                   200-54
200.7.60  GETSTANDALONE Function                   200-55
200.7.61  GETSYSTEMID Functions                    200-55
200.7.62  GETTAGNAME Function                     200-56
200.7.63  GETVALUE Function                       200-56
200.7.64  GETVERSION Function                     200-57
200.7.65  GETXMLTYPE Function                      200-57
200.7.66  HASATTRIBUTE Functions                  200-58
200.7.67  HASATTRIBUTES Function                   200-59
200.7.68  HASCHILDNODES Function                   200-59
200.7.69  HASFEATURE Function                      200-60
200.7.70  IMPORTNODE Function                      200-60
200.7.71  INSERTBEFORE Function                    200-61
200.7.72  INSERTDATA Procedure                    200-62
200.7.73  ISNULL Functions                        200-62
200.7.74  ITEM Functions                          200-65
200.7.75  MAKEATTR Function                        200-66
200.7.76  MAKECDATASECTION Function                200-66
200.7.77  MAKECHARACTERDATA Function               200-67
200.7.78  MAKECOMMENT Function                     200-67
200.7.79  MADEDOCUMENT Function                    200-68
200.7.80  MADEDOCUMENTFRAGMENT Function            200-68
200.7.81  MADEDOCUMENTTYPE Function                200-69
200.7.82  MAKEELEMENT Function                     200-69
200.7.83  MAKEENTITY Function                      200-70
200.7.84  MAKEENTITYREFERENCE Function            200-70
200.7.85  MAKENODE Functions                       200-71
200.7.86  MAKENOTATION Function                    200-73
200.7.87  MAKEPROCESSINGINSTRUCTION Function        200-73
200.7.88  MAKETEXT Function                        200-74
200.7.89  NEWDOMDOCUMENT Functions                 200-74
200.7.90  NORMALIZE Procedure                     200-75
| 200.7.91 | REMOVEATTRIBUTE Procedures | 200-75 |
| 200.7.92 | REMOVEATTRIBUTENODE Function | 200-76 |
| 200.7.93 | REMOVECHILD Function | 200-77 |
| 200.7.94 | REMOVENAMEDITEM Function | 200-77 |
| 200.7.95 | REPLACECHILD Function | 200-78 |
| 200.7.96 | REPLACEDATA Procedure | 200-79 |
| 200.7.97 | RESOLVENAMESPACEPREFIX Function | 200-79 |
| 200.7.98 | SETATTRIBUTE Procedures | 200-80 |
| 200.7.99 | SETATTRIBUTENODE Functions | 200-81 |
| 200.7.100 | SETCHARSET Procedure | 200-81 |
| 200.7.101 | SETDATA Procedures | 200-82 |
| 200.7.102 | SETDOCTYPE Procedure | 200-82 |
| 200.7.103 | SETNAMEDITEM Function | 200-83 |
| 200.7.104 | SETNODEVALUE Procedure | 200-84 |
| 200.7.105 | SETNODEVALUEASBINARYSTREAM Function & Procedure | 200-84 |
| 200.7.106 | SETNODEVALUEASCHARACTERSTREAM Function & Procedure | 200-85 |
| 200.7.107 | SETPREFIX Procedure | 200-86 |
| 200.7.108 | SETSTANDALONE Procedure | 200-86 |
| 200.7.109 | SETVALUE Procedure | 200-87 |
| 200.7.110 | SETVERSION Procedure | 200-87 |
| 200.7.111 | SPLITTEXT Function | 200-88 |
| 200.7.112 | SUBSTRINGDATA Function | 200-88 |
| 200.7.113 | USEBINARYSTREAM Function | 200-89 |
| 200.7.114 | WritetoBuffer Procedures | 200-89 |
| 200.7.115 | WritetoClob Procedures | 200-90 |
| 200.7.116 | WritetoFile Procedures | 200-91 |

201  DBMS_XMLGEN

| 201.1 | DBMS_XMLGEN Security Model | 201-1 |
| 201.2 | Summary of DBMS_XMLGEN Subprograms | 201-1 |
| 201.2.1 | CLOSECONTEXT Procedure | 201-2 |
| 201.2.2 | CONVERT Functions | 201-2 |
| 201.2.3 | GETNUMROWSPROCESSED Function | 201-3 |
| 201.2.4 | GETXML Functions | 201-4 |
| 201.2.5 | GETXMLTYPE Functions | 201-5 |
| 201.2.6 | NEWCONTEXT Functions | 201-5 |
| 201.2.7 | NEWCONTEXTFROMHIERARCHY Function | 201-6 |
| 201.2.8 | RESTARTQUERY Procedure | 201-6 |
| 201.2.9 | SETCONVERTSPECIALCHARS Procedure | 201-7 |
| 201.2.10 | SETMAXROWS Procedure | 201-7 |
### DBMS_XMLINDEX

**202.1** DBMS_XMLINDEX Overview 202-1

**202.2** DBMS_XMLINDEX Security Model 202-1

**202.3** Summary of DBMS_XMLINDEX Subprograms 202-1

- **202.3.1** CREATEDATEINDEX Procedure 202-2
- **202.3.2** CREATENUMBERINDEX Procedure 202-3
- **202.3.3** DROPPARAMETER Procedure 202-4
- **202.3.4** MODIFYPARAMETER Procedure 202-4
- **202.3.5** PROCESS_PENDING Procedure 202-5
- **202.3.6** REGISTERPARAMETER Procedure 202-5
- **202.3.7** SYNCINDEX Procedure 202-6

### DBMS_XMLPARSER

**203.1** DBMS_XMLPARSER Security Model 203-1

**203.2** Summary of DBMS_XMLPARSER Subprograms 203-1

- **203.2.1** FREEPARSER 203-2
- **203.2.2** GETDOCTYPE 203-3
- **203.2.3** GETDOCUMENT 203-3
- **203.2.4** GETRELEASEVERSION 203-3
- **203.2.5** GETVALIDATIONMODE 203-4
- **203.2.6** NEWPARSER 203-4
- **203.2.7** PARSE 203-4
- **203.2.8** PARSEBUFFER 203-5
- **203.2.9** PARSECLOB 203-5
- **203.2.10** PARSEDTD 203-6
- **203.2.11** PARSEDTDBUFFER 203-6
- **203.2.12** PARSEDTDCLOB 203-7
- **203.2.13** SETBASEDIR 203-7
- **203.2.14** SETDOCTYPE Procedure 203-8
- **203.2.15** SETERRORLOG Procedure 203-8
- **203.2.16** SETPRESERVEWHITESPACE 203-9
- **203.2.17** SETVALIDATIONMODE 203-9
204 DBMS_XMLQUERY

204.1 DBMS_XMLQUERY Security Model 204-1
204.2 DBMS_XMLQUERY Constants 204-1
204.3 Types 204-2
204.4 Summary of DBMS_XMLQUERY Subprograms 204-2
  204.4.1 CLOSECONTEXT 204-4
  204.4.2 GETDTD 204-4
  204.4.3 GETEXCEPTIONCONTENT 204-5
  204.4.4 GETNUMROWSPROCESSED 204-5
  204.4.5 GETVERSION 204-6
  204.4.6 GETXML 204-6
  204.4.7 NEWCONTEXT 204-7
  204.4.8 PROPAGATEORIGINALEXCEPTION 204-7
  204.4.9 REMOVEXSLTPARAM 204-8
  204.4.10 SETBINDVALUE 204-8
  204.4.11 SETCOLLIDATTRNAME 204-9
  204.4.12 SETDATAHEADER 204-9
  204.4.13 SETDATEFORMAT 204-9
  204.4.14 SETENCODINGTAG Procedure 204-10
  204.4.15 SETERRORTAG Procedure 204-10
  204.4.16 SETMAXROWS Procedure 204-11
  204.4.17 SETMETAHEADER Procedure 204-11
  204.4.18 SETRAISEEXCEPTION 204-11
  204.4.19 SETRAISENOROWSEXCEPTION 204-12
  204.4.20 SETROWIDATTRNAME 204-12
  204.4.21 SETROWIDATTRVALUE 204-12
  204.4.22 SETROWSETTAG 204-13
  204.4.23 SETROWTAG 204-13
  204.4.24 SETSKIPROWS 204-14
  204.4.25 SETSQLTOXMLNAMEESCAPING 204-14
  204.4.26 SETSTYLESHEETHEADER 204-14
  204.4.27 SETTAGCASE 204-15
  204.4.28 SETXSLT 204-15
  204.4.29 SETXSLTPARAM 204-16
  204.4.30 USENULLATTRIBUTEINDICATOR 204-16
  204.4.31 USETYPEFORCOLLELEMTAG 204-17
### DBMS_XMLSAVE

205.1 DBMS_XMLSAVE Security Model 205-1
205.2 DBMS_XMLSAVE Constants 205-1
205.3 Types 205-2
205.4 Summary of DBMS_XMLSAVE Subprograms 205-2
  205.4.1 CLEARKEYCOLUMNLIST 205-3
  205.4.2 CLEARUPDATECOLUMNLIST 205-3
  205.4.3 CLOSECONTEXT 205-4
  205.4.4 DELETEXML 205-4
  205.4.5 GETEXCEPTIONCONTENT 205-5
  205.4.6 INSERTXML 205-6
  205.4.7 NEWCONTEXT 205-6
  205.4.8 PROPAGATEORIGINALEXCEPTION 205-7
  205.4.9 REMOVEXSLTPARAM 205-7
  205.4.10 SETBATCHSIZE 205-7
  205.4.11 SETCOMMITBATCH 205-8
  205.4.12 SETDATEFORMAT 205-8
  205.4.13 SETIGNORECASE 205-9
  205.4.14 SETKEYCOLUMN 205-9
  205.4.15 SETPRESERVEWHITESPACE 205-10
  205.4.16 SETROWTAG 205-10
  205.4.17 SETSQLTOXMLNAMEESCAPING 205-11
  205.4.18 UPDATECOLUMN 205-11
  205.4.19 SETXSLT 205-11
  205.4.20 SETXSLTPARAM 205-12
  205.4.21 UPDATEXML 205-13

### DBMS_XMLSCHEMA

206.1 DBMS_XMLSCHEMA Overview 206-1
206.2 DBMS_XMLSCHEMA Security Model 206-1
206.3 DBMS_XMLSCHEMA Constants 206-2
206.4 Views 206-3
206.5 DBMS_XMLSCHEMA Operational Notes 206-4
206.6 Summary of DBMS_XMLSCHEMA Subprograms 206-4
  206.6.1 COMPILESHEMA Procedure 206-5
  206.6.2 COPYEVOLVE Procedure 206-5
  206.6.3 DELETESHEMA Procedure 206-8
  206.6.4 INPLACEEVOLVE Procedure 206-9
  206.6.5 PURGESHEMA Procedure 206-10
  206.6.6 REGISTERSHEMA Procedures 206-11
207  DBMS_XMLSCHEMA_ANNOTATE

207.1  DBMS_XMLSCHEMA_ANNOTATE Overview  207-1
207.2  DBMS_XMLSCHEMA_ANNOTATE Security Model  207-1
207.3  Summary of DBMS_XMLSCHEMA_ANNOTATE Subprograms  207-1
  207.3.1  ADDBNAMESPACE Procedure  207-3
  207.3.2  DISABLEDEFAULTTABLECREATION Procedure  207-3
  207.3.3  DISABLEMAINTAINDOM Procedure  207-4
  207.3.4  ENABLEDEFAULTTABLECREATION Procedure  207-5
  207.3.5  ENABLEMAINTAINDOM Procedure  207-5
  207.3.6  GETSCHEMAANNOTATIONS Function  207-6
  207.3.7  GETSIDXDEFFROMVIEW Function  207-7
  207.3.8  PRINTWARNINGS Procedure  207-7
  207.3.9  REMOVEANYSTORAGE Procedure  207-8
  207.3.10 REMOVEDEFAULTTABLE Procedure  207-8
  207.3.11 REMOVEMAINTAINDOM Procedure  207-9
  207.3.12 REMOVEOUTOFLINE Procedure  207-9
  207.3.13 REMOVESQLCOLLTYPE Procedure  207-10
  207.3.14 REMOVESQLNAME Procedure  207-11
  207.3.15 REMOVESQLTYPE Procedure  207-12
  207.3.16 REMOVESQLTYPEMAPPING Procedure  207-12
  207.3.17 REMOVETABLEPROPS Procedure  207-13
  207.3.18 REMOVETIMESTAMPWITHTIMEZONE Procedure  207-14
  207.3.19 SETANYSTORAGE Procedure  207-14
  207.3.20 SETDEFAULTTABLE Procedure  207-15
  207.3.21 SETOUTOFLINE Procedure  207-15
  207.3.22 SETSCHEMAANNOTATIONS Procedure  207-17
  207.3.23 SETSQLCOLLTYPE Procedure  207-18
  207.3.24 SETSQLNAME Procedure  207-19
  207.3.25 SETSQLTYPE Procedure  207-20
  207.3.26 SETSQLTYPEMAPPING Procedure  207-21
  207.3.27 SETTABLEPROPS Procedure  207-22
  207.3.28 SETTIMESTAMPWITHTIMEZONE Procedure  207-23

208  DBMS_XMLSTORAGE_MANAGE

208.1  DBMS_XMLSTORAGE_MANAGE Overview  208-1
208.2  DBMS_XMLSTORAGE_MANAGE Security Model  208-1
208.3  Summary of DBMS_XMLSTORAGE_MANAGE Subprograms  208-1
<table>
<thead>
<tr>
<th>Section</th>
<th>Subprogram Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>208.3.1</td>
<td>DISABLEINDEXESANDCONSTRAINTS Procedure</td>
</tr>
<tr>
<td>208.3.2</td>
<td>ENABLEINDEXESANDCONSTRAINTS Procedure</td>
</tr>
<tr>
<td>208.3.3</td>
<td>EXCHANGEPOSTPROC Procedure</td>
</tr>
<tr>
<td>208.3.4</td>
<td>EXCHANGEPREPROC Procedure</td>
</tr>
<tr>
<td>208.3.5</td>
<td>INDEXXMLREFERENCES Procedure</td>
</tr>
<tr>
<td>208.3.6</td>
<td>REFPARTITIONEXCHANGEIN Procedure</td>
</tr>
<tr>
<td>208.3.7</td>
<td>REFPARTITIONEXCHANGEOUT Procedure</td>
</tr>
<tr>
<td>208.3.8</td>
<td>RENAMECOLLECTIONTABLE Procedure</td>
</tr>
<tr>
<td>208.3.9</td>
<td>SCOPEXMLREFERENCES Procedure</td>
</tr>
<tr>
<td>208.3.10</td>
<td>XPATH2TABCOLMAPPING Function</td>
</tr>
<tr>
<td>209</td>
<td>DBMS_XMLSTORE</td>
</tr>
<tr>
<td>209.1</td>
<td>DBMS_XMLSTORE Security Model</td>
</tr>
<tr>
<td>209.2</td>
<td>Types</td>
</tr>
<tr>
<td>209.3</td>
<td>Summary of DBMS_XMLSTORE Subprograms</td>
</tr>
<tr>
<td>209.3.1</td>
<td>CLEARKEYCOLUMNLIST</td>
</tr>
<tr>
<td>209.3.2</td>
<td>CLEARUPDATECOLUMNLIST</td>
</tr>
<tr>
<td>209.3.3</td>
<td>CLOSECONTEXT</td>
</tr>
<tr>
<td>209.3.4</td>
<td>DELETEXML</td>
</tr>
<tr>
<td>209.3.5</td>
<td>INSERTXML</td>
</tr>
<tr>
<td>209.3.6</td>
<td>NEWCONTEXT</td>
</tr>
<tr>
<td>209.3.7</td>
<td>SETKEYCOLUMN</td>
</tr>
<tr>
<td>209.3.8</td>
<td>SETROWTAG</td>
</tr>
<tr>
<td>209.3.9</td>
<td>SETUPDATECOLUMN</td>
</tr>
<tr>
<td>209.3.10</td>
<td>UPDATEXML</td>
</tr>
<tr>
<td>210</td>
<td>DBMS_XPLAN</td>
</tr>
<tr>
<td>210.1</td>
<td>DBMS_XPLAN Overview</td>
</tr>
<tr>
<td>210.2</td>
<td>DBMS_XPLAN Security Model</td>
</tr>
<tr>
<td>210.3</td>
<td>Examples</td>
</tr>
<tr>
<td>210.4</td>
<td>Summary of DBMS_XPLAN Subprograms</td>
</tr>
<tr>
<td>210.4.1</td>
<td>DIFF_PLAN Function</td>
</tr>
<tr>
<td>210.4.2</td>
<td>DISPLAY Function</td>
</tr>
<tr>
<td>210.4.3</td>
<td>DISPLAY_AWR Function</td>
</tr>
<tr>
<td>210.4.4</td>
<td>DISPLAY_CURSOR Function</td>
</tr>
<tr>
<td>210.4.5</td>
<td>DISPLAY_PLAN Function</td>
</tr>
<tr>
<td>210.4.6</td>
<td>DISPLAY_SQL_PLAN_BASELINE Function</td>
</tr>
<tr>
<td>210.4.7</td>
<td>DISPLAY_SQLSET Function</td>
</tr>
</tbody>
</table>
211 DBMS_XSLPROCESSOR

211.1 DBMS_XSLPROCESSOR Overview 211-1
211.2 DBMS_XSLPROCESSOR Deprecated Subprograms 211-2
211.3 DBMS_XSLPROCESSOR Security Model 211-2
211.4 Summary of DBMS_XSLPROCESSOR Subprograms 211-2
  211.4.1 CLOB2FILE Procedure 211-3
  211.4.2 FREEPROCESSOR Procedure 211-4
  211.4.3 FREESTYLESHEET Procedure 211-4
  211.4.4 NEWPROCESSOR Function 211-4
  211.4.5 NEWSTYLESHEET Functions 211-4
  211.4.6 PROCESSXSL Functions and Procedures 211-5
  211.4.7 READ2CLOB Function 211-7
  211.4.8 REMOVEPARAM Procedure 211-8
  211.4.9 RESETPARAMS Procedure 211-8
  211.4.10 SELECTNODES Function 211-9
  211.4.11 SELECTSINGLENODE Function 211-9
  211.4.12 SETERRORLOG Procedure 211-10
  211.4.13 SETPARAM Procedure 211-10
  211.4.14 SHOWWARNINGS Procedure 211-11
  211.4.15 TRANSFORMNODE Function 211-11
  211.4.16 VALUEOF Function and Procedure 211-11

212 DBMS_XSTREAM_ADM

212.1 DBMS_XSTREAM_ADM Overview 212-1
212.2 DBMS_XSTREAM_ADM Security Model 212-2
212.3 DBMS_XSTREAM_ADM Operational Notes 212-3
212.4 Summary of DBMS_XSTREAM_ADM Subprograms 212-3
  212.4.1 ADD_COLUMN Procedure 212-5
  212.4.2 ADD_GLOBAL_PROPAGATION_RULES Procedure 212-8
  212.4.3 ADD_GLOBAL_RULES Procedure 212-12
  212.4.4 ADD_OUTBOUND Procedure 212-17
  212.4.5 ADD_SCHEMA_PROPAGATION_RULES Procedure 212-21
  212.4.6 ADD_SCHEMA_RULES Procedure 212-26
  212.4.7 ADD_SUBSET_OUTBOUND_RULES Procedure 212-31
  212.4.8 ADD_SUBSETPROPAGATION_RULES Procedure 212-33
  212.4.9 ADD_SUBSET_RULES Procedure 212-37
  212.4.10 ADD_TABLE_PROPAGATION_RULES Procedure 212-42
  212.4.11 ADD_TABLE_RULES Procedure 212-46
  212.4.12 ALTER_INBOUND Procedure 212-52
  212.4.13 ALTER_OUTBOUND Procedure 212-53
212.4.14 CREATE_INBOUND Procedure 212-58
212.4.15 CREATE_OUTBOUND Procedure 212-60
212.4.16 DELETE_COLUMN Procedure 212-65
212.4.17 DROP_INBOUND Procedure 212-68
212.4.18 DROP_OUTBOUND Procedure 212-68
212.4.19 ENABLE_GG_XSTREAM_FOR_STREAMS Procedure 212-69
212.4.20 GET_MESSAGE_TRACKING Function 212-70
212.4.21 GET_TAG Function 212-71
212.4.22 IS_GG_XSTREAM_FOR_STREAMS Function 212-71
212.4.23 KEEP_COLUMNS Procedure 212-72
212.4.24 MERGE_STREAMS Procedure 212-74
212.4.25 MERGE_STREAMS_JOB Procedure 212-77
212.4.26 PURGE_SOURCE_CATALOG Procedure 212-80
212.4.27 RECOVER_OPERATION Procedure 212-81
212.4.28 REMOVE_QUEUE Procedure 212-83
212.4.29 REMOVE_RULE Procedure 212-84
212.4.30 REMOVE_SUBSET_OUTBOUND_RULES Procedure 212-85
212.4.31 REMOVE_XSTREAM_CONFIGURATION Procedure 212-86
212.4.32 RENAME_COLUMN Procedure 212-88
212.4.33 RENAME_SCHEMA Procedure 212-91
212.4.34 RENAME_TABLE Procedure 212-93
212.4.35 SET_MESSAGE_TRACKING Procedure 212-95
212.4.36 SET_PARAMETER Procedure 212-96
212.4.37 SET_TAG Procedure 212-97
212.4.38 SET_UP_QUEUE Procedure 212-98
212.4.39 SPLIT_STREAMS Procedure 212-100
212.4.40 START_OUTBOUND Procedure 212-104
212.4.41 STOP_OUTBOUND Procedure 212-105

213 DBMS_XSTREAM_AUTH

213.1 DBMS_XSTREAM_AUTH Overview 213-1
213.2 DBMS_XSTREAM_AUTH Security Model 213-1
213.3 Summary of DBMS_XSTREAM_AUTH Subprograms 213-1
    213.3.1 GRANT_ADMIN_PRIVILEGE Procedure 213-2
    213.3.2 GRANT_REMOTE_ADMIN_ACCESS Procedure 213-6
    213.3.3 REVOKE_ADMIN_PRIVILEGE Procedure 213-6
    213.3.4 REVOKE_REMOTE_ADMIN_ACCESS Procedure 213-9
214 DEBUG_EXTPROC

214.1 DEBUG_EXTPROC Security Model 214-1
214.2 DEBUG_EXTPROC Operational Notes 214-1
214.3 Rules and Limits 214-2
214.4 Summary of DEBUG_EXTPROC Subprograms 214-2
  214.4.1 STARTUP_EXTPROC_AGENT Procedure 214-2

215 HTF

215.1 HTF Deprecated Subprograms 215-1
215.2 HTF Operational Notes 215-2
215.3 Rules and Limits 215-2
215.4 HTF Examples 215-2
215.5 Summary of Tags 215-2
215.6 Summary of HTF Subprograms 215-6
  215.6.1 ADDRESS Function 215-12
  215.6.2 ANCHOR Function 215-12
  215.6.3 ANCHOR2 Function 215-13
  215.6.4 APPLETCLOSE Function 215-14
  215.6.5 APPLETOPEN Function 215-14
  215.6.6 AREA Function 215-15
  215.6.7 BASE Function 215-15
  215.6.8 BASEFONT Function 215-16
  215.6.9 BGSOUND Function 215-16
  215.6.10 BIG Function 215-17
  215.6.11 BLOCKQUOTECLOSE Function 215-17
  215.6.12 BLOCKQUOTEOPEN Function 215-18
  215.6.13 BODYCLOSE Function 215-18
  215.6.14 BODYOPEN Function 215-19
  215.6.15 BOLD Function 215-19
  215.6.16 BR Function 215-20
  215.6.17 CENTER Function 215-20
  215.6.18 CENTERCLOSE Function 215-21
  215.6.19 CENTEROPEN Function 215-21
  215.6.20 CITE Function 215-22
  215.6.21 CODE Function 215-22
  215.6.22 COMMENT Function 215-23
  215.6.23 DFN Function 215-23
  215.6.24 DIRLISTCLOSE Function 215-24
  215.6.25 DIRLISTOPEN Function 215-24
  215.6.26 DIV Function 215-25
215.6.68  ISINDEX Function  215-49
215.6.69  ITALIC Function  215-50
215.6.70  KBD Function  215-50
215.6.71  KEYBOARD Function  215-51
215.6.72  LINE Function  215-51
215.6.73  LINKREL Function  215-52
215.6.74  LINKREV Function  215-53
215.6.75  LISTHEADER Function  215-53
215.6.76  LISTINGCLOSE Function  215-54
215.6.77  LISTINGOPEN Function  215-54
215.6.78  LISTITEM Function  215-54
215.6.79  MAILTO Function  215-55
215.6.80  MAPCLOSE Function  215-56
215.6.81  MAPOPEN Function  215-56
215.6.82  MENULISTCLOSE Function  215-57
215.6.83  MENULISTOPEN Function  215-57
215.6.84  META Function  215-57
215.6.85  NL Function  215-58
215.6.86  NOBR Function  215-59
215.6.87  NOFRAMESCLOSE Function  215-59
215.6.88  NOFRAMESOPEN Function  215-59
215.6.89  OLISTCLOSE Function  215-60
215.6.90  OLISTOPEN Function  215-60
215.6.91  PARA Function  215-61
215.6.92  PARAGRAPH Function  215-61
215.6.93  PARAM Function  215-62
215.6.94  PLAINTEXT Function  215-62
215.6.95  PRECLOSE Function  215-63
215.6.96  PREOPEN Function  215-63
215.6.97  PRINT Functions  215-64
215.6.98  PRN Functions  215-64
215.6.99  S Function  215-65
215.6.100  SAMPLE Function  215-66
215.6.101  SCRIPT Function  215-66
215.6.102  SMALL Function  215-67
215.6.103  STRIKE Function  215-67
215.6.104  STRONG Function  215-68
215.6.105  STYLE Function  215-68
215.6.106  SUB Function  215-69
215.6.107  SUP Function  215-70
215.6.108  TABLECAPTION Function  215-70
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>215.6.109</td>
<td>TABLECLOSE Function</td>
<td>215-71</td>
</tr>
<tr>
<td>215.6.110</td>
<td>TABLEDATA Function</td>
<td>215-71</td>
</tr>
<tr>
<td>215.6.111</td>
<td>TABLEHEADER Function</td>
<td>215-72</td>
</tr>
<tr>
<td>215.6.112</td>
<td>TABLEOPEN Function</td>
<td>215-73</td>
</tr>
<tr>
<td>215.6.113</td>
<td>TABLEROWCLOSE Function</td>
<td>215-73</td>
</tr>
<tr>
<td>215.6.114</td>
<td>TABLEROWOPEN Function</td>
<td>215-74</td>
</tr>
<tr>
<td>215.6.115</td>
<td>TELETYP E Function</td>
<td>215-74</td>
</tr>
<tr>
<td>215.6.116</td>
<td>TITLE Function</td>
<td>215-75</td>
</tr>
<tr>
<td>215.6.117</td>
<td>ULISTCLOSE Function</td>
<td>215-75</td>
</tr>
<tr>
<td>215.6.118</td>
<td>ULISTOPEN Function</td>
<td>215-76</td>
</tr>
<tr>
<td>215.6.119</td>
<td>UNDERLINE Function</td>
<td>215-76</td>
</tr>
<tr>
<td>215.6.120</td>
<td>VARIABLE Function</td>
<td>215-77</td>
</tr>
<tr>
<td>215.6.121</td>
<td>WBR Function</td>
<td>215-77</td>
</tr>
</tbody>
</table>

216  

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>216.1</td>
<td>HTP Operational Notes</td>
<td>216-1</td>
</tr>
<tr>
<td>216.2</td>
<td>HTP Rules and Limits</td>
<td>216-2</td>
</tr>
<tr>
<td>216.3</td>
<td>HTP Examples</td>
<td>216-2</td>
</tr>
<tr>
<td>216.4</td>
<td>HTP Summary of Tags</td>
<td>216-2</td>
</tr>
<tr>
<td>216.5</td>
<td>Summary of HTP Subprograms</td>
<td>216-5</td>
</tr>
<tr>
<td>216.5.1</td>
<td>ADDRESS Procedure</td>
<td>216-11</td>
</tr>
<tr>
<td>216.5.2</td>
<td>ANCHOR Procedure</td>
<td>216-12</td>
</tr>
<tr>
<td>216.5.3</td>
<td>ANCHOR2 Procedure</td>
<td>216-13</td>
</tr>
<tr>
<td>216.5.4</td>
<td>APPLETCLOSE Procedure</td>
<td>216-13</td>
</tr>
<tr>
<td>216.5.5</td>
<td>APPLETOPEN Procedure</td>
<td>216-14</td>
</tr>
<tr>
<td>216.5.6</td>
<td>AREA Procedure</td>
<td>216-15</td>
</tr>
<tr>
<td>216.5.7</td>
<td>BASE Procedure</td>
<td>216-15</td>
</tr>
<tr>
<td>216.5.8</td>
<td>BASEFONT Procedure</td>
<td>216-16</td>
</tr>
<tr>
<td>216.5.9</td>
<td>BGSOUND Procedure</td>
<td>216-16</td>
</tr>
<tr>
<td>216.5.10</td>
<td>BIG Procedure</td>
<td>216-17</td>
</tr>
<tr>
<td>216.5.11</td>
<td>BLOCKQUOTECLOSE Procedure</td>
<td>216-17</td>
</tr>
<tr>
<td>216.5.12</td>
<td>BLOCKQUOTEOPEN Procedure</td>
<td>216-18</td>
</tr>
<tr>
<td>216.5.13</td>
<td>BODYCLOSE Procedure</td>
<td>216-18</td>
</tr>
<tr>
<td>216.5.14</td>
<td>BODYOPEN Procedure</td>
<td>216-19</td>
</tr>
<tr>
<td>216.5.15</td>
<td>BOLD Procedure</td>
<td>216-19</td>
</tr>
<tr>
<td>216.5.16</td>
<td>BR Procedure</td>
<td>216-20</td>
</tr>
<tr>
<td>216.5.17</td>
<td>CENTER Procedure</td>
<td>216-20</td>
</tr>
<tr>
<td>216.5.18</td>
<td>CENTERCLOSE Procedure</td>
<td>216-21</td>
</tr>
<tr>
<td>216.5.19</td>
<td>CENTEROPEN Procedure</td>
<td>216-21</td>
</tr>
<tr>
<td>216.5.20</td>
<td>CITE Procedure</td>
<td>216-22</td>
</tr>
</tbody>
</table>
216.5.21  CODE Procedure 216-22
216.5.22  COMMENT Procedure 216-23
216.5.23  DFN Procedure 216-23
216.5.24  DIRLISTCLOSE Procedure 216-23
216.5.25  DIRLISTOPEN Procedure 216-24
216.5.26  DIV Procedure 216-24
216.5.27  DLISTCLOSE Procedure 216-25
216.5.28  DLISTDEF Procedure 216-25
216.5.29  DLISTOPEN Procedure 216-26
216.5.30  DLISTTERM Procedure 216-27
216.5.31  EM Procedure 216-27
216.5.32  EMPHASIS Procedure 216-28
216.5.33  ESCAPE_SC Procedure 216-28
216.5.34  FONTCLOSE Procedure 216-29
216.5.35  FONTOPEN Procedure 216-29
216.5.36  FORMCHECKBOX Procedure 216-30
216.5.37  FORMCLOSE Procedure 216-30
216.5.38  FORMOPEN Procedure 216-31
216.5.39  FORMFILE Procedure 216-31
216.5.40  FORMHIDDEN Procedure 216-32
216.5.41  FORMIMAGE Procedure 216-32
216.5.42  FORMPASSWORD Procedure 216-33
216.5.43  FORMRADIO Procedure 216-34
216.5.44  FORMRESET Procedure 216-34
216.5.45  FORMSELECTCLOSE Procedure 216-35
216.5.46  FORMSELECTOPEN Procedure 216-35
216.5.47  FORMSELECTOPTION Procedure 216-36
216.5.48  FORMSUBMIT Procedure 216-37
216.5.49  FORMTEXT Procedure 216-37
216.5.50  FORMTEXTAREA Procedure 216-38
216.5.51  FORMTEXTAREA2 Procedure 216-39
216.5.52  FORMTEXTAREACLOSE Procedure 216-39
216.5.53  FORMTEXTAREAOPEN Procedure 216-40
216.5.54  FORMTEXTAREAOPEN2 Procedure 216-40
216.5.55  FRAME Procedure 216-41
216.5.56  FRAMESETCLOSE Procedure 216-42
216.5.57  FRAMESETOPEN Procedure 216-42
216.5.58  HEADCLOSE Procedure 216-43
216.5.59  HEADER Procedure 216-43
216.5.60  HEADOPEN Procedure 216-44
216.5.61  HR Procedure 216-44
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>216.5.62</td>
<td>HTMLCLOSE Procedure</td>
<td>216-45</td>
</tr>
<tr>
<td>216.5.63</td>
<td>HTMLOPEN Procedure</td>
<td>216-45</td>
</tr>
<tr>
<td>216.5.64</td>
<td>IMG Procedure</td>
<td>216-45</td>
</tr>
<tr>
<td>216.5.65</td>
<td>IMG2 Procedure</td>
<td>216-46</td>
</tr>
<tr>
<td>216.5.66</td>
<td>ISINDEX Procedure</td>
<td>216-47</td>
</tr>
<tr>
<td>216.5.67</td>
<td>ITALIC Procedure</td>
<td>216-47</td>
</tr>
<tr>
<td>216.5.68</td>
<td>KBD Procedure</td>
<td>216-48</td>
</tr>
<tr>
<td>216.5.69</td>
<td>KEYBOARD Procedure</td>
<td>216-48</td>
</tr>
<tr>
<td>216.5.70</td>
<td>LINE Procedure</td>
<td>216-49</td>
</tr>
<tr>
<td>216.5.71</td>
<td>LINKREL Procedure</td>
<td>216-49</td>
</tr>
<tr>
<td>216.5.72</td>
<td>LINKREV Procedure</td>
<td>216-50</td>
</tr>
<tr>
<td>216.5.73</td>
<td>LISTHEADER Procedure</td>
<td>216-50</td>
</tr>
<tr>
<td>216.5.74</td>
<td>LISTINGCLOSE Procedure</td>
<td>216-51</td>
</tr>
<tr>
<td>216.5.75</td>
<td>LISTINGOPEN Procedure</td>
<td>216-51</td>
</tr>
<tr>
<td>216.5.76</td>
<td>LISTITEM Procedure</td>
<td>216-52</td>
</tr>
<tr>
<td>216.5.77</td>
<td>MAILTO Procedure</td>
<td>216-52</td>
</tr>
<tr>
<td>216.5.78</td>
<td>MAPCLOSE Procedure</td>
<td>216-53</td>
</tr>
<tr>
<td>216.5.79</td>
<td>MAPOPEN Procedure</td>
<td>216-53</td>
</tr>
<tr>
<td>216.5.80</td>
<td>MENULISTCLOSE Procedure</td>
<td>216-54</td>
</tr>
<tr>
<td>216.5.81</td>
<td>MENULISTOPEN Procedure</td>
<td>216-54</td>
</tr>
<tr>
<td>216.5.82</td>
<td>META Procedure</td>
<td>216-55</td>
</tr>
<tr>
<td>216.5.83</td>
<td>NL Procedure</td>
<td>216-55</td>
</tr>
<tr>
<td>216.5.84</td>
<td>NOBR Procedure</td>
<td>216-56</td>
</tr>
<tr>
<td>216.5.85</td>
<td>NOFRAMESCLOSE Procedure</td>
<td>216-56</td>
</tr>
<tr>
<td>216.5.86</td>
<td>NOFRAMESOPEN Procedure</td>
<td>216-57</td>
</tr>
<tr>
<td>216.5.87</td>
<td>OLISTCLOSE Procedure</td>
<td>216-57</td>
</tr>
<tr>
<td>216.5.88</td>
<td>OLISTOPEN Procedure</td>
<td>216-57</td>
</tr>
<tr>
<td>216.5.89</td>
<td>PARA Procedure</td>
<td>216-58</td>
</tr>
<tr>
<td>216.5.90</td>
<td>PARAGRAPh Procedure</td>
<td>216-58</td>
</tr>
<tr>
<td>216.5.91</td>
<td>PARAM Procedure</td>
<td>216-59</td>
</tr>
<tr>
<td>216.5.92</td>
<td>PLAINTEXT Procedure</td>
<td>216-60</td>
</tr>
<tr>
<td>216.5.93</td>
<td>PRECLOSE Procedure</td>
<td>216-60</td>
</tr>
<tr>
<td>216.5.94</td>
<td>PREOPEN Procedure</td>
<td>216-60</td>
</tr>
<tr>
<td>216.5.95</td>
<td>PRINT Procedures</td>
<td>216-61</td>
</tr>
<tr>
<td>216.5.96</td>
<td>PRINTS Procedure</td>
<td>216-62</td>
</tr>
<tr>
<td>216.5.97</td>
<td>PRN Procedures</td>
<td>216-62</td>
</tr>
<tr>
<td>216.5.98</td>
<td>PS Procedure</td>
<td>216-63</td>
</tr>
<tr>
<td>216.5.99</td>
<td>S Procedure</td>
<td>216-64</td>
</tr>
<tr>
<td>216.5.100</td>
<td>SAMPLE Procedure</td>
<td>216-64</td>
</tr>
<tr>
<td>216.5.101</td>
<td>SCRIPT Procedure</td>
<td>216-65</td>
</tr>
<tr>
<td>216.5.102</td>
<td>SMALL Procedure</td>
<td>216-65</td>
</tr>
</tbody>
</table>
216.5.103  STRIKE Procedure  216-66
216.5.104  STRONG Procedure  216-66
216.5.105  STYLE Procedure  216-67
216.5.106  SUB Procedure  216-67
216.5.107  SUP Procedure  216-68
216.5.108  TABLECAPTION Procedure  216-68
216.5.109  TABLECLOSE Procedure  216-69
216.5.110  TABLEDATA Procedure  216-69
216.5.111  TABLEHEADER Procedure  216-70
216.5.112  TABLEOPEN Procedure  216-71
216.5.113  TABLEROWCLOSE Procedure  216-71
216.5.114  TABLEROWOPEN Procedure  216-71
216.5.115  TELETYPE Procedure  216-72
216.5.116  TITLE Procedure  216-73
216.5.117  ULISTCLOSE Procedure  216-73
216.5.118  ULISTOPEN Procedure  216-74
216.5.119  UNDERLINE Procedure  216-74
216.5.120  VARIABLE Procedure  216-75
216.5.121  WBR Procedure  216-75

217  ORD_AUDIO

218  ORD_DICOM

219  ORD_DICOM_ADMIN

220  ORD_DOC

221  ORD_IMAGE
## OWA_CACHE

233.1 OWA_CACHE Constants 233-1
233.2 Summary of OWA_CACHE Subprograms 233-1
  233.2.1 DISABLE Procedure 233-2
  233.2.2 GET_ETAG Function 233-2
  233.2.3 GET_LEVEL Function 233-2
  233.2.4 SET_CACHE Procedure 233-2
  233.2.5 SET_EXPIRES Procedure 233-3
  233.2.6 SET_NOT_MODIFIED Procedure 233-3
  233.2.7 SET_SURROGATE_CONTROL Procedure 233-4

## OWA_COOKIE

234.1 OWA_CUSTOM Overview 234-1
234.2 OWA_COOKIE Types 234-1
234.3 OWA_COOKIE Rules and Limits 234-1
234.4 Summary of OWA_COOKIE Subprograms 234-2
  234.4.1 GET Function 234-2
  234.4.2 GET_ALL Procedure 234-2
  234.4.3 REMOVE Procedure 234-3
  234.4.4 SEND procedure 234-3

## OWA_CUSTOM

235.1 OWA_CUSTOM Constants 235-1
235.2 Summary of OWA_CUSTOM Subprograms 235-1
  235.2.1 AUTHORIZE Function 235-1

## OWA_IMAGE

236.1 OWA_IMAGE Overview 236-1
236.2 OWA_IMAGE Types 236-1
236.3 OWA_IMAGE Variables 236-1
236.4 OWA_IMAGE Examples 236-2
236.5 Summary of OWA_IMAGE Subprograms 236-2
  236.5.1 GET_X Function 236-2
230.2 Summary of OWA_TEXT Subprograms

230.2.1 ADD2MULTI Procedure

230.2.2 NEW_ROW_LIST Function and Procedure

230.2.3 PRINT_MULTI Procedure

230.2.4 PRINT_ROW_LIST Procedure

230.2.5 STREAM2MULTI Procedure

231 OWA_UTIL

231.1 OWA_UTIL Overview

231.2 OWA_UTIL Types

231.2.1 DATETYP Datatype

231.2.2 IDENT_ARR Datatype

231.2.3 IP_ADDRESS Datatype

231.3 Summary of OWA_UTIL Subprograms

231.3.1 BIND_VARIABLES Function

231.3.2 CALENDARPRINT Procedures

231.3.3 CELLSPRINT Procedures

231.3.4 CHOOSE_DATE Procedure

231.3.5 GET_CGI_ENV Function

231.3.6 GET_OWA_SERVICE_PATH Function

231.3.7 GET_PROCEDURE Function

231.3.8 HTTP_HEADER_CLOSE Procedure

231.3.9 LISTPRINT Procedure

231.3.10 MIME_HEADER Procedure

231.3.11 PRINT_CGI_ENV Procedure

231.3.12 REDIRECT_URL Procedure

231.3.13 SHOWPAGE Procedure

231.3.14 SHOWSOURCE Procedure

231.3.15 SIGNATURE procedure

231.3.16 STATUS_LINE Procedure

231.3.17 TABLEPRINT Function

231.3.18 TODATE Function

231.3.19 WHO_CALLED_ME Procedure

232 SDO_CS
233 SDO_CSW_PROCESS
234 SDO_GCDR
235 SDO_GEOM
236 SDO_GEOR
237 SDO_GEOR_ADMIN
238 SDO_GEOR_AGGR
239 SDO_GEOR_RA
240 SDO_GEOR_UTL
241 SDO_LRS
242 SDO_MIGRATE
243 SDO_NET
244  SDO_NFE
245  SDO_OLS
246  SDO_PC_PKG
247  SDO_SAM
248  SDO_TIN_PKG
249  SDO_TOPO
250  SDO_TOPO_MAP
251  SDO_TUNE
252  SDO_UTIL
253  SDO_WFS_LOCK
254  SDO_WFS_PROCESS
## UTL_CALL_STACK

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>260.1</td>
<td>UTL_CALL_STACK Overview</td>
<td>260-1</td>
</tr>
<tr>
<td>260.2</td>
<td>UTL_CALL_STACK Security Model</td>
<td>260-2</td>
</tr>
<tr>
<td>260.3</td>
<td>UTL_CALL_STACK Operational Notes</td>
<td>260-3</td>
</tr>
<tr>
<td>260.4</td>
<td>UTL_CALL_STACK Exceptions</td>
<td>260-3</td>
</tr>
<tr>
<td>260.5</td>
<td>UTL_CALL_STACK Data Structures</td>
<td>260-3</td>
</tr>
<tr>
<td>260.5.1</td>
<td>UNIT_QUALIFIED_NAME</td>
<td>260-3</td>
</tr>
<tr>
<td>260.6</td>
<td>Summary of UTL_CALL_STACK Subprograms</td>
<td>260-4</td>
</tr>
<tr>
<td>260.6.1</td>
<td>BACKTRACE_DEPTH Function</td>
<td>260-4</td>
</tr>
<tr>
<td>260.6.2</td>
<td>BACKTRACE_LINE Function</td>
<td>260-5</td>
</tr>
<tr>
<td>260.6.3</td>
<td>BACKTRACE_UNIT Function</td>
<td>260-5</td>
</tr>
<tr>
<td>260.6.4</td>
<td>CURRENT_EDITION Function</td>
<td>260-6</td>
</tr>
<tr>
<td>260.6.5</td>
<td>CONCATENATE_SUBPROGRAM Function</td>
<td>260-6</td>
</tr>
<tr>
<td>260.6.6</td>
<td>DYNAMIC_DEPTH Function</td>
<td>260-6</td>
</tr>
<tr>
<td>260.6.7</td>
<td>ERROR_DEPTH Function</td>
<td>260-7</td>
</tr>
<tr>
<td>260.6.8</td>
<td>ERROR_MSG Function</td>
<td>260-7</td>
</tr>
<tr>
<td>260.6.9</td>
<td>ERROR_NUMBER Function</td>
<td>260-7</td>
</tr>
<tr>
<td>260.6.10</td>
<td>LEXICAL_DEPTH Function</td>
<td>260-8</td>
</tr>
<tr>
<td>260.6.11</td>
<td>OWNER Function</td>
<td>260-8</td>
</tr>
<tr>
<td>260.6.12</td>
<td>UNIT_LINE Function</td>
<td>260-9</td>
</tr>
<tr>
<td>260.6.13</td>
<td>SUBPROGRAM Function</td>
<td>260-9</td>
</tr>
</tbody>
</table>
261  UTL_COLL

261.1  Summary of UTL_COLL Subprograms

261.1.1  IS_LOCATOR Function

262  UTL_COMPRESS

262.1  UTL_COMPRESS Constants

262.2  UTL_COMPRESS Exceptions

262.3  UTL_COMPRESS Operational Notes

262.4  Summary of UTL_COMPRESS Subprograms

262.4.1  ISOPEN Function

262.4.2  LZ_COMPRESS Functions and Procedures

262.4.3  LZ_COMPRESS_ADD Procedure

262.4.4  LZ_COMPRESS_CLOSE

262.4.5  LZ_COMPRESS_OPEN

262.4.6  LZ_UNCOMPRESS Functions and Procedures

262.4.7  LZ_UNCOMPRESS_EXTRACT Procedure

262.4.8  LZ_UNCOMPRESS_OPEN Function

262.4.9  LZ_UNCOMPRESS_CLOSE Procedure

263  UTL_ENCODE

263.1  Summary of UTL_ENCODE Subprograms

263.1.1  BASE64_DECODE Function

263.1.2  BASE64_ENCODE Function

263.1.3  MIMEHEADER_DECODE Function

263.1.4  MIMEHEADER_ENCODE Function

263.1.5  QUOTED_PRINTABLE_DECODE Function

263.1.6  QUOTED_PRINTABLE_ENCODE Function

263.1.7  TEXT_DECODE Function

263.1.8  TEXT_ENCODE Function

263.1.9  UUDECODE Function

263.1.10  UUENCODE Function

264  UTL_FILE

264.1  UTL_FILE Security Model

264.2  UTL_FILE Operational Notes

264.3  UTL_FILE Rules and Limits

264.4  UTL_FILE Exceptions

264.5  UTL_FILE Examples
264.6 UTL_FILE Data Structures
   264.6.1 FILETYPE Record Type

264.7 Summary of UTL_FILE Subprograms
   264.7.1 FCLOSE Procedure
   264.7.2 FCLOSE_ALL Procedure
   264.7.3 FCOPY Procedure
   264.7.4 FFLUSH Procedure
   264.7.5 FGETATTR Procedure
   264.7.6 FGETPOS Function
   264.7.7 FOPEN Function
   264.7.8 FOPEN_NCHAR Function
   264.7.9 FREMOVE Procedure
   264.7.10 FRENAM Procedure
   264.7.11 FSEEK Procedure
   264.7.12 GET_LINE Procedure
   264.7.13 GET_LINE_NCHAR Procedure
   264.7.14 GET_RAW Procedure
   264.7.15 IS_OPEN Function
   264.7.16 NEW_LINE Procedure
   264.7.17 PUT Procedure
   264.7.18 PUT_LINE Procedure
   264.7.19 PUT_LINE_NCHAR Procedure
   264.7.20 PUT_NCHAR Procedure
   264.7.21 PUTF Procedure
   264.7.22 PUTF_NCHAR Procedure
   264.7.23 PUT_RAW Procedure

265 UTL_HTTP

265.1 UTL_HTTP Overview
265.2 UTL_HTTP Security Model
265.3 UTL_HTTP Constants
265.4 UTL_HTTP Exceptions
265.5 UTL_HTTP Examples
   265.5.1 UTL_HTTP General Usage
   265.5.2 UTL_HTTP Retrieving HTTP Response Headers
   265.5.3 UTL_HTTP Handling HTTP Authentication
   265.5.4 UTL_HTTP Retrieving and Restoring Cookies
   265.5.5 UTL_HTTP Making HTTP Request with Private Wallet and Cookie Table
265.6 UTL_HTTP Data Structures
   265.6.1 REQ Type
265.6.2 REQUEST_CONTEXT_KEY Type
265.6.3 RESP Type
265.6.4 COOKIE and COOKIE_TABLE Types
265.6.5 CONNECTION Type

265.7 UTL_HTTP Operations
265.7.1 UTL_HTTP Operational Flow
265.7.2 UTL_HTTP Simple HTTP Fetches
265.7.3 UTL_HTTP HTTP Requests
265.7.4 UTL_HTTP HTTP Responses
265.7.5 UTL_HTTP HTTP Cookies
265.7.6 UTL_HTTP HTTP Persistent Connections
265.7.7 UTL_HTTP Error Conditions
265.7.8 UTL_HTTP Session Settings
265.7.9 UTL_HTTP Request Context
265.7.10 UTL_HTTP External Password Store

265.8 UTL_HTTP Subprogram Groups
265.8.1 UTL_HTTP Simple HTTP Fetches in a Single Call Subprograms
265.8.2 UTL_HTTP Session Settings Subprograms
265.8.3 UTL_HTTP HTTP Requests Subprograms
265.8.4 UTL_HTTP HTTP Request Contexts Subprograms
265.8.5 UTL_HTTP HTTP Responses Subprograms
265.8.6 UTL_HTTP HTTP Cookies Subprograms
265.8.7 UTL_HTTP HTTP Persistent Connections Subprograms
265.8.8 UTL_HTTP Error Conditions Subprograms

265.9 Summary of UTL_HTTP Subprograms
265.9.1 ADD_COOKIES Procedure
265.9.2 BEGIN_REQUEST Function
265.9.3 CLEAR_COOKIES Procedure
265.9.4 CLOSE_PERSISTENT_CONN Procedure
265.9.5 CLOSE_PERSISTENT_CONNS Procedure
265.9.6 CREATE_REQUEST_CONTEXT Function
265.9.7 DESTROY_REQUEST_CONTEXT Procedure
265.9.8 END_REQUEST Procedure
265.9.9 END_RESPONSE Procedure
265.9.10 GET_AUTHENTICATION Procedure
265.9.11 GET_BODY_CHARSET Procedure
265.9.12 GET_COOKIE_COUNT Function
265.9.13 GET_COOKIE_SUPPORT Procedure
265.9.14 GET_COOKIES Function
265.9.15 GET_DETAILED_EXCP_SUPPORT Procedure
265.9.16 GET_DETAILED_SQLCODE Function
266.4.3 GET_DEFAULT_CHARSET Function 266-5
266.4.4 GET_DEFAULT_ISO_CURRENCY Function 266-7
266.4.5 GET_DEFAULT_LINGUISTIC_SORT Function 266-7
266.4.6 GET_LOCALLANGUAGES Function 266-8
266.4.7 GET_LOCAL_LINGUISTIC_SORTS Function 266-9
266.4.8 GET_LOCAL_TERRITORIES Function 266-10
266.4.9 GET_LOCAL_TIME_ZONES Function 266-10
266.4.10 GET_MAX_CHARACTER_SIZE Function 266-12
266.4.11 GET_TRANSLATION Function 266-13
266.4.12 MAP_CHARSET Function 266-14
266.4.13 MAP_FROM_SHORT_LANGUAGE Function 266-15
266.4.14 MAP_LANGUAGE_FROM_ISO Function 266-16
266.4.15 MAP_LOCALE_TO_ISO Function 266-17
266.4.16 MAP_TERRITORY_FROM_ISO Function 266-18
266.4.17 MAP_TO_SHORT_LANGUAGE Function 266-18
266.4.18 RAW_TO_CHAR Functions 266-19
266.4.19 RAW_TO_NCHAR Functions 266-21
266.4.20 STRING_TO_RAW Function 266-23
266.4.21 TRANSLITERATE Function 266-24
266.4.22 UNESCAPE_REFERENCE Function 266-25
266.4.23 VALIDATE_CHARACTER_ENCODING Functions 266-26

267 UTL_IDENT
267.1 UTL_IDENT Overview 267-1
267.2 UTL_IDENT Security Model 267-1
267.3 UTL_IDENT Constants 267-1

268 UTL_INADDR
268.1 UTL_INADDR Security Model 268-1
268.2 UTL_INADDR Exceptions 268-1
268.3 UTL_INADDR Examples 268-1
268.4 Summary of UTL_INADDR Subprograms 268-2
   268.4.1 GET_HOST_ADDRESS Function 268-2
   268.4.2 GET_HOST_NAME Function 268-3

269 UTL_LMS
269.1 UTL_LMS Security Model 269-1
269.2 Summary of UTL_LMS Subprograms 269-1
   269.2.1 FORMAT_MESSAGE Function 269-1
270  UTL_MAIL

270.1  UTL_MAIL Security Model
270.2  UTL_MAIL Operational Notes
270.3  UTL_MAIL Rules and Limits
270.4  Summary of UTL_MAIL Subprograms
  270.4.1  SEND Procedure
  270.4.2  SEND_ATTACHMENT_RAW Procedure
  270.4.3  SEND_ATTACHMENT_VARCHAR2 Procedure

271  UTL_MATCH

271.1  UTL_MATCH Overview
271.2  UTL_MATCH Security Model
271.3  Summary of UTL_MATCH Subprograms
  271.3.1  EDIT_DISTANCE Function
  271.3.2  EDIT_DISTANCE_SIMILARITY Function
  271.3.3  JARO_WINKLER Function
  271.3.4  JARO_WINKLER_SIMILARITY Function

272  UTL_NLA

272.1  UTL_NLA Overview
272.2  UTL_NLA Rules and Limits
272.3  UTL_NLA Security Model
272.4  Subprogram Groups
  272.4.1  UTL_NLA BLAS Level 1 (Vector-Vector Operations) Subprograms
  272.4.2  UTL_NLA BLAS Level 2 (Matrix-Vector Operations) Subprograms
  272.4.3  UTL_NLA BLAS Level 3 (Matrix-Matrix Operations) Subprograms
  272.4.4  UTL_NLA LAPACK Driver Routines (Linear Equations) Subprograms
  272.4.5  UTL_NLA LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
272.5  Summary of UTL_NLA Subprograms
  272.5.1  BLAS_ASUM Functions
  272.5.2  BLAS_AXPY Procedures
  272.5.3  BLAS_COPY Procedures
  272.5.4  BLAS_DOT Functions
  272.5.5  BLAS_GBMV Procedures
  272.5.6  BLAS_GEMM Procedures
  272.5.7  BLAS_GEMV Procedures
<table>
<thead>
<tr>
<th>Section</th>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>272.5.8</td>
<td>BLAS_GER Procedures</td>
<td>272-23</td>
</tr>
<tr>
<td>272.5.9</td>
<td>BLAS_IAMAX Functions</td>
<td>272-25</td>
</tr>
<tr>
<td>272.5.10</td>
<td>BLAS_NRM2 Functions</td>
<td>272-25</td>
</tr>
<tr>
<td>272.5.11</td>
<td>BLAS_ROT Procedures</td>
<td>272-26</td>
</tr>
<tr>
<td>272.5.12</td>
<td>BLAS_ROTG Procedures</td>
<td>272-27</td>
</tr>
<tr>
<td>272.5.13</td>
<td>BLAS_SCAL Procedures</td>
<td>272-28</td>
</tr>
<tr>
<td>272.5.14</td>
<td>BLAS_SPMV Procedures</td>
<td>272-29</td>
</tr>
<tr>
<td>272.5.15</td>
<td>BLAS_SPR Procedures</td>
<td>272-30</td>
</tr>
<tr>
<td>272.5.16</td>
<td>BLAS_SPR2 Procedures</td>
<td>272-32</td>
</tr>
<tr>
<td>272.5.17</td>
<td>BLAS_SBMV Procedures</td>
<td>272-34</td>
</tr>
<tr>
<td>272.5.18</td>
<td>BLAS_SWAP Procedures</td>
<td>272-35</td>
</tr>
<tr>
<td>272.5.19</td>
<td>BLAS_SYMM Procedures</td>
<td>272-36</td>
</tr>
<tr>
<td>272.5.20</td>
<td>BLAS_SYMV Procedures</td>
<td>272-39</td>
</tr>
<tr>
<td>272.5.21</td>
<td>BLAS_SYR Procedures</td>
<td>272-40</td>
</tr>
<tr>
<td>272.5.22</td>
<td>BLAS_SYR2 Procedures</td>
<td>272-42</td>
</tr>
<tr>
<td>272.5.23</td>
<td>BLAS_SYR2K Procedures</td>
<td>272-43</td>
</tr>
<tr>
<td>272.5.24</td>
<td>BLAS_SYRK Procedures</td>
<td>272-46</td>
</tr>
<tr>
<td>272.5.25</td>
<td>BLAS_TBMV Procedures</td>
<td>272-48</td>
</tr>
<tr>
<td>272.5.26</td>
<td>BLAS_TBSV Procedures</td>
<td>272-50</td>
</tr>
<tr>
<td>272.5.27</td>
<td>BLAS_TPMV Procedures</td>
<td>272-52</td>
</tr>
<tr>
<td>272.5.28</td>
<td>BLAS_TPSV Procedures</td>
<td>272-53</td>
</tr>
<tr>
<td>272.5.29</td>
<td>BLAS_TRMM Procedures</td>
<td>272-55</td>
</tr>
<tr>
<td>272.5.30</td>
<td>BLAS_TRMV Procedures</td>
<td>272-57</td>
</tr>
<tr>
<td>272.5.31</td>
<td>BLAS_TRSM Procedures</td>
<td>272-58</td>
</tr>
<tr>
<td>272.5.32</td>
<td>BLAS_TRSV Procedures</td>
<td>272-61</td>
</tr>
<tr>
<td>272.5.33</td>
<td>LAPACK_GBSV Procedures</td>
<td>272-62</td>
</tr>
<tr>
<td>272.5.34</td>
<td>LAPACK_GEES Procedures</td>
<td>272-64</td>
</tr>
<tr>
<td>272.5.35</td>
<td>LAPACK_GELS Procedures</td>
<td>272-66</td>
</tr>
<tr>
<td>272.5.36</td>
<td>LAPACK_GESDD Procedures</td>
<td>272-68</td>
</tr>
<tr>
<td>272.5.37</td>
<td>LAPACK_GESV Procedure</td>
<td>272-70</td>
</tr>
<tr>
<td>272.5.38</td>
<td>LAPACK_GESVD Procedures</td>
<td>272-71</td>
</tr>
<tr>
<td>272.5.39</td>
<td>LAPACK_GEEV Procedures</td>
<td>272-74</td>
</tr>
<tr>
<td>272.5.40</td>
<td>LAPACK_GTSV Procedure</td>
<td>272-76</td>
</tr>
<tr>
<td>272.5.41</td>
<td>LAPACK_PBSV Procedures</td>
<td>272-78</td>
</tr>
<tr>
<td>272.5.42</td>
<td>LAPACK_POSV Procedures</td>
<td>272-79</td>
</tr>
<tr>
<td>272.5.43</td>
<td>LAPACK_PPSV Procedures</td>
<td>272-81</td>
</tr>
<tr>
<td>272.5.44</td>
<td>LAPACK_PTSV Procedures</td>
<td>272-83</td>
</tr>
<tr>
<td>272.5.45</td>
<td>LAPACK_SBEV Procedures</td>
<td>272-84</td>
</tr>
<tr>
<td>272.5.46</td>
<td>LAPACK_SBEVD Procedures</td>
<td>272-86</td>
</tr>
<tr>
<td>272.5.47</td>
<td>LAPACK_SPEV Procedures</td>
<td>272-88</td>
</tr>
<tr>
<td>272.5.48</td>
<td>LAPACK_SPEVD Procedures</td>
<td>272-89</td>
</tr>
</tbody>
</table>
273  UTL_RAW

273.1  UTL_RAW Overview
273.2  UTL_RAW Operational Notes
273.3  Summary of UTL_RAW Subprograms

273.3.1  BIT_AND Function
273.3.2  BIT_COMPLEMENT Function
273.3.3  BIT_OR Function
273.3.4  BIT_XOR Function
273.3.5  CAST_FROM_BINARY_DOUBLE Function
273.3.6  CAST_FROM_BINARY_FLOAT Function
273.3.7  CAST_FROM_BINARY_INTEGER Function
273.3.8  CAST_FROM_NUMBER Function
273.3.9  CAST_TO_BINARY_DOUBLE Function
273.3.10 CAST_TO_BINARY_FLOAT Function
273.3.11 CAST_TO_BINARY_INTEGER Function
273.3.12 CAST_TO_NUMBER Function
273.3.13 CAST_TO_NVARCHAR2 Function
273.3.14 CAST_TO_RAW Function
273.3.15 CAST_TO_VARCHAR2 Function
273.3.16 COMPARE Function
273.3.17 CONCAT Function
273.3.18 CONVERT Function
273.3.19 COPIES Function
273.3.20 LENGTH Function
273.3.21 OVERLAY Function
273.3.22 REVERSE Function
273.3.23 SUBSTR Function
273.3.24 TRANSFORM Function
273.3.25 TRANSLITERATE Function
273.3.26 X RANGE Function
276.10.10 MAIL Function and Procedure 276-16
276.10.11 NOOP Function and Procedure 276-17
276.10.12 OPEN_CONNECTION Functions 276-18
276.10.13 OPEN_DATA Function and Procedure 276-19
276.10.14 QUIT Function and Procedure 276-20
276.10.15 RCPT Function 276-21
276.10.16 RSET Function and Procedure 276-22
276.10.17 STARTTLS Function and Procedure 276-23
276.10.18 VRFY Function 276-24
276.10.19 WRITE_DATA Procedure 276-25
276.10.20 WRITE_RAW_DATA Procedure 276-26

277 UTL_SPADV

277.1 UTL_SPADV Overview 277-1
277.2 DBMS_COMPARISON Security Model 277-1
277.3 UTL_SPADV Operational Notes 277-2
277.4 Summary of UTL_SPADV Subprograms 277-11
  277.4.1 ALTER_MONITORING Procedure 277-12
  277.4.2 COLLECT_STATS Procedure 277-13
  277.4.3 IS_MONITORING Function 277-15
  277.4.4 SHOW_STATS Procedure 277-15
  277.4.5 SHOW_STATS_HTML Procedure 277-18
  277.4.6 START_MONITORING Procedure 277-20
  277.4.7 STOP_MONITORING Procedure 277-22

278 UTL_TCP

278.1 UTL_TCP Overview 278-1
278.2 UTL_TCP Security Model 278-1
278.3 UTL_TCP Types 278-1
278.4 UTL_TCP Exceptions 278-3
278.5 UTL_TCP Rules and Limits 278-3
278.6 UTL_TCP Examples 278-4
278.7 Summary of UTL_TCP Subprograms 278-5
  278.7.1 AVAILABLE Function 278-5
  278.7.2 CLOSE_ALL_CONNECTIONS Procedure 278-7
  278.7.3 CLOSE_CONNECTION Procedure 278-7
  278.7.4 FLUSH Procedure 278-7
  278.7.5 GET_LINE Function 278-8
  278.7.6 GET_LINE_NCHAR Function 278-9
## ANYDATASET TYPE

282.1 ANYDATASET TYPE Construction 282-1

282.2 Summary of ANYDATASET TYPE Subprograms 282-1

- 282.2.1 ADDINSTANCE Member Procedure 282-2
- 282.2.2 BEGINCREATE Static Procedure 282-3
- 282.2.3 ENDCREATE Member Procedure 282-3
- 282.2.4 GET* Member Functions 282-3
- 282.2.5 GETCOUNT Member Function 282-6
- 282.2.6 GETINSTANCE Member Function 282-7
- 282.2.7 GETTYPE Member Function 282-7
- 282.2.8 GETTYPENAME Member Function 282-8
- 282.2.9 PIECEWISE Member Procedure 282-8
- 282.2.10 SET* Member Procedures 282-9

## ANYTYPE TYPE

283.1 Summary of ANYTYPE Subprograms 283-1

- 283.1.1 BEGINCREATE Static Procedure 283-1
- 283.1.2 SETINFO Member Procedure 283-2
- 283.1.3 ADDATTR Member Procedure 283-3
- 283.1.4 ENDCREATE Member Procedure 283-3
- 283.1.5 GETPERSISTENT Static Function 283-5
- 283.1.6 GETINFO Member Function 283-5
- 283.1.7 GETATTRELEMINFO Member Function 283-6

## Oracle Database Advanced Queuing (AQ) Types

284.1 Advanced Queuing (AQ) Types Security Model 284-1

284.2 Oracle Database Advanced Queuing (AQ) Types — Summary of Types 284-1

- 284.2.1 AQ$_AGENT Type 284-2
- 284.2.2 AQ$_AGENT_LIST_T Type 284-2
- 284.2.3 AQ$_DESCRIPTOR Type 284-3
- 284.2.4 AQ$_NTFN_DESCRIPTOR Type 284-3
- 284.2.5 AQ$_NTFN_MSGID_ARRAY Type 284-4
- 284.2.6 AQ$_POST_INFO Type 284-4
- 284.2.7 AQ$_POST_INFO_LIST Type 284-5
- 284.2.8 AQ$_PURGE_OPTIONS_T Type 284-5
- 284.2.9 AQ$_RECIPIENT_LIST_T Type 284-6
- 284.2.10 AQ$_REG_INFO Type 284-6
- 284.2.11 AQ$_REG_INFO_LIST Type 284-9
- 284.2.12 AQ$_SUBSCRIBER_LIST_T Type 284-9
284.2.13 DEQUEUE_OPTIONS_T Type 284-9
284.2.14 ENQUEUE_OPTIONS_T Type 284-12
284.2.15 QUEUE_PROPS_T Type 284-13
284.2.16 SYS.MSG_PROP_T Type 284-13
284.2.17 MESSAGE_PROPERTIES_T Type 284-16
284.2.18 MESSAGE_PROPERTIES_ARRAY_T Type 284-19
284.2.19 MSGID_ARRAY_T Type 284-20

285 **DBFS Content Interface Types**

285.1 DDBFS Content Interface Types Overview 285-1
285.2 DBFS Content Interface Types Security Model 285-1
285.3 DBFS Content Interface Types —Data Structures 285-1
  285.3.1 DBMS_DBFS_CONTENT_CONTEXT_T Object Type 285-2
  285.3.2 DBMS_DBFS_CONTENT_LIST_ITEM_T Object Type 285-2
  285.3.3 DBMS_DBFS_CONTENT_PROPERTY_T Object Type 285-3
  285.3.4 DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type 285-3
  285.3.5 DBMS_DBFS_CONTENT_PROPERTIES_T Table Type 285-4
  285.3.6 DBMS_DBFS_CONTENT_RAW_T Table Type 285-4

286 **Database URI TYPES**

286.1 Summary of URITYPE Supertype Subprograms 286-1
  286.1.1 UriType Supertype GETBLOB 286-2
  286.1.2 UriType Supertype GETCLOB 286-3
  286.1.3 UriType Supertype GETCONTENTTYPE 286-3
  286.1.4 UriType Supertype GETEXTERNALURL 286-3
  286.1.5 UriType Supertype GETURL 286-4
  286.1.6 UriType Supertype GETXML 286-4

286.2 Summary of HTTPURITYPE Subtype Subprograms 286-4
  286.2.1 HttpUriType Subtype CREATEURI 286-5
  286.2.2 HttpUriType Subtype GETBLOB 286-5
  286.2.3 HttpUriType Subtype GETCLOB 286-6
  286.2.4 HttpUriType Subtype GETCONTENTTYPE 286-7
  286.2.5 HttpUriType Subtype GETEXTERNALURL 286-7
  286.2.6 HttpUriType Subtype GETURL 286-7
  286.2.7 HttpUriType Subtype GETXML 286-7
  286.2.8 HttpUriType Subtype HTTPURITYPE 286-8

286.3 Summary of DBURITYPE Subtype Subprograms 286-8
  286.3.1 DBUriType Subtype CREATEURI 286-9
  286.3.2 DBUriType Subtype DBURITYPE 286-9
286.3.3  DBUriType Subtype GETBLOB 286-9
286.3.4  DBUriType Subtype GETCLOB 286-10
286.3.5  DBUriType Subtype GETCONTENTTYPE 286-11
286.3.6  DBUriType Subtype GETEXTERNALURL 286-11
286.3.7  DBUriType Subtype GETURL 286-11
286.3.8  DBUriType Subtype GETXML 286-11
286.4  Summary of XDBURITYPE Subtype Subprograms 286-12
  286.4.1  XDBUriType Subtype CREATEURI 286-12
  286.4.2  XDBUriType Subtype GETBLOB 286-13
  286.4.3  XDBUriType Subtype GETCLOB 286-13
  286.4.4  XDBUriType Subtype GETCONTENTTYPE 286-14
  286.4.5  XDBUriType Subtype GETEXTERNALURL 286-14
  286.4.6  XDBUriType Subtype GETURL 286-14
  286.4.7  XDBUriType Subtype GETXML 286-14
  286.4.8  XDBUriType subtype XDBURITYPE 286-15
286.5  Summary of URIFACTORY Package Subprograms 286-16
  286.5.1  UriFactory Package GETURI 286-16
  286.5.2  UriFactory Package ESCAPEURI 286-16
  286.5.3  UriFactory Package UNESCAPEURI 286-17
  286.5.4  UriFactory Package REGISTERURLHANDLER 286-17
  286.5.5  UriFactory Package UNREGISTERURLHANDLER 286-18

287  JMS Types

287.1  JMS Types Overview 287-1
287.2  JMS Types Security Model 287-2
287.3  Java Versus PL/SQL Datatypes 287-2
287.4  More on Bytes, Stream and Map Messages 287-3
287.5  Upcasting and Downcasting Between General and Specific Messages 287-6
287.6  JMS Types Error Reporting 287-7
287.7  Oracle JMS Type Constants 287-8
287.8  CONVERT_JMS_SELECTOR 287-9
287.9  Summary of JMS Types 287-10
  287.9.1  SYS.AQ$_JMS_MESSAGE Type 287-10
  287.9.2  SYS.AQ$_JMS_TEXT_MESSAGE Type 287-16
  287.9.3  SYS.AQ$_JMS_BYTES_MESSAGE Type 287-20
  287.9.4  SYS.AQ$_JMS_MAP_MESSAGE Type 287-29
  287.9.5  SYS.AQ$_JMS_STREAM_MESSAGE Type 287-38
  287.9.6  SYS.AQ$_JMS_OBJECT_MESSAGE Type 287-48
  287.9.7  SYS.AQ$_JMS_NAMEARRAY Type 287-48
  287.9.8  SYS.AQ$_JMS_VALUE Type 287-48
288 JSON Data Structures

288.1 JSON_ELEMENT_T Object Type
288.2 JSON_OBJECT_T Object Type
288.3 JSON_ARRAY_T Object Type
288.4 JSON_SCALAR_T Object Type
288.5 JSON_KEY_LIST Type

289 Logical Change Record TYPEs

289.1 Logical Change Record TYPEs Overview
289.2 Logical Change Record Types Security Model
289.3 Summary of Logical Change Record Types
289.3.1 LCR$_DDL_RECORD Type
289.3.2 LCR$_ROW_RECORD Type
289.3.3 Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD
289.3.4 LCR$_ROW_LIST Type
289.3.5 LCR$_ROW_UNIT Type

290 Oracle Multimedia ORDAudio TYPE

291 Oracle Multimedia ORDDicom TYPE

292 Oracle Multimedia ORDDoc TYPE

293 Oracle Multimedia ORDImage TYPE

294 Oracle Multimedia SQL/MM Still Image TYPES
Oracle Multimedia ORDVideo TYPE

MGD_ID Package Types

296.1 MGD_ID Package Types Security Model
296.2 Summary of Types
296.2.1 MGD_ID_COMPONENT Object Type
296.2.2 MGD_ID_COMPONENT_VARRAY Object Type
296.2.3 MGD_ID Object Type
296.3 Summary of MGD_ID Subprograms
296.3.1 MGD_ID Constructor Function
296.3.2 FORMAT Function
296.3.3 GET_COMPONENT Function
296.3.4 TO_STRING Function
296.3.5 TRANSLATE Function

Rule TYPEs

297.1 Rule TYPEs Overview
297.2 Rule TYPEs Security Model
297.3 Summary of Rule Types
297.3.1 RE$ATTRIBUTE_VALUE Type
297.3.2 RE$ATTRIBUTE_VALUE_LIST Type
297.3.3 RE$COLUMN_VALUE Type
297.3.4 RE$COLUMN_VALUE_LIST Type
297.3.5 RE$NAME_ARRAY Type
297.3.6 RE$NV_ARRAY Type
297.3.7 RE$NV_LIST Type
297.3.8 RE$NV_NODE Type
297.3.9 RE$RULE_HIT Type
297.3.10 RE$RULE_HIT_LIST Type
297.3.11 RE$RULE_NAME Type
297.3.12 RE$RULE_NAME_LIST Type
297.3.13 RE$TABLE_ALIAS Type
297.3.14 RE$TABLE_ALIAS_LIST Type
297.3.15 RE$TABLE_VALUE Type
297.3.16 RE$TABLE_VALUE_LIST Type
297.3.17 RE$VALUE Type
297.3.18 RE$VALUE_LIST Type
297.3.19 RE$VARIABLE_TYPE Type
## SODA Types

### 298.1 SODA Types Overview

### 298.2 SODA Types Security Model

### 298.3 Summary of SODA Types

#### 298.3.1 SODA_Collection_T Type
- **298.3.1.1** CREATE_INDEX Function
- **298.3.1.2** DROP_INDEX Function
- **298.3.1.3** FIND Function
- **298.3.1.4** FIND_ONE Function
- **298.3.1.5** GET_DATA_GUIDE Function
- **298.3.1.6** GET_METADATA Function
- **298.3.1.7** GET_NAME Function
- **298.3.1.8** INSERT_ONE Function
- **298.3.1.9** INSERT_ONE_AND_GET Function
- **298.3.1.10** REMOVE_ONE Function
- **298.3.1.11** REPLACE_ONE Function
- **298.3.1.12** REPLACE_ONE_AND_GET Function

#### 298.3.2 SODA_Document_T Type
- **298.3.2.1** GET_BLOB Function
- **298.3.2.2** GET_CLOB Function
- **298.3.2.3** GET_CREATED_ON Function
- **298.3.2.4** GET_DATA_TYPE Function
- **298.3.2.5** GET_KEY Function
- **298.3.2.6** GET_LAST_MODIFIED Function
- **298.3.2.7** GET_MEDIA_TYPE Function
- **298.3.2.8** GET_VARCHAR2 Function
- **298.3.2.9** GET_VERSION Function
- **298.3.2.10** SODA_Document_T Function

#### 298.3.3 SODA_Operation_T Type
- **298.3.3.1** COUNT Function
- **298.3.3.2** FILTER Function
- **298.3.3.3** GET_CURSOR Function
- **298.3.3.4** GET_ONE Function
- **298.3.3.5** KEY Function
- **298.3.3.6** KEYS Function
- **298.3.3.7** LIMIT Function
299 UTL Streams Types

299.1 UTL Streams Types Security Model 299-1
299.2 Summary of UTL Binary Streams Types 299-1
   299.2.1 UTL_BINARYINPUTSTREAM Type 299-1
   299.2.2 UTL_BINARYOUTPUTSTREAM Type 299-2
   299.2.3 UTL_CHARACTERINPUTSTREAM Type 299-3
   299.2.4 UTL_CHARACTEROUTPUTSTREAM Type 299-4

300 XMLTYPE

300.1 Summary of XMLType Subprograms 300-1
   300.1.1 CREATENONSCHEMABASEDXML 300-2
   300.1.2 CREATESCHEMABASEDXML 300-2
   300.1.3 CREATEXML 300-3
   300.1.4 EXISTSNODE 300-5
   300.1.5 EXTRACT 300-5
   300.1.6 GETBLOBVAL 300-6
   300.1.7 GETCLOBVAL 300-6
   300.1.8 GETNAMESPACE 300-7
   300.1.9 GETNUMBERVAL 300-7
   300.1.10 GETROOTELEMENT 300-7
   300.1.11 GETSCHEMAMURL 300-7
   300.1.12 GETSTRINGVAL 300-7
   300.1.13 ISFRAGMENT 300-8
   300.1.14 ISSCHEMABASED 300-8
   300.1.15 ISSCHEMAVALID 300-8
   300.1.16 ISSCHEMAVALIDATED 300-9
   300.1.17 SCHEMAVALIDATE 300-9
   300.1.18 SETSCHEMAVALIDATED 300-9
   300.1.19 TOOBJECT 300-10
Preface

This Preface contains these topics:

• Audience
• Related Documents
• Conventions

Audience

Oracle Database PL/SQL Packages and Types Reference is intended for programmers, systems analysts, project managers, and others interested in developing database applications. This manual assumes a working knowledge of application programming and familiarity with SQL to access information in relational database systems. Some sections also assume a knowledge of basic object-oriented programming.

Documentation Accessibility

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents:

• Oracle Database Development Guide
• Oracle Database PL/SQL Language Reference

Many of the examples in this book use the sample schemas, which are installed by default when you select the Basic Installation option with an Oracle Database installation. Refer to Oracle Database Sample Schemas for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at http://shop.oracle.com/

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN).
## Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

### Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.</td>
<td>When you specify this clause, you create an index-organized table.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Italic typeface indicates book titles or emphasis.</td>
<td>Oracle Database Concepts</td>
</tr>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, column names, packages and classes, usernames, and roles.</td>
<td>You can specify this clause only for a NUMBER column. You can back up the database by using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Use the DBMS_STATS.GENERATE_STATS procedure.</td>
</tr>
<tr>
<td><strong>lowercase</strong></td>
<td>Lowercase monospace typeface indicates executable programs, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names and connect identifiers, user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.</td>
<td>Enter sqlplus to start SQL*Plus. The password is specified in the orapwd file. Back up the datafiles and control files in the /disk1/oracle/dbs directory. The department_id, department_name, and location_id columns are in the hr.departments table. Set the QUERY_REWRITE_ENABLED initialization parameter to true. The JRepUtil class implements these methods.</td>
</tr>
<tr>
<td><strong>lowercase italic</strong></td>
<td>Lowercase italic monospace font represents placeholders or variables.</td>
<td>You can specify the parallel_clause. Run old_release.SQL where old_release refers to the release you installed prior to upgrading.</td>
</tr>
</tbody>
</table>
Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Anything enclosed in brackets is optional.</td>
<td>DECIMAL (digits [ , precision ])</td>
</tr>
<tr>
<td>{ }</td>
<td>Braces are used for grouping items.</td>
<td>{ENABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A vertical bar represents a choice of two options.</td>
</tr>
<tr>
<td>...</td>
<td>Ellipsis points mean repetition in syntax descriptions. In addition, ellipsis points can mean an omission in code examples or text.</td>
<td>CREATE TABLE ... AS subquery; SELECT col1, col2, ..., coln FROM employees;</td>
</tr>
<tr>
<td>Other symbols</td>
<td>You must use symbols other than brackets ([ ]), braces ({ }), vertical bars (</td>
<td>), and ellipsis points (...) exactly as shown.</td>
</tr>
<tr>
<td>Italicics</td>
<td>Italized text indicates placeholders or variables for which you must supply particular values.</td>
<td>enter password DB_NAME = database_name</td>
</tr>
<tr>
<td>UPPERCASE</td>
<td>Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. Because these terms are not case sensitive, you can use them in either UPPERCASE or lowercase.</td>
<td>SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;</td>
</tr>
<tr>
<td>lowercase</td>
<td>Lowercase typeface indicates user-defined programmatic elements, such as names of tables, columns, or files. <strong>Note:</strong> Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.</td>
<td>SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;</td>
</tr>
</tbody>
</table>
Changes in This Release for Oracle Database PL/SQL Packages and Types Reference

This preface contains:

• Changes in Oracle Database Release 18c, Version 18.1

Changes in Oracle Database Release 18c, Version 18.1

The following are changes in Oracle Database PL/SQL Packages and Types Reference for Oracle Database release 18c, version 18.1.

• New Features
• Deprecated Features
• Desupported Features

See Oracle Database New Features Guide for a complete list of new features in this release.

New Features

This section lists the major new features in Oracle Database release 18c version 18.1 PL/SQL Packages and Types Reference.

The following features are new in this release:

• DBMS_DATA_MINING

New Mining function TIME_SERIES is added. See DBMS_DATA_MINING — Mining Functions.

The following model settings are new:

– DBMS_DATA_MINING — Algorithm Settings: CUR Matrix Decomposition
– DBMS_DATA_MINING — Algorithm Settings: Exponential Smoothing Models
– DBMS_DATA_MINING — Algorithm Settings: Neural Network
– DBMS_DATA_MINING — Algorithm Settings: Random Forest

New solver settings ADMM and LBFGS are added. See DBMS_DATA_MINING — Solver Settings.

The following procedures are new:

– Create Model Using Registration Information
– DROP_ALGORITHM Procedure
– DROP_PARTITION Procedure
– EXPORT_SERMODEL Procedure
– FETCH_JSON_SCHEMA Procedure
– IMPORT_SERMODEL Procedure
– JSON Schema for R Extensible Algorithm
– REGISTER_ALGORITHM Procedure

• DBMS_DATAPUMP

The Export operation under DATA_OPTIONS parameter of the SET_PARAMETER Procedure supports a new value KU$_DATAOPT_XMLTYPE_CLOB.

• See SET_PARAMETER Procedures

• DBMS_HPROF

The new CREATE_TABLES procedure allows you to create hierarchical profiler database tables and data structures in the user's session.

CREATE_TABLES Procedure

• DBMS_INMEMORY_ADMIN

Two new procedures added to the DBMS_INMEMORY_ADMIN package.

See
– AIM_GET_PARAMETER Procedure
– AIM_SET_PARAMETER Procedure

• DBMS_MEMOPTIMIZE

The new package DBMS_MEMOPTIMIZE provides an interface for managing data in the memoptimize pool.

See “DBMS_MEMOPTIMIZE”

• DBMS_PDB

New procedures added to the DBMS_PDB package.

See EXPORTRMANBACKUP Procedure

• DBMS_WORKLOAD_REPLAY

The LOAD_LONG_SQLTEXT procedure of the package DBMS_WORKLOAD_REPLAY loads the captured SQL statements that are longer than 1000 characters to the DBA_WORKLOAD_LONG_SQLTEXT view.

• UTL_I18N

The new overloaded functions VALIDATE_CHARACTER_ENCODING of the package UTL_I18N validate the character encoding of VARCHAR2, NVARCHAR2, CLOB, and NCLOB data.

See "VALIDATE_CHARACTER_ENCODING Functions"

• DBMS_SODA

The new package DBMS_SODA is a PL/SQL package implementing Simple Oracle Document Access (SODA) which allows you to use the Oracle Database as a NoSQL document store.

See "DBMS_SODA"
• **DBMS_TF**
  The new DBMS_TF package contains utilities for Polymorphic Table Functions (PTF) implementation.
  See "DBMS_TF"

• **SODA Types**
  "SODA Types"

### Deprecated Features

List of deprecated features in Oracle Database release 18c version 18.1 PL/SQL Packages and Types Reference.

The following features are deprecated in this release, and may be desupported in a future release:

• **Extended Datatype Support (EDS)** is deprecated from the DBMS_LOGSTDBY package. All EDS-supported Oracle data types are now supported natively by logical standbys or Oracle GoldenGate.

• **Deprecation of the DBMS_LOCK.SLEEP Procedure.**
  Use DBMS_SESSION.SLEEP Procedure instead.

• **GET_MODEL_DETAILS** are deprecated and are replaced with Model Detail Views. See Summary of DBMS_DATA_MINING Subprograms for a list of deprecated functions and Deprecated Types for deprecated types. See Oracle Data Mining User’s Guide for Model Detail Views and Oracle Database Reference for catalog views.

• **Deprecation of the DBMS_XMLQUERY package**
  Use DBMS_XMLGEN instead.

• **Deprecation of the DBMS_XMLSAVE package**
  Use DBMS_XMLSTORE instead.

• **Deprecation of these Oracle Multimedia packages and types:**
  - ORD_AUDIO
  - ORD_DOC
  - ORD_IMAGE
  - ORD_VIDEO
  - Oracle Multimedia ORDAudio
  - Oracle Multimedia ORDDoc
  - Oracle Multimedia ORDImage
  - Oracle Multimedia ORDVideo

  Use Oracle SecureFiles with third-party products for metadata extraction and for image processing and conversion instead of Oracle Multimedia.
Desupported Features

This section lists the desupported features in Oracle Database release 18c version 18.1 PL/SQL Packages and Types Reference.

The following features are desupported in this release:

- The following subprograms are desupported from the **DBMS_XDB** package:
  - ADDHTTPEXPIREMAPPING
  - ADDMIMEMAPPING
  - ADDSCHEMALOCMAPPING
  - ADDSERVLETMAPPING
  - ADDSERVLETSECROLE
  - ADDXMLEXTENSION
  - CFG_GET
  - CFG_REFRESH
  - CFG_UPDATE
  - DELETEHTTPEXPIREMAPPING
  - DELETERMIMEMAPPING
  - DELETESCHEMALOCMAPPING
  - DELETESERVLET
  - DELETESERVLETMAPPING
  - DELETESERVLETSECROLE
  - DELETXMLEXTENSION
  - GETFTPPORT
  - GETHTTPPORT
  - GETLISTENERENDPOINT
  - SETFTPPORT
  - SETHTTPPORT
  - SETLISTENERENDPOINT
  - SETLISTENERLOCALACCESS

  Use the subprograms available in package **DBMS_XDB_CONFIG** instead.

- The following initialization parameter is desupported from the **DBMS_LOGMNR_D** package:
  - UTL_FILE_DIR

  Use a directory object instead.

- The following constants are desupported from the **DBMS_XDB** package:
  - XDB_ENDPOINT_HTTP
Use the constants available in package `DBMS_XDB_CONFIG` instead.

- The following subprograms are desupported from the `DBMS_XMLSCHEMA` package:
  - GENERATESCHEMA
  - GENERATESCHEMAS

There are no replacements for these subprograms and there is no workaround for this change.

- The `DBMS_XMLTRANSLATIONS` package is desupported. There are no replacements for these subprograms and there is no workaround for this change.
1

Introduction to Oracle Supplied PL/SQL Packages & Types

Oracle supplies many PL/SQL packages with the Oracle server to extend database functionality and provide PL/SQL access to SQL features. You can use the supplied packages when creating your applications or for ideas in creating your own stored procedures.

This manual covers the packages provided with the Oracle database server. Packages supplied with other products, such as Oracle Developer or the Oracle Application Server, are not covered.

Note that not every package or type described in this manual or elsewhere in the Oracle Database Documentation Library is installed by default. In such cases, the documentation states this and explains how to install the object. Run this query as a suitably privileged user:

```
SELECT DISTINCT Owner, Object_Type, Object_Name FROM DBA_Objects_AE
WHERE Owner IN ('SYS', 'OUTLN', 'SYSTEM', 'CTXSYS', 'DBSNMP', 'LOGSTDBY_ADMINISTRATOR', 'ORDSYS', 'ORDPLUGINS', 'OEM_MONITOR', 'MDSYS', 'LBACSYS', 'DMSYS', 'VMSYS', 'OLAPDBA', 'OLAPSVR', 'OLAP_USER', 'OLAPSYS', 'EXFSYS', 'SYMAN', 'MDDATA', 'SI_INFORMTN_SCHEMA', 'XDB', 'ODM')
AND Object_Type IN ('PACKAGE', 'TYPE')
ORDER BY Owner, Object_Type, Object_Name
```

This lists every Oracle-supplied package and type that is currently installed in the database. Note that it lists a number of objects not mentioned in the Oracle Database Documentation Library. This is deliberate. Some of the Oracle-supplied packages and types are intended to be used only by other Oracle-supplied components. Any package or type that is not described in the Oracle Database Documentation Library is not supported for direct customer use.

This chapter contains the following topics:

- Package Overview
- Summary of Oracle Supplied PL/SQL Packages and Types

See Also:

Oracle Database Development Guide for information on how to create your own packages
1.1 Package Overview

A package is an encapsulated collection of related program objects stored together in the database. Program objects are procedures, functions, variables, constants, cursors, and exceptions.

1.1.1 Package Components

PL/SQL packages have two parts: the specification and the body, although sometimes the body is unnecessary. The specification is the interface to your application; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the specification.

Unlike subprograms, packages cannot be called, parameterized, or nested. However, the formats of a package and a subprogram are similar:

```
CREATE PACKAGE name AS  -- specification (visible part)
    -- public type and item declarations
    -- subprogram specifications
END [name];

CREATE PACKAGE BODY name AS  -- body (hidden part)
    -- private type and item declarations
    -- subprogram bodies
BEGIN
    -- initialization statements
END [name];
```

The specification holds public declarations that are visible to your application. The body holds implementation details and private declarations that are hidden from your application. You can debug, enhance, or replace a package body without changing the specification. You can change a package body without recompiling calling programs because the implementation details in the body are hidden from your application.

1.1.2 Using Oracle Supplied Packages

Most Oracle supplied packages are automatically installed when the database is created. Certain packages are not installed automatically. Special installation instructions for these packages are documented in the individual chapters.

To call a PL/SQL function from SQL, you must either own the function or have EXECUTE privileges on the function. To select from a view defined with a PL/SQL function, you must have SELECT privileges on the view. No separate EXECUTE privileges are needed to select from the view. Instructions on special requirements for packages are documented in the individual chapters.

1.1.3 Creating New Packages

To create packages and store them permanently in an Oracle database, use the CREATE PACKAGE and CREATE PACKAGE BODY statements. You can execute these statements interactively from SQL*Plus or Enterprise Manager.

To create a new package, do the following:
1. Create the package specification with the `CREATE PACKAGE` statement.
   You can declare program objects in the package specification. Such objects are called `public` objects. Public objects can be referenced outside the package, as well as by other objects in the package.

   **Note:**
   It is often more convenient to add the `OR REPLACE` clause in the `CREATE PACKAGE` statement. But note that `CREATE PACKAGE` warns you if you are about to overwrite an existing package with the same name while `CREATE OR REPLACE` just overwrites it with no warning.

2. Create the package body with the `CREATE PACKAGE BODY` statement.
   You can declare and define program objects in the package body.
   - You must define public objects declared in the package specification.
   - You can declare and define additional package objects, called `private` objects. Private objects are declared in the package body rather than in the package specification, so they can be referenced only by other objects in the package. They cannot be referenced outside the package.

   **See Also:**
   - `Oracle Database PL/SQL Language Reference`
   - `Oracle Database Development Guide` for more information on creating new packages
   - `Oracle Database Concepts` for more information on storing and executing packages

### 1.1.3.1 Separating the Specification and Body

The specification of a package declares the public types, variables, constants, and subprograms that are visible outside the immediate scope of the package. The body of a package defines the objects declared in the specification, as well as private objects that are not visible to applications outside the package.

Oracle stores the specification and body of a package separately in the database. Other schema objects that call or reference public program objects depend only on the package specification, not on the package body. Using this distinction, you can change the definition of a program object in the package body without causing Oracle to invalidate other schema objects that call or reference the program object. Oracle invalidates dependent schema objects only if you change the declaration of the program object in the package specification.
1.1.3.1.1 Creating a New Package: Example

The following example shows a package specification for a package named EMPLOYEE_MANAGEMENT. The package contains one stored function and two stored procedures.

CREATE PACKAGE employee_management AS
    FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
        mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
        deptno NUMBER) RETURN NUMBER;
    PROCEDURE fire_emp (emp_id NUMBER);
    PROCEDURE sal_raise (emp_id NUMBER, sal_incr NUMBER);
END employee_management;

The body for this package defines the function and the procedures:

CREATE PACKAGE BODY employee_management AS
    FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
        mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
        deptno NUMBER) RETURN NUMBER IS

    new_empno    NUMBER(10);
    BEGIN
        SELECT emp_sequence.NEXTVAL INTO new_empno FROM dual;
        INSERT INTO emp VALUES (new_empno, name, job, mgr,
            hiredate, sal, comm, deptno);
        RETURN (new_empno);
    END hire_emp;

    PROCEDURE fire_emp(emp_id IN NUMBER) AS
    BEGIN
        DELETE FROM emp WHERE empno = emp_id;
        IF SQL\NOTFOUND THEN
            raise_application_error(-20011, 'Invalid Employee Number: ' || TO_CHAR(emp_id));
        END IF;
    END fire_emp;

    PROCEDURE sal_raise (emp_id IN NUMBER, sal_incr IN NUMBER) AS
    BEGIN
        -- If employee exists, then update salary with increase.
        UPDATE emp
        WHERE empno = emp_id;
        IF SQL\NOTFOUND THEN
        END IF;
    END sal_raise;
END employee_management;

The function accepts all arguments for the fields in the employee table except for the employee number. A value for this field is supplied by a sequence. The function returns the sequence number generated by the call to this function.

The procedure deletes the employee with an employee number that corresponds to the argument emp_id. If no employee is found, then an exception is raised.

The procedure accepts two arguments. Emp_id is a number that corresponds to an employee number. Sal_incr is the amount by which to increase the employee’s salary.
raise_application_error(-20011, 'Invalid Employee Number: ' || TO_CHAR(emp_id));
END IF;
END sal_raise;
END employee_management;

**Note:**
If you want to try this example, then first create the sequence number `emp_sequence`. You can do this using the following SQL*Plus statement:

```
SQL> CREATE SEQUENCE emp_sequence
>   START WITH 8000 INCREMENT BY 10;
```

1.4 Referencing Package Contents

When you want to reference the types, items, and subprograms declared in a package specification, use the dot notation.

For example:

```
package_name.type_name
package_name.item_name
package_name.subprogram_name
```

1.2 Summary of Oracle Supplied PL/SQL Packages and Types

PL/SQL packages are supplied with the Oracle server that extend database functionality and provide PL/SQL access to SQL features.

These packages, listed in the following table, run as the invoking user, rather than the package owner. Unless otherwise noted, the packages are callable through public synonyms of the same name.

**Note:**
- The procedures and functions provided in these packages and their external interfaces are reserved by Oracle and are subject to change.
- Modifying Oracle supplied packages can cause internal errors and database security violations. Do not modify supplied packages.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APEX_APPLICATION</td>
<td>For a complete description of all the packages that ship with Oracle Application Express, see the Oracle Application Express API Reference</td>
</tr>
<tr>
<td>APEX_ZIP</td>
<td></td>
</tr>
<tr>
<td>CTX_ADM</td>
<td>Lets you administer servers and the data dictionary</td>
</tr>
</tbody>
</table>
## Table 1-1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTX_ANL</td>
<td>The CTX_ANL package is used with AUTO_LEXER and provides procedures for adding and dropping a custom dictionary from the lexer.</td>
</tr>
<tr>
<td>CTX_CLS</td>
<td>Lets you generate CTXRULE rules for a set of documents</td>
</tr>
<tr>
<td>CTX_DDL</td>
<td>Lets you create and manage the preferences, section lists and stopgroups required for Text indexes</td>
</tr>
<tr>
<td>CTX_DOC</td>
<td>Lets you request document services</td>
</tr>
<tr>
<td>CTX_ENTITY</td>
<td>The CTX_ENTITY package enables you to search for terms that are unknown to you without specifying a particular search text.</td>
</tr>
<tr>
<td>CTX_OUTPUT</td>
<td>Lets you manage the index log</td>
</tr>
<tr>
<td>CTX_QUERY</td>
<td>Lets you generate query feedback, count hits, and create stored query expressions</td>
</tr>
<tr>
<td>CTX_REPORT</td>
<td>Lets you create various index reports</td>
</tr>
<tr>
<td>CTX_THESES</td>
<td>Lets you to manage and browse thesauri</td>
</tr>
<tr>
<td>CTX_ULEXER</td>
<td>For use with the user-lexer</td>
</tr>
<tr>
<td>DBMS_ADDM</td>
<td>Facilitates the use of Advisor functionality regarding the Automatic Database Diagnostic Monitor</td>
</tr>
<tr>
<td>DBMS_ADVANCED_REWRITE</td>
<td>Contains interfaces for advanced query rewrite users to create, drop, and maintain functional equivalence declarations for query rewrite</td>
</tr>
<tr>
<td>DBMS_ADVISOR</td>
<td>Part of the SQLAccess Advisor, an expert system that identifies and helps resolve performance problems relating to the execution of SQL statements</td>
</tr>
<tr>
<td>DBMS_ALERT</td>
<td>Provides support for the asynchronous notification of database events</td>
</tr>
<tr>
<td>DBMS_APP_CONT</td>
<td>The DBMS_APP_CONT package provides an interface to determine if the in-flight transaction on a now unavailable session committed or not, and if the last call on that session completed or not.</td>
</tr>
<tr>
<td>DBMS_APPLICATION_INFO</td>
<td>Lets you register an application name with the database for auditing or performance tracking purposes</td>
</tr>
<tr>
<td>DBMS_APPLY_ADM</td>
<td>Provides administrative procedures to start, stop, and configure an apply process</td>
</tr>
<tr>
<td>DBMS_AQ</td>
<td>Lets you add a message (of a predefined object type) onto a queue or to dequeue a message</td>
</tr>
<tr>
<td>DBMS_AQADM</td>
<td>Lets you perform administrative functions on a queue or queue table for messages of a predefined object type</td>
</tr>
<tr>
<td>DBMS_AQELM</td>
<td>Provides procedures to manage the configuration of Advanced Queuing asynchronous notification by e-mail and HTTP</td>
</tr>
<tr>
<td>DBMS_AQIN</td>
<td>Plays a part in providing secure access to the Oracle JMS interfaces</td>
</tr>
<tr>
<td>DBMS_ASSERT</td>
<td>Provides an interface to validate properties of the input value</td>
</tr>
</tbody>
</table>
Table 1-1  (Cont.) Summary of Oracle Supplied PL/SQL Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_AUDIT_UTIL</td>
<td>The DBMS_AUDIT_UTIL package provides functions that enable you to format the output of queries to the DBA_FGA_AUDIT_TRAIL, DBA_AUDIT_TRAIL, UNIFIED_AUDIT_TRAIL, and V$XML_AUDIT_TRAIL views.</td>
</tr>
<tr>
<td>DBMS_AUDIT_MGMT</td>
<td>The DBMS_AUDIT_MGMT package provides subprograms to manage audit trail records. These subprograms enable audit administrators to manage the audit trail.</td>
</tr>
<tr>
<td>DBMS_AUTO_REPORT</td>
<td>The DBMS_AUTO_REPORT package provides an interface to view SQL Monitoring and Real-time Automatic Database Diagnostic Monitor (ADDM) data that has been captured into Automatic Workload Repository (AWR). It also provides subprograms to control the behavior of how these data are captured to AWR.</td>
</tr>
<tr>
<td>DBMS_AUTO_SQLTUNE</td>
<td>The DBMS_AUTO_SQLTUNE package is the interface for managing the Automatic SQL Tuning task.</td>
</tr>
<tr>
<td>DBMS_AUTO_TASK_ADMIN</td>
<td>Used by the DBA as well as Enterprise Manager to access the AUTOTASK controls</td>
</tr>
<tr>
<td>DBMS_CREDENTIAL</td>
<td>The DBMS_CREDENTIAL package provides an interface for authenticating and impersonating EXTPROC callout functions, as well as external jobs, remote jobs and file watchers from the SCHEDULER.</td>
</tr>
<tr>
<td>DBMS_COMPRESSION</td>
<td>Provides an interface to facilitate choosing the correct compression level for an application</td>
</tr>
<tr>
<td>DBMS_CSX_ADMIN</td>
<td>Provides an interface to manage the Database Resident Connection Pool</td>
</tr>
<tr>
<td>DBMS_CUBE</td>
<td>Contains subprograms that create OLAP cubes and dimensions, and that load and process the data for querying</td>
</tr>
<tr>
<td>DBMS_CUBE_ADVISE</td>
<td>Contains subprograms for evaluating cube materialized views to support log-based fast refresh and query rewrite</td>
</tr>
<tr>
<td>DBMS_CUBE_LOG</td>
<td>DBMS_CUBE_LOG contains subprograms for creating and managing logs for cubes and cube dimensions.</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DBMS_DATA_MINING</td>
<td>Implements the Oracle Data Mining interface for creating, evaluating, and managing mining models</td>
</tr>
<tr>
<td>DBMS_DATA_MINING_TRANSFORM</td>
<td>Provides subroutines that can be used to prepare data for Oracle Data Mining</td>
</tr>
<tr>
<td>DBMS_DATAPUMP</td>
<td>Lets you move all, or part of, a database between databases, including both data and metadata</td>
</tr>
<tr>
<td>DBMS_DB_VERSION</td>
<td>The DBMS_DB_VERSION package specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.</td>
</tr>
<tr>
<td>DBMS_DBCOMP</td>
<td>The DBMS_DBCOMP package performs block comparison to detect lost writes or database inconsistencies between a primary database and one or more physical standby databases.</td>
</tr>
<tr>
<td>DBMS_DBFS_CONTENT</td>
<td>Provides an interface comprising a file system-like abstraction backed by one or more Store Providers</td>
</tr>
<tr>
<td>DBMS_DBFS_CONTENT_SPI</td>
<td>Provides the Application Programming Interface (API) specification for DBMS_DBFS_CONTENT service providers</td>
</tr>
<tr>
<td>DBMS_DBFS_HS</td>
<td>Provides users the ability to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management for their database tables</td>
</tr>
<tr>
<td>DBMS_DBFS_SFS</td>
<td>Provides an interface to operate a SecureFile-based store (SFS) for the content interface described in the DBMS_DBFS_CONTENT package</td>
</tr>
<tr>
<td>DBMS_DDL</td>
<td>Provides access to some SQL DDL statements from stored procedures, and provides special administration operations not available as DDLs</td>
</tr>
<tr>
<td>DBMS_DEBUG</td>
<td>Implements server-side debuggers and provides a way to debug server-side PL/SQL program units</td>
</tr>
<tr>
<td>DBMS_DEBUG_JDWP</td>
<td>The DBMS_DEBUG_JDWP provides the interface to initiate and control the debugging of PL/SQL stored procedures and Java stored procedures over Java Debug Wire Protocol (JDWP).</td>
</tr>
<tr>
<td>DBMS_DEBUG_JDWP_CUSTOM</td>
<td>The DBMS_DEBUG_JDWP_CUSTOM package is a PL/SQL package that provides database users a means to perform custom handling of a debug connection request of a database session to a debugger using the Java Debug Wire Protocol (JDWP).</td>
</tr>
<tr>
<td>DBMS_DESCRIBE</td>
<td>Describes the arguments of a stored procedure with full name translation and security checking</td>
</tr>
<tr>
<td>DBMS_DG</td>
<td>Allows applications to notify the primary database in an Oracle Data Guard broker environment to initiate a fast-start failover when the application encounters a condition that warrants a failover</td>
</tr>
<tr>
<td>DBMS_DIMENSION</td>
<td>Enables you to verify dimension relationships and provides an alternative to the Enterprise Manager Dimension Wizard for displaying a dimension definition</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DBMS_DISTRIBUTED_TRUST_ADMIN</td>
<td>Maintains the Trusted Database List, which is used to determine if a privileged database link from a particular server can be accepted</td>
</tr>
<tr>
<td>DBMS_DNFS</td>
<td>The DBMS_DNFS package provides an interface to assist in creating a database using files in the backup set.</td>
</tr>
<tr>
<td>DBMS_DST</td>
<td>The DBMS_DST package provides an interface to apply the Daylight Saving Time (DST) patch to the Timestamp with Time Zone datatype.</td>
</tr>
<tr>
<td>DBMS_EDITIONS_UTILITIES</td>
<td>The DBMS_EDITIONS_UTILITIES package provides helper functions for edition-related operations.</td>
</tr>
<tr>
<td>DBMS_EPG</td>
<td>Implements the embedded PL/SQL gateway that enables a Web browser to invoke a PL/SQL stored procedure through an HTTP listener.</td>
</tr>
<tr>
<td>DBMS_ERRLOG</td>
<td>Provides a procedure that enables you to create an error logging table so that DML operations can continue after encountering errors rather than abort and roll back.</td>
</tr>
<tr>
<td>DBMS_FGA</td>
<td>Provides fine-grained security functions.</td>
</tr>
<tr>
<td>DBMS_FILE_GROUP</td>
<td>One of a set of Streams packages, provides administrative interfaces for managing file groups, file group versions, files, and file group repositories.</td>
</tr>
<tr>
<td>DBMS_FILE_TRANSFER</td>
<td>Lets you copy a binary file within a database or to transfer a binary file between databases.</td>
</tr>
<tr>
<td>DBMS_FLASHBACK</td>
<td>Lets you flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN).</td>
</tr>
<tr>
<td>DBMS_FLASHBACK_ARCHIVE</td>
<td>Contains procedures for disassociation and re-association of a Flashback Data Archive (FDA) enabled table from/with its underlying FDA, respectively.</td>
</tr>
<tr>
<td>DBMS_FREQUENT_ITEMSET</td>
<td>Enables frequent itemset counting.</td>
</tr>
<tr>
<td>DBMS_FS</td>
<td>The DBMS_FS package for performing operations on an Oracle file system (make, mount, unmount, and destroy operations) in an Oracle database.</td>
</tr>
<tr>
<td>DBMS_GOLDENGATE_AUTH</td>
<td>The DBMS_GOLDENGATE_AUTH package provides subprograms for granting privileges to and revoking privileges from GoldenGate administrators.</td>
</tr>
<tr>
<td>DBMS_HADOOP</td>
<td>The DBMS_HADOOP package provides a PL/SQL procedure called CREATE_EXTDDL_FOR_HIVE(), that creates an Oracle external table for a given Hive table.</td>
</tr>
<tr>
<td>DBMS_HANG_MANAGER</td>
<td>The DBMS_HANG_MANAGER package provides a method of changing some Hang Manager configuration parameters.</td>
</tr>
<tr>
<td>DBMS_HEAT_MAP</td>
<td>The DBMS_HEAT_MAP package provides an interface to externalize heatmaps at various levels of storage including block, extent, segment, object, and tablespace.</td>
</tr>
<tr>
<td>DBMS_HIERARCHY</td>
<td>Contains functions and procedures for validating the data in tables used by analytic views and hierarchies and for creating a log table.</td>
</tr>
</tbody>
</table>
Table 1-1 (Cont.) Summary of Oracle Supplied PL/SQL Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_HM</td>
<td>Contains constants and procedure declarations for health check management</td>
</tr>
<tr>
<td>DBMS_HPROF</td>
<td>Provides an interface for profiling the execution of PL/SQL applications</td>
</tr>
<tr>
<td>DBMS_HS_PARALLEL</td>
<td>Enables parallel processing for heterogeneous targets access</td>
</tr>
<tr>
<td>DBMS_HS_PASSTHROUGH</td>
<td>Lets you use Heterogeneous Services to send pass-through SQL statements to non-Oracle systems</td>
</tr>
<tr>
<td>DBMS_ILM</td>
<td>The DBMS_ILM package provides an interface for implementing Information Lifecycle Management (ILM) strategies using Automatic Data Optimization (ADO) policies.</td>
</tr>
<tr>
<td>DBMS_ILM_ADMIN</td>
<td>The DBMS_ILM_ADMIN package provides an interface to customize Automatic Data Optimization (ADO) policy execution.</td>
</tr>
<tr>
<td>DBMS_INMEMORY</td>
<td>The DBMS_INMEMORY package provides an interface for In-Memory Column Store (IM column store) functionality.</td>
</tr>
<tr>
<td>DBMS_INMEMORY_ADMIN</td>
<td>The DBMS_INMEMORY_ADMIN package provides interfaces for managing an In-Memory FastStart (IM FastStart) area and In-Memory Expressions (IM expressions).</td>
</tr>
<tr>
<td>DBMS_IOT</td>
<td>Creates a table into which references to the chained rows for an Index Organized Table can be placed using the ANALYZE command</td>
</tr>
<tr>
<td>DBMS_JAVA</td>
<td>Provides a PL/SQL interface for accessing database functionality from Java</td>
</tr>
<tr>
<td>DBMS_JOB</td>
<td>Schedules and manages jobs in the job queue (Deprecated)</td>
</tr>
<tr>
<td>DBMS_JSON</td>
<td>Provides an interface for data-guide operations</td>
</tr>
<tr>
<td>DBMS_LDAP</td>
<td>Provides functions and procedures to access data from LDAP servers</td>
</tr>
<tr>
<td>DBMS_LDAP_U TL</td>
<td>Provides the Oracle Extension utility functions for LDAP</td>
</tr>
<tr>
<td>DBMS_LIBCACHE</td>
<td>Prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution</td>
</tr>
<tr>
<td>DBMS_LOB</td>
<td>Provides general purpose routines for operations on Oracle Large Object (LOBs) datatypes - BLOB, CLOB (read/write), and BFILES (read-only)</td>
</tr>
<tr>
<td>DBMS_LOCK</td>
<td>Lets you request, convert and release locks through Oracle Lock Management services</td>
</tr>
<tr>
<td>DBMS_LOGMNR</td>
<td>Provides functions to initialize and run the log reader</td>
</tr>
<tr>
<td>DBMS_LOGMNR_D</td>
<td>Queries the dictionary tables of the current database, and creates a text based file containing their contents</td>
</tr>
<tr>
<td>DBMS_LOGSTDBY</td>
<td>Describes procedures for configuring and managing the logical standby database environment</td>
</tr>
<tr>
<td>DBMS_LOGSTDBY_CONTEXT</td>
<td>You can use the procedures provided in the DBMS_LOGSTDBY_CONTEXT package to set and retrieve various parameters associated with LSBY_APPLY_CONTEXT.</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DBMS_METADATA</td>
<td>Lets callers easily retrieve complete database object definitions (metadata) from the dictionary</td>
</tr>
<tr>
<td>DBMS_METADATA_DIFF</td>
<td>Contains the interfaces for comparing two metadata documents in SXML format. The result of the comparison is an SXML difference document. This document can be converted to other formats using the DBMS_METADATA submit interface and the CONVERT API.</td>
</tr>
<tr>
<td>DBMS_MGD_ID_UTL</td>
<td>Provides a set of utility subprograms</td>
</tr>
<tr>
<td>DBMS_MGWADM</td>
<td>Describes the Messaging Gateway administrative interface; used in Advanced Queuing</td>
</tr>
<tr>
<td>DBMS_MGWMSG</td>
<td>Describes object types (used by the canonical message types to convert message bodies) and helper methods, constants, and subprograms for working with the Messaging Gateway message types; used in Advanced Queuing.</td>
</tr>
<tr>
<td>DBMS_MONITOR</td>
<td>Let you use PL/SQL for controlling additional tracing and statistics gathering</td>
</tr>
<tr>
<td>DBMS_MVIEW</td>
<td>Lets you refresh snapshots that are not part of the same refresh group and purge logs. DBMS_SNAPSHOT is a synonym.</td>
</tr>
<tr>
<td>DBMS_MVIEW_STATS</td>
<td>The DBMS_MVIEW_STATS package provides an interface to manage the collection and retention of statistics for materialized view refresh operations.</td>
</tr>
<tr>
<td>DBMS_NETWORK_ACL_ADMIN</td>
<td>Provides the interface to administer the network Access Control List (ACL)</td>
</tr>
<tr>
<td>DBMS_NETWORK_ACLUTILITY</td>
<td>The DBMS_NETWORK_ACLUTILITY package provides the utility functions to facilitate the evaluation of access control list (ACL) assignments governing TCP connections to network hosts.</td>
</tr>
<tr>
<td>DBMS_ODCI</td>
<td>Returns the CPU cost of a user function based on the elapsed time of the function</td>
</tr>
<tr>
<td>DBMS_OUTLN</td>
<td>Provides the interface for procedures and functions associated with management of stored outlines Synonymous with OUTLN_PKG</td>
</tr>
<tr>
<td>DBMS_OUTPUT</td>
<td>Accumulates information in a buffer so that it can be retrieved later</td>
</tr>
<tr>
<td>DBMS_PARALLEL_EXECUTE</td>
<td>Enables the user to incrementally update table data in parallel</td>
</tr>
<tr>
<td>DBMS_PART</td>
<td>The DBMS_PART package provides an interface for maintenance and management operations on partitioned objects.</td>
</tr>
<tr>
<td>DBMS_PCLXUTIL</td>
<td>Provides intra-partition parallelism for creating partition-wise local indexes</td>
</tr>
<tr>
<td>DBMS_PDB</td>
<td>The DBMS_PDB package provides an interface to examine and manipulate data about pluggable databases (PDBs) in a multitenant container database (CDB). It also contains an interface specify which database objects are application common objects.</td>
</tr>
</tbody>
</table>
### Table 1-1  (Cont.) Summary of Oracle Supplied PL/SQL Packages

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_PDB_ALTER_SHARING</td>
<td>In an application container with a pre-installed application, the DBMS_PDB_ALTER_SHARING package provides an interface to set database objects as application common objects or to specify that a database object is not an application common object.</td>
</tr>
<tr>
<td>DBMS_PERF</td>
<td>The DBMS_PERF package provides an interface to generate active reports for monitoring database performance.</td>
</tr>
<tr>
<td>DBMSPIPE</td>
<td>Provides a DBMS pipe service which enables messages to be sent between sessions</td>
</tr>
<tr>
<td>DBMS_PLSQL_CODE_COVERAGE</td>
<td>The DBMS_PLSQL_CODE_COVERAGE package provides an interface for the collection of code coverage data of PL/SQL applications at the basic block level.</td>
</tr>
<tr>
<td>DBMS_PREDICTIVE_ANALYTICS</td>
<td>Provides subroutines that implement automatic data mining operations for predict, explain, and profile.</td>
</tr>
<tr>
<td>DBMS_PREPROCESSOR</td>
<td>Provides an interface to print or retrieve the source text of a PL/SQL unit in its post-processed form</td>
</tr>
<tr>
<td>DBMS_PRIVILEGE_CAPTURE</td>
<td>The DBMS_PRIVILEGE_CAPTURE package provides an interface to database privilege analysis.</td>
</tr>
<tr>
<td>DBMS_PROCESS</td>
<td>The DBMS_PROCESS package provides an interface to manage the prespawned servers.</td>
</tr>
<tr>
<td>DBMS_PROFILER</td>
<td>Provides a Probe Profiler API to profile existing PL/SQL applications and identify performance bottlenecks.</td>
</tr>
<tr>
<td>DBMS_PROPAGATION_ADM</td>
<td>Provides administrative procedures for configuring propagation from a source queue to a destination queue</td>
</tr>
<tr>
<td>DBMS_QOPATCH</td>
<td>The DBMS_QOPATCH package provides an interface to view the installed database patches.</td>
</tr>
<tr>
<td>DBMS_RANDOM</td>
<td>Provides a built-in random number generator</td>
</tr>
<tr>
<td>DBMS_REDACT</td>
<td>The DBMS_REDACT package provides an interface to Oracle Data Redaction, which enables you to mask (redact) data that is returned from queries issued by lowprivileged users or an application.</td>
</tr>
<tr>
<td>DBMS_REDEFINITION</td>
<td>Lets you perform an online reorganization of tables</td>
</tr>
<tr>
<td>DBMS_REFRESH</td>
<td>Lets you create groups of snapshots that can be refreshed together to a transactionally consistent point in time Requires the Distributed Option</td>
</tr>
<tr>
<td>DBMS_REPAIR</td>
<td>Provides data corruption repair procedures</td>
</tr>
<tr>
<td>DBMS_RESCONFIG</td>
<td>Provides an interface to operate on the Resource Configuration List, and to retrieve listener information for a resource</td>
</tr>
<tr>
<td>DBMS_RESOURCE_MANAGER</td>
<td>Maintains plans, consumer groups, and plan directives; it also provides semantics so that you may group together changes to the plan schema</td>
</tr>
<tr>
<td>DBMS_RESOURCE_MANAGER_PRIVS</td>
<td>Maintains privileges associated with resource consumer groups</td>
</tr>
<tr>
<td>DBMS_RESULT_CACHE</td>
<td>Provides an interface to operate on the Result Cache</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DBMS_RESUMABLE</td>
<td>Lets you suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution.</td>
</tr>
<tr>
<td>DBMS_RLS</td>
<td>Provides row level security administrative interface</td>
</tr>
<tr>
<td>DBMS_ROLLING</td>
<td>The DBMS_ROLLING PL/SQL package is used to implement the Rolling Upgrade Using Active Data Guard feature, which streamlines the process of upgrading Oracle Database software in a Data Guard configuration in a rolling fashion.</td>
</tr>
<tr>
<td>DBMS_ROWID</td>
<td>Provides procedures to create rowids and to interpret their contents</td>
</tr>
<tr>
<td>DBMS_RULE</td>
<td>Describes the EVALUATE procedure used in Streams</td>
</tr>
<tr>
<td>DBMS_RULE_ADM</td>
<td>Describes the administrative interface for creating and managing rules, rule sets, and rule evaluation contexts; used in Streams</td>
</tr>
<tr>
<td>DBMS_SCHEDULER</td>
<td>Provides a collection of scheduling functions that are callable from any PL/SQL program</td>
</tr>
<tr>
<td>DBMS_SERVER_ALERT</td>
<td>Lets you issue alerts when some threshold has been violated</td>
</tr>
<tr>
<td>DBMS_SERVICE</td>
<td>Lets you create, delete, activate and deactivate services for a single instance</td>
</tr>
<tr>
<td>DBMS_SESSION</td>
<td>Provides access to SQL ALTER SESSION statements, and other session information, from stored procedures</td>
</tr>
<tr>
<td>DBMS_SHARED_POOL</td>
<td>Lets you keep objects in shared memory, so that they will not be aged out with the normal LRU mechanism</td>
</tr>
<tr>
<td>DBMS_SODA</td>
<td>Implements Simple Oracle Document Access (SODA) which allows you to use the Oracle Database as a NoSQL document store.</td>
</tr>
<tr>
<td>DBMS_SPACE</td>
<td>Provides segment space information not available through standard SQL</td>
</tr>
<tr>
<td>DBMS_SPACE_ADMIN</td>
<td>Provides tablespace and segment space administration not available through the standard SQL</td>
</tr>
<tr>
<td>DBMS_SPD</td>
<td>The DBMS_SPD package provides subprograms for managing SQL plan directives (SPD).</td>
</tr>
<tr>
<td>DBMS_SPM</td>
<td>Supports the SQL plan management feature by providing an interface for the DBA or other user to perform controlled manipulation of plan history and SQL plan baselines maintained for various SQL statements</td>
</tr>
<tr>
<td>DBMS_SQL</td>
<td>Lets you use dynamic SQL to access the database</td>
</tr>
<tr>
<td>DBMS_SQL_MONITOR</td>
<td>The DBMS_SQL_MONITOR package provides information about Real-time SQL Monitoring and Real-time Database Operation Monitoring.</td>
</tr>
<tr>
<td>DBMS_SQL_TRANSLATOR</td>
<td>The DBMS_SQL_TRANSLATOR package provides an interface for creating, configuring, and using SQL translation profiles.</td>
</tr>
<tr>
<td>DBMS_SQLDIAG</td>
<td>Provides an interface to the SQL Diagnosability functionality</td>
</tr>
<tr>
<td>DBMS_SQLPA</td>
<td>Provides an interface to implement the SQL Performance Analyzer.</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DBMS_SQLTUNE</td>
<td>Provides the interface to tune SQL statements</td>
</tr>
<tr>
<td>DBMS_STAT_FUNCS</td>
<td>Provides statistical functions</td>
</tr>
<tr>
<td>DBMS_STATS</td>
<td>Provides a mechanism for users to view and modify optimizer statistics</td>
</tr>
<tr>
<td>DBMS_STORAGE_MAP</td>
<td>Communicates with FMON to invoke mapping operations</td>
</tr>
<tr>
<td>DBMS_STREAMS</td>
<td>Describes the interface to convert SYS.AnyData objects into LCR objects and</td>
</tr>
<tr>
<td></td>
<td>an interface to annotate redo entries generated by a session with a binary</td>
</tr>
<tr>
<td></td>
<td>tag.</td>
</tr>
<tr>
<td>DBMS_STREAMS_ADMIN</td>
<td>Describes administrative procedures for adding and removing simple rules,</td>
</tr>
<tr>
<td></td>
<td>without transformations, for capture, propagation, and apply at the table,</td>
</tr>
<tr>
<td></td>
<td>schema, and database level</td>
</tr>
<tr>
<td>DBMS_STREAMS_ADVISOR_ADM</td>
<td>Provides an interface to gather information about an Oracle Streams</td>
</tr>
<tr>
<td></td>
<td>environment and advise database administrators based on the information</td>
</tr>
<tr>
<td></td>
<td>gathered.</td>
</tr>
<tr>
<td>DBMS_STREAMS_AUTH</td>
<td>Provides interfaces for granting privileges to Streams administrators and</td>
</tr>
<tr>
<td></td>
<td>revoking privileges from Streams administrators</td>
</tr>
<tr>
<td>DBMS_STREAMS_HANDLER_ADM</td>
<td>Provides interfaces to enqueue messages into and dequeue messages from a</td>
</tr>
<tr>
<td></td>
<td>SYS.AnyData queue</td>
</tr>
<tr>
<td>DBMS_STREAMS_MESSAGING</td>
<td>Provides interfaces to enqueue messages into and dequeue messages from a</td>
</tr>
<tr>
<td></td>
<td>SYS.AnyData queue</td>
</tr>
<tr>
<td>DBMS_STREAMS_TABLESPACE_ADM</td>
<td>Provides administrative procedures for copying tablespaces between</td>
</tr>
<tr>
<td></td>
<td>databases and moving tablespaces from one database to another</td>
</tr>
<tr>
<td>DBMS_SYNC_REFRESH</td>
<td>The DBMS_SYNC_REFRESH package provides an interface to perform a synchronous</td>
</tr>
<tr>
<td></td>
<td>refresh of materialized views.</td>
</tr>
<tr>
<td>DBMS_TDB</td>
<td>Reports whether a database can be transported between platforms using the RMAN</td>
</tr>
<tr>
<td></td>
<td>CONVERT DATABASE command. It verifies that databases on the current host</td>
</tr>
<tr>
<td></td>
<td>platform are of the same endian format as the destination platform, and that</td>
</tr>
<tr>
<td></td>
<td>the state of the current database does not prevent transport of the</td>
</tr>
<tr>
<td></td>
<td>database.</td>
</tr>
<tr>
<td>DBMS_TF</td>
<td>The DBMS_TF package contains utilities for POLYMORPHIC TABLE functions</td>
</tr>
<tr>
<td></td>
<td>(PTFs) implementation. You can use DBMS_TF subprograms to consume and</td>
</tr>
<tr>
<td></td>
<td>produce data, and get information about its execution environment.</td>
</tr>
<tr>
<td>DBMS_TNS</td>
<td>The DBMS_TNS package provides the RESOLVE_TNSNAME function to resolve a TNS</td>
</tr>
<tr>
<td></td>
<td>name and return the corresponding Oracle Net8 connection string.</td>
</tr>
<tr>
<td>DBMS_TRACE</td>
<td>Provides routines to start and stop PL/SQL tracing</td>
</tr>
<tr>
<td>DBMS_TRANSACTION</td>
<td>Provides access to SQL transaction statements from stored procedures and</td>
</tr>
<tr>
<td></td>
<td>monitors transaction activities</td>
</tr>
<tr>
<td>DBMS_TRANSFORM</td>
<td>Provides an interface to the message format transformation features of</td>
</tr>
<tr>
<td></td>
<td>Oracle Advanced Queuing</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DBMS_TSDP_MANAGE</td>
<td>The DBMS_TSDP_MANAGE package provides an interface to import and manage sensitive columns and sensitive column types in the database.</td>
</tr>
<tr>
<td>DBMS_TSDP_PROTECT</td>
<td>The DBMS_TSDP_PROTECT package provides an interface to configure transparent sensitive data protection (TSDP) policies in conjunction with the DBMS_TSDP_MANAGE package. The DBMS_TSDP_PROTECT package is available with the Enterprise Edition only.</td>
</tr>
<tr>
<td>DBMS_TTS</td>
<td>Checks if the transportable set is self-contained</td>
</tr>
<tr>
<td>DBMS_TYPES</td>
<td>Consists of constants, which represent the built-in and user-defined types</td>
</tr>
<tr>
<td>DBMS_UMF</td>
<td>The DBMS_UMF package provides an interface for deploying the Remote Management Framework (RMF) for an Oracle Database. The RMF is used for collecting performance statistics for an Oracle Database.</td>
</tr>
<tr>
<td>DBMS_UTILITY</td>
<td>Provides various utility routines</td>
</tr>
<tr>
<td>DBMS_WARNING</td>
<td>Provides the interface to query, modify and delete current system or session settings</td>
</tr>
<tr>
<td>DBMS_WM</td>
<td>Describes how to use the programming interface to Oracle Database Workspace Manager to work with long transactions</td>
</tr>
<tr>
<td>DBMS_WORKLOAD_CAPTURE</td>
<td>Configures the Workload Capture system and produce the workload capture data.</td>
</tr>
<tr>
<td>DBMS_WORKLOAD_REPLAY</td>
<td>Provides an interface to replay and report on a record of a workload on a production or test system</td>
</tr>
<tr>
<td>DBMS_WORKLOAD_REPOSITORY</td>
<td>Lets you manage the Workload Repository, performing operations such as managing snapshots and baselines</td>
</tr>
<tr>
<td>DBMS_XA</td>
<td>Contains the XA/Open interface for applications to call XA interface in PL/SQL</td>
</tr>
<tr>
<td>DBMS_XDB</td>
<td>Describes Resource Management and Access Control interface for PL/SQL</td>
</tr>
<tr>
<td>DBMS_XDB_ADMIN</td>
<td>Provides an interface to implement XMLIndex administration operation</td>
</tr>
<tr>
<td>DBMS_XDB_CONFIG</td>
<td>The DBMS_XDB_CONFIG package provides an interface for configuring Oracle XML DB and its repository.</td>
</tr>
<tr>
<td>DBMS_XDB_CONSTANTS</td>
<td>The DBMS_XDB_CONSTANTS package provides an interface to commonly used constants.</td>
</tr>
<tr>
<td>DBMS_XDB_REPOS</td>
<td>The DBMS_XDB_REPOS package provides an interface to operate on the Oracle XML database Repository.</td>
</tr>
<tr>
<td>DBMS_XDBRESOURCE</td>
<td>Provides an interface to operate on the XDB resource’s metadata and contents</td>
</tr>
<tr>
<td>DBMS_XDB_VERSION</td>
<td>Describes the versioning interface</td>
</tr>
</tbody>
</table>
| DBMS_XDBT           | Describes how an administrator can create a ConText index on the XML DB hierarchy and configure it for automatic maintenance
<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_XDBZ</td>
<td>Controls the Oracle XML DB repository security, which is based on Access Control Lists (ACLs)</td>
</tr>
<tr>
<td>DBMS_XEVENT</td>
<td>Provides event-related types and supporting subprograms</td>
</tr>
<tr>
<td>DBMS_XMLDOM</td>
<td>Explains access to XMLType objects</td>
</tr>
<tr>
<td>DBMS_XMLGEN</td>
<td>Converts the results of a SQL query to a canonical XML format</td>
</tr>
<tr>
<td>DBMS_XMLINDEX</td>
<td>Provides an interface to implement asynchronous indexing and apply node referencing</td>
</tr>
<tr>
<td>DBMS_XMLPARSER</td>
<td>Explains access to the contents and structure of XML documents</td>
</tr>
<tr>
<td>DBMS_XMLQUERY</td>
<td>Provides database-to-XMLType functionality</td>
</tr>
<tr>
<td>DBMS_XMLESAVE</td>
<td>Provides XML-to-database-type functionality</td>
</tr>
<tr>
<td>DBMS_XMLSCHEMA</td>
<td>Explains procedures to register and delete XML schemas</td>
</tr>
<tr>
<td>DBMS_XMLSTORE</td>
<td>Provides the ability to store XML data in relational tables</td>
</tr>
<tr>
<td>DBMS_XMLTRANSLATIONS</td>
<td>Provides an interface to perform translations so that strings can be searched or displayed in various languages</td>
</tr>
<tr>
<td>DBMS_XPLAN</td>
<td>Describes how to format the output of the EXPLAIN PLAN command</td>
</tr>
<tr>
<td>DBMS_XSLPROCESSOR</td>
<td>Explains access to the contents and structure of XML documents</td>
</tr>
<tr>
<td>DEBUG_EXTPROC</td>
<td>Lets you debug external procedures on platforms with debuggers that attach to a running process</td>
</tr>
<tr>
<td>HTF</td>
<td>Hypertext functions generate HTML tags</td>
</tr>
<tr>
<td>HTP</td>
<td>Hypertext procedures generate HTML tags</td>
</tr>
<tr>
<td>ORD_DICOM</td>
<td>Supports the management and manipulation of Digital Imaging and Communications in Medicine (DICOM) content stored in BLOBs or BFILES rather than in an ORDDicom object type</td>
</tr>
<tr>
<td>ORD_DICOM_ADMIN</td>
<td>Used by Oracle Multimedia Digital Imaging and Communications in Medicine (DICOM) administrators to maintain the Oracle Multimedia DICOM data model repository</td>
</tr>
<tr>
<td>OWA_CACHE</td>
<td>Provides an interface that enables the PL/SQL Gateway cache to improve the performance of PL/SQL Web applications</td>
</tr>
<tr>
<td>OWA_COOKIE</td>
<td>Provides an interface for sending and retrieving HTTP cookies from the client’s browser</td>
</tr>
<tr>
<td>OWA_CUSTOM</td>
<td>Provides a Global PLSQL Agent Authorization callback function</td>
</tr>
<tr>
<td>OWA_IMAGE</td>
<td>Provides an interface to access the coordinates where a user clicked on an image</td>
</tr>
<tr>
<td>OWA_OPT_LOCK</td>
<td>Contains subprograms that impose optimistic locking strategies so as to prevent lost updates</td>
</tr>
<tr>
<td>OWA_PATTERN</td>
<td>Provides an interface to locate text patterns within strings and replace the matched string with another string</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OWA_SEC</td>
<td>Provides an interface for custom authentication</td>
</tr>
<tr>
<td>OWA_TEXT</td>
<td>Contains subprograms used by OWA_PATTERN for manipulating strings. They are externalized so you can use them directly.</td>
</tr>
<tr>
<td>OWA_UTIL</td>
<td>Contains utility subprograms for performing operations such as getting the value of CGI environment variables, printing the data that is returned to the client, and printing the results of a query in an HTML table</td>
</tr>
<tr>
<td>SDO_CS</td>
<td>Provides functions for coordinate system transformation</td>
</tr>
<tr>
<td>SDO_CSW_PROCESS</td>
<td>Contains subprograms for various processing operations related to support for Catalog Services for the Web (CSW)</td>
</tr>
<tr>
<td>SDO_GCDR</td>
<td>Contains the Oracle Spatial geocoding subprograms, which let you geocode unformatted postal addresses</td>
</tr>
<tr>
<td>SDO_GEOM</td>
<td>Provides functions implementing geometric operations on spatial objects</td>
</tr>
<tr>
<td>SDO_GEOR</td>
<td>Contains functions and procedures for the Spatial GeoRaster feature, which lets you store, index, query, analyze, and deliver raster image data and its associated Spatial vector geometry data and metadata</td>
</tr>
<tr>
<td>SDO_GEOR_ADMIN</td>
<td>Contains subprograms for administrative operations related to GeoRaster.</td>
</tr>
<tr>
<td>SDO_GEOR_UTL</td>
<td>Contains utility functions and procedures for the Spatial GeoRaster feature, including those related to using triggers with GeoRaster data</td>
</tr>
<tr>
<td>SDO_LRS</td>
<td>Provides functions for linear referencing system support</td>
</tr>
<tr>
<td>SDO_MIGRATE</td>
<td>Provides functions for migrating spatial data from previous releases</td>
</tr>
<tr>
<td>SDO_NET</td>
<td>Provides functions and procedures for working with data modeled as nodes and links in a network</td>
</tr>
<tr>
<td>SDO_NET_MEM</td>
<td>Contains functions and procedures for performing editing and analysis operations on network data using a network memory object</td>
</tr>
<tr>
<td>SDO_OLS</td>
<td>Contains functions and procedures for performing editing and analysis operations on network data using a network memory object</td>
</tr>
<tr>
<td>SDO_PC_PKG</td>
<td>Contains subprograms to support the use of point clouds in Spatial</td>
</tr>
<tr>
<td>SDO_SAM</td>
<td>Contains functions and procedures for spatial analysis and data mining</td>
</tr>
<tr>
<td>SDO_TIN_PKG</td>
<td>Contains subprograms to support the use of triangulated irregular networks (TINs) in Spatial</td>
</tr>
<tr>
<td>SDO_TOPO</td>
<td>Provides procedures for creating and managing Spatial topologies</td>
</tr>
<tr>
<td>SDO_TOPO_MAP</td>
<td>Contains subprograms for editing Spatial topologies using a cache (TopoMap object)</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SDO_TUNE</td>
<td>Provides functions for selecting parameters that determine the behavior of</td>
</tr>
<tr>
<td></td>
<td>the spatial indexing scheme used in Oracle Spatial</td>
</tr>
<tr>
<td>SDO_UTIL</td>
<td>Provides utility functions and procedures for Oracle Spatial</td>
</tr>
<tr>
<td>SDO_WFS_LOCK</td>
<td>Contains subprograms for WFS support for registering and unregistering</td>
</tr>
<tr>
<td></td>
<td>feature tables</td>
</tr>
<tr>
<td>SDO_WFS_PROC</td>
<td>Provides utility functions and procedures for Oracle Spatial</td>
</tr>
<tr>
<td>SEM_APIS</td>
<td>Contains subprograms for working with the Resource Description Framework</td>
</tr>
<tr>
<td></td>
<td>(RDF) and Web Ontology Language (OWL) in an Oracle database.</td>
</tr>
<tr>
<td>SEM_PERF</td>
<td>Contains subprograms for examining and enhancing the performance of the</td>
</tr>
<tr>
<td></td>
<td>Resource Description Framework (RDF) and Web Ontology Language (OWL)</td>
</tr>
<tr>
<td></td>
<td>support in an Oracle database.</td>
</tr>
<tr>
<td>SEM_RDFCTX</td>
<td>Contains subprograms for managing extractor policies and semantic indexes</td>
</tr>
<tr>
<td></td>
<td>created for documents</td>
</tr>
<tr>
<td>SEM_RDFSA</td>
<td>Contains subprograms for providing fine-grained access control to RDF</td>
</tr>
<tr>
<td></td>
<td>data, using either a virtual private database (VPD) or Oracle Label</td>
</tr>
<tr>
<td></td>
<td>Security (OLS)</td>
</tr>
<tr>
<td>UTL_COLL</td>
<td>Enables PL/SQL programs to use collection locators to query and update</td>
</tr>
<tr>
<td>UTL_COMPRESS</td>
<td>Provides a set of data compression utilities</td>
</tr>
<tr>
<td>UTL_DBWS</td>
<td>Provides database Web services</td>
</tr>
<tr>
<td>UTL_ENCODE</td>
<td>Provides functions that encode RAW data into a standard encoded format so</td>
</tr>
<tr>
<td></td>
<td>that the data can be transported between hosts</td>
</tr>
<tr>
<td>UTL_FILE</td>
<td>Enables your PL/SQL programs to read and write operating system text</td>
</tr>
<tr>
<td></td>
<td>files and provides a restricted version of standard operating system stream</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
</tr>
<tr>
<td>UTL_HTTP</td>
<td>Enables HTTP callouts from PL/SQL and SQL to access data on the Internet</td>
</tr>
<tr>
<td></td>
<td>or to call Oracle Web Server Cartridges</td>
</tr>
<tr>
<td>UTL_I18N</td>
<td>Provides a set of services (Oracle Globalization Service) that help</td>
</tr>
<tr>
<td></td>
<td>developers build multilingual applications</td>
</tr>
<tr>
<td>UTL_INADDR</td>
<td>Provides a procedure to support internet addressing</td>
</tr>
<tr>
<td>UTL_IDENT</td>
<td>Specifies which database or client PL/SQL is running</td>
</tr>
<tr>
<td>UTL_LMS</td>
<td>Retrieves and formats error messages in different languages</td>
</tr>
<tr>
<td>UTL_MAIL</td>
<td>A utility for managing email which includes commonly used email features,</td>
</tr>
<tr>
<td></td>
<td>such as attachments, CC, BCC, and return receipt</td>
</tr>
<tr>
<td>UTL_NLA</td>
<td>Exposes a subset of the BLAS and LAPACK (Version 3.0) operations on</td>
</tr>
<tr>
<td></td>
<td>vectors and matrices represented as VARRAYs</td>
</tr>
<tr>
<td>UTL_RAW</td>
<td>Provides SQL functions for manipulating RAW datatypes</td>
</tr>
<tr>
<td>UTL_RECOMP</td>
<td>Recompiles invalid PL/SQL modules, invalid views, Java classes, index</td>
</tr>
<tr>
<td></td>
<td>types and operators in a database, either sequentially or in parallels</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UTL_REF</td>
<td>Enables a PL/SQL program to access an object by providing a reference to the object</td>
</tr>
<tr>
<td>UTL_SMTP</td>
<td>Provides PL/SQL functionality to send emails</td>
</tr>
<tr>
<td>UTL_SPADV</td>
<td>Provides subprograms to collect and analyze statistics for the Oracle Streams components in a distributed database environment</td>
</tr>
<tr>
<td>UTL_TCP</td>
<td>Provides PL/SQL functionality to support simple TCP/IP-based communications between servers and the outside world</td>
</tr>
<tr>
<td>UTL_URL</td>
<td>Provides escape and unescape mechanisms for URL characters</td>
</tr>
<tr>
<td>WPG_DOCLOAD</td>
<td>Provides an interface to download files, both BLOBs and BFILEs</td>
</tr>
<tr>
<td>ANYDATA TYPE</td>
<td>A self-describing data instance type containing an instance of the type plus a description</td>
</tr>
<tr>
<td>ANYDATASET TYPE</td>
<td>Contains a description of a given type plus a set of data instances of that type</td>
</tr>
<tr>
<td>ANYTYPE TYPE</td>
<td>Contains a type description of any persistent SQL type, named or unnamed, including object types and collection types; or, it can be used to construct new transient type descriptions</td>
</tr>
<tr>
<td>Oracle Database Advanced Queuing Types</td>
<td>Describes the types used in Advanced Queuing</td>
</tr>
<tr>
<td>DBFS Content Interface Types</td>
<td>Describes public types defined to support the DBMS_DBFS_CONTENT interface.</td>
</tr>
<tr>
<td>Database URI Type</td>
<td>Contains URI Support, UriType Super Type, HttpUriType Subtype, DBUriType Subtype, XDBUriType Subtype, UriFactory Package</td>
</tr>
<tr>
<td>Expression Filter Types</td>
<td>Expression Filter feature is supplied with a set of predefined types and public synonyms for these types.</td>
</tr>
<tr>
<td>JMS TYPES</td>
<td>Describes JMS types so that a PL/SQL application can use JMS queues of JMS types</td>
</tr>
<tr>
<td>Oracle Multimedia ORDAudio TYPE</td>
<td>Supports the storage and management of audio data</td>
</tr>
<tr>
<td>Oracle Multimedia ORDDicom Type</td>
<td>Supports the storage, management, and manipulation of Digital Imaging and Communications in Medicine (DICOM) data</td>
</tr>
<tr>
<td>Oracle Multimedia ORDDoc TYPE</td>
<td>Supports the storage and management of heterogeneous media data including image, audio, and video</td>
</tr>
<tr>
<td>Oracle Multimedia ORDImage TYPE</td>
<td>Supports the storage, management, and manipulation of image data</td>
</tr>
<tr>
<td>Oracle Multimedia SQL/MM Still Image TYPE</td>
<td>Supports the SQL/MM Still Image Standard, which lets you store, retrieve, and modify images in the database and locate images using visual predicates</td>
</tr>
<tr>
<td>Oracle Multimedia ORDVideo TYPE</td>
<td>Supports the storage and management of video data</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOGICAL CHANGE RECORD TYPES</td>
<td>Describes LCR types, which are message payloads that contain information about changes to a database, used in Streams</td>
</tr>
<tr>
<td>MG_ID Package Types</td>
<td>Provides an extensible framework that supports current RFID tags with the standard family of EPC bit encodings for the supported encoding types</td>
</tr>
<tr>
<td>POLYMORPHIC TABLE FUNCTION (PTF) Package Types</td>
<td>Describes types defined in the DBMS_TF package to support PTF</td>
</tr>
<tr>
<td>RULES TYPES</td>
<td>Describes the types used with rules, rule sets, and evaluation contexts</td>
</tr>
<tr>
<td>RULES Manager Types</td>
<td>Rules Manager is supplied with one predefined type and a public synonym</td>
</tr>
<tr>
<td>SODA Types</td>
<td>Describes the SODA Types.</td>
</tr>
<tr>
<td>UTL Streams Types</td>
<td>Describes abstract streams types used with Oracle XML functionality</td>
</tr>
<tr>
<td>XMLType</td>
<td>Describes the types and functions used for native XML support in the server</td>
</tr>
</tbody>
</table>
Oracle Application Express Packages
APEX_APPLICATION Through APEX_ZIP

The Oracle Application Express supplies PL/SQL packages for application developers who are building database-centric web applications using Oracle Application Express.

For a complete description of all the packages that ship with Oracle Application Express, see the *Oracle Application Express API Reference*. 
The CTX_ADM package lets you administer the Oracle Text data dictionary.

Note that you must install this package in order to use it.

For a complete description of this package within the context of Oracle Text, see CTX_ADM in the Oracle Text Reference.
The `CTX_ANL` package is used with AUTO_LEXER and provides procedures for adding and dropping a custom dictionary from the lexer.

A custom dictionary might be one that you develop for a special field of study or for your industry. In most cases, the dictionaries supplied for the supported languages with Oracle Text are more than sufficient to handle your requirements.

For a complete description of this package within the context of Oracle Text, see `CTX_ANL` in the Oracle Text Reference.
The CTX_CLS package enables generation of CTXRULE rules for a set of documents.

For a complete description of this package within the context of Oracle Text, see CTX_CLS in the Oracle Text Reference.
The CTX_DDL package lets you create and manage the preferences, section groups, and stoplists required for Text indexes.

Note that you must install this package in order to use it.

For complete description of this package within the context of Oracle Text, see CTX_DDL in the Oracle Text Reference.
The **CTX_DOC** package lets you request document services.

Note that you must install this package in order to use it.

For a complete description of this package within the context of Oracle Text, see **CTX_DOC** in
the Oracle Text Reference.
The **CTX_ENTITY** package enables you to search for terms that are unknown to you without specifying a particular search text.

It does this by identifying names, places, dates, and other objects when they are mentioned in a document and by tagging each occurrence (called a mention) with its type and subtype. This process enables you to produce a structured view of a document that can later be used for text and data mining and more comprehensive intelligence analysis.

For the complete description of this package within the context of Oracle Text, see **CTX_ENTITY** in the *Oracle Text Reference*. 
This Oracle Text package lets you manage the index log.

Note that you must install this package in order to use it.

For a complete description of this package within the context of Oracle Text, see CTX_OUTPUT in the Oracle Text Reference.
This Oracle Text package lets you generate query feedback, count hits, and create stored query expressions.

Note that you must install this package in order to use it.

For a complete description of this package within the context of Oracle Text, see CTX_QUERY in the Oracle Text Reference.
This Oracle Text package lets you create various index reports.

Note that you must install this package in order to use it.

For a complete description of this package within the context of Oracle Text, see `CTX_REPORT` in the Oracle Text Reference.
This Oracle Text package lets you to manage and browse thesauri.

Note that you must install this package in order to use it.

For a complete description of this package within the context of Oracle Text, see CTX_THES in the Oracle Text Reference.
13

CTX_ULEXER

This Oracle Text package is for use with the user-lexer.

Note that you must install this package in order to use it.

For a complete description of this package within the context of Oracle Text, see CTX_ULEXER in the Oracle Text Reference.
The DBMS_ADDM package facilitates the use of Advisor functionality regarding the Automatic Database Diagnostic Monitor.

This chapter contains the following topics:

- Security Model
- Summary of DBMS_ADDM Subprograms

**See Also:**

- Oracle Real Application Clusters Administration and Deployment Guide for more information about “Automatic Workload Repository in Oracle Real Application Clusters Environments”
- Oracle Database Performance Tuning Guide for more information about “Automatic Performance Diagnostics”

### 14.1 DBMS_ADDM Security Model

The DBMS_ADDM package runs with the caller's permission, not the definer's, and then applies the security constraints required by the DBMS_ADVISOR package.

**See Also:**

The DBMS_ADVISOR package for more information about "Security Model".

### 14.2 Summary of DBMS_ADDM Subprograms

The table in this topic lists and describes the DBMS_ADDM subprograms.

**Table 14-1  DBMS_ADDM Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZE_DB Procedure</td>
<td>Creates an ADDM task for analyzing in database analysis mode and executes it</td>
</tr>
<tr>
<td>ANALYZE_INST Procedure</td>
<td>Creates an ADDM task for analyzing in instance analysis mode and executes it</td>
</tr>
<tr>
<td>ANALYZE_PARTIAL Procedure</td>
<td>Creates an ADDM task for analyzing a subset of instances in partial analysis mode and executes it</td>
</tr>
</tbody>
</table>
Table 14-1  (Cont.) DBMS_ADDM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPARE_CAPTURE_REPLAY_REPORT Function</td>
<td>Produces a Compare Period ADDM report comparing the performance of a capture to a replay</td>
</tr>
<tr>
<td>COMPARE_DATABASES Function</td>
<td>Produces a Compare Period ADDM report for a database-wide performance comparison</td>
</tr>
<tr>
<td>COMPARE_INSTANCES Function</td>
<td>Produces a Compare Period ADDM report for an instance-level performance comparison</td>
</tr>
<tr>
<td>COMPARE_REPLAY_REPLAY_REPORT Function</td>
<td>Produces a Compare Period ADDM report comparing the performance of a replay to another replay</td>
</tr>
<tr>
<td>DELETE Procedure</td>
<td>Deletes an already created ADDM task (of any kind)</td>
</tr>
<tr>
<td>DELETE_FINDING_DIRECTIVE Procedure</td>
<td>Deletes a finding directive</td>
</tr>
<tr>
<td>DELETE_PARAMETER_DIRECTIVE Procedure</td>
<td>Deletes a parameter directive</td>
</tr>
<tr>
<td>DELETE_SEGMENT_DIRECTIVE Procedure</td>
<td>Deletes a segment directive</td>
</tr>
<tr>
<td>DELETE_SQL_DIRECTIVE Procedure</td>
<td>Deletes a SQL directive</td>
</tr>
<tr>
<td>GET_ASH_QUERY Function</td>
<td>Returns a string containing the SQL text of an ASH query identifying the rows in ASH with impact for the finding</td>
</tr>
<tr>
<td>GET_REPORT Function</td>
<td>Retrieves the default text report of an executed ADDM task</td>
</tr>
<tr>
<td>INSERT_FINDING_DIRECTIVE Procedure</td>
<td>Creates a directive to limit reporting of a specific finding type.</td>
</tr>
<tr>
<td>INSERT_PARAMETER_DIRECTIVE Procedure</td>
<td>Creates a directive to prevent ADDM from creating actions to alter the value of a specific system parameter</td>
</tr>
<tr>
<td>INSERT_SEGMENT_DIRECTIVE Procedure</td>
<td>Creates a directive to prevent ADDM from creating actions to &quot;run Segment Advisor&quot; for specific segments</td>
</tr>
<tr>
<td>INSERT_SQL_DIRECTIVE Procedure</td>
<td>Creates a directive to limit reporting of actions on specific SQL</td>
</tr>
<tr>
<td>REAL_TIME_ADDM_REPORT Function</td>
<td>Produces a real-time report of ADDM activity</td>
</tr>
</tbody>
</table>

14.2.1 ANALYZE_DB Procedure

This procedure creates an ADDM task for analyzing in database analysis mode and executes it.

Syntax

```sql
DBMS_ADDM.ANALYZE_DB (  
    task_name            IN OUT VARCHAR2,  
    begin_snapshot       IN     NUMBER,  
    end_snapshot         IN     NUMBER,  
    db_id                IN     NUMBER := NULL);  
```
Parameters

Table 14-2 ANALYZE_DB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task to be created</td>
</tr>
<tr>
<td>begin_snapshot</td>
<td>Number of the snapshot that starts the analysis period</td>
</tr>
<tr>
<td>end_snapshot</td>
<td>Number of the snapshot that ends the analysis period</td>
</tr>
<tr>
<td>db_id</td>
<td>Database ID for the database you to analyze. By default, this is the database currently connected</td>
</tr>
</tbody>
</table>

Return Values

The name of the created task is returned in the task_name parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

Examples

To create an ADDM task in database analysis mode and execute it, with its name in variable tname:

```sql
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_database_analysis_mode_task';
  DBMS_ADDM.ANALYZE_DB(:tname, 1, 2);
END
```

To see a report:

```
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a CLOB, formatted to fit line size of 80.

14.2.2 ANALYZE_INST Procedure

This procedure creates an ADDM task for analyzing in instance analysis mode and executes it.

Syntax

```sql
DBMS_ADDM.ANALYZE_INST (  
  task_name IN OUT VARCHAR2,  
  begin_snapshot IN NUMBER,  
  end_snapshot IN NUMBER,  
  instance_number IN NUMBER := NULL,  
  db_id IN NUMBER := NULL);  
```
Parameters

Table 14-3  ANALYZE_INST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task to be created</td>
</tr>
<tr>
<td>begin_snapshot</td>
<td>Number of the snapshot that starts the analysis period</td>
</tr>
<tr>
<td>end_snapshot</td>
<td>Number of the snapshot that ends the analysis period</td>
</tr>
<tr>
<td>instance_number</td>
<td>Number of the instance to analyze. By default it is the instance currently connected</td>
</tr>
<tr>
<td>db_id</td>
<td>Database ID for the database you to analyze. By default, this is the database currently connected</td>
</tr>
</tbody>
</table>

Return Values

The name of the created task is returned in the task_name parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

Usage Notes

On single instance systems (when not using Oracle RAC) the resulting task is identical to using the ANALYZE_DB procedure.

Examples

To create an ADDM task in instance analysis mode and execute it, with its name in variable tname:

```sql
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END
```

To see a report:

```sql
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a CLOB, formatted to fit line size of 80.

14.2.3 ANALYZE_PARTIAL Procedure

This procedure creates an ADDM task for analyzing a subset of instances in partial analysis mode and executes it.

Syntax

```sql
DBMS_ADDM.ANALYZE_PARTIAL (    task_name      IN OUT VARCHAR2,    instance_numbers IN VARCHAR2,    begin_snapshot   IN NUMBER,    ...)
```
Parameters

Table 14-4  ANALYZE_PARTIAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task to be created</td>
</tr>
<tr>
<td>instance_numbers</td>
<td>Comma separated list of instance numbers to analyze</td>
</tr>
<tr>
<td>begin_snapshot</td>
<td>Number of the snapshot that starts the analysis period</td>
</tr>
<tr>
<td>end_snapshot</td>
<td>Number of the snapshot that ends the analysis period</td>
</tr>
<tr>
<td>db_id</td>
<td>Database ID for the database you to analyze. By default, this is the database currently connected</td>
</tr>
</tbody>
</table>

Return Values

The name of the created task is returned in the task_name parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

Examples

To create an ADDM task in partial analysis mode and execute it, with its name in variable tname:

```sql
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_partial_analysis_modetask';
  DBMS_ADDM.ANALYZE_PARTIAL(:tname, '1,2,3', 1, 2);
END
```

To see a report:

```sql
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a CLOB, formatted to fit line size of 80.

14.2.4 COMPARE_CAPTURE_REPLAY_REPORT Function

This function produces a Compare Period ADDM report comparing the performance of a capture to a replay.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

Syntax

```sql
DBMS_ADDM.COMPRESS_CAPTURE_REPLAY_REPORT (  
  replay_id             IN NUMBER,  
  report_type           IN VARCHAR2 := 'HTML')  
RETURN CLOB;
```
Parameters

Table 14-5  COMPARE_CAPTURE_REPLAY_REPORT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id</td>
<td>Replay ID to use as the base period. The base period is the baseline period to compare in order to determine improvement or regression.</td>
</tr>
<tr>
<td>report_type</td>
<td>'HTML' (the default) for an HTML active report, 'XML' for an XML report</td>
</tr>
</tbody>
</table>

Return Values

A CLOB containing a compare period ADDM report

14.2.5 COMPARE_DATABASES Function

This function produces a Compare Period ADDM report comparing the performance of a database over two different time periods or the performance of two different databases over two different time periods.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

Syntax

```
DBMS_ADDM.COMPARE_DATABASES ( 
    base_dbid             IN NUMBER := NULL, 
    base_begin_snap_id    IN NUMBER, 
    base_end_snap_id      IN NUMBER, 
    comp_dbid             IN NUMBER := NULL, 
    comp_begin_snap_id    IN NUMBER, 
    comp_end_snap_id      IN NUMBER, 
    report_type           IN VARCHAR2 := 'HTML')
RETURN CLOB;
```

Parameters

Table 14-6  COMPARE_DATABASES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_dbid</td>
<td>Database id (DBID) of the base period. The base period is the baseline period that we compare to in order to determine improvement or regression.</td>
</tr>
<tr>
<td>base_begin_snap_id</td>
<td>Begin AWR snapshot ID of the base period.</td>
</tr>
<tr>
<td>base_end_snap_id</td>
<td>End AWR snapshot ID of the base period.</td>
</tr>
<tr>
<td>comp_dbid</td>
<td>Database id (DBID) of the comparison period. The comparison period is the period we compare to the base period.</td>
</tr>
<tr>
<td>comp_begin_snap_id</td>
<td>Begin AWR snapshot ID of the comparison period</td>
</tr>
</tbody>
</table>
### Table 14-6 (Cont.) COMPARE_DATABASES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp_end_snap_id</td>
<td>End AWR snapshot ID of the comparison period</td>
</tr>
<tr>
<td>report_type</td>
<td>'HTML' (the default) for an HTML active report, 'XML' for an XML report</td>
</tr>
</tbody>
</table>

#### Return Values

A **CLOB** containing a compare period ADDM report.

### 14.2.6 COMPARE_INSTANCES Function

This function produces a Compare Period ADDM report comparing the performance of a single instance over two different time periods or the performance of two different instances over two different time periods.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

#### Syntax

```sql
DBMS_ADDM.COMPARE_INSTANCES (
    base_dbid             IN NUMBER := NULL,
    base_instance_id      IN NUMBER,
    base_begin_snap_id    IN NUMBER,
    base_end_snap_id      IN NUMBER,
    comp_dbid             IN NUMBER := NULL,
    comp_instance_id      IN NUMBER,
    comp_begin_snap_id    IN NUMBER,
    comp_end_snap_id      IN NUMBER,
    report_type           IN VARCHAR2 := 'HTML')
RETURN CLOB;
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_dbid</td>
<td>Database id (DBID) of the base period. The base period is the baseline period that we compare to in order to determine improvement or regression.</td>
</tr>
<tr>
<td>base_instance_id</td>
<td>Instance number of the database instance to include from the base period</td>
</tr>
<tr>
<td>base_begin_snap_id</td>
<td>Begin AWR snapshot ID of the base period.</td>
</tr>
<tr>
<td>base_end_snap_id</td>
<td>End AWR snapshot ID of the base period.</td>
</tr>
<tr>
<td>comp_dbid</td>
<td>Database id (DBID) of the comparison period. The comparison period is the period we compare to the base period.</td>
</tr>
<tr>
<td>comp_instance_id</td>
<td>Instance number of the database instance to include from the comparison period</td>
</tr>
<tr>
<td>comp_begin_snap_id</td>
<td>Begin AWR snapshot ID of the comparison period</td>
</tr>
<tr>
<td>comp_end_snap_id</td>
<td>End AWR snapshot ID of the comparison period</td>
</tr>
</tbody>
</table>
Table 14-7  (Cont.) COMPARE_INSTANCES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report_type</td>
<td>'HTML' (the default) for an HTML active report, 'XML' for an XML report</td>
</tr>
</tbody>
</table>

**Return Values**

A CLOB containing a compare period ADDM report

### 14.2.7 COMPARE_REPLAY_REPLAY_REPORT Function

This function produces a Compare Period ADDM report comparing the performance of a replay to another replay.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

**Syntax**

```sql
DBMS_ADDM.COMPARE_CAPTURE_REPLAY_REPORT (replay_id1            IN NUMBER,
                                         replay_id2            IN NUMBER,
                                         report_type           IN VARCHAR2 := 'HTML')
RETURN CLOB;
```

**Parameters**

Table 14-8  COMPARE_REPLAY_REPLAY_REPORT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id1</td>
<td>Replay ID to use as the base period. The base period is the baseline period to compare in order to determine improvement or regression.</td>
</tr>
<tr>
<td>replay_id2</td>
<td>Replay ID to use as the comparison period. The comparison period is the period to compare to the base period in order to determine improvement or regression.</td>
</tr>
<tr>
<td>report_type</td>
<td>'HTML' (the default) for an HTML active report, 'XML' for an XML report</td>
</tr>
</tbody>
</table>

**Return Values**

A CLOB containing a compare period ADDM report

### 14.2.8 DELETE Procedure

This procedure deletes an already created ADDM task (of any kind). For database analysis mode and partial analysis mode this deletes the local tasks associated with the main task.

**Syntax**

```sql
DBMS_ADDM.DELETE (task_name IN VARCHAR2);)
```
Parameters

Table 14-9  DELETE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task to be deleted</td>
</tr>
</tbody>
</table>

Examples

BEGIN
  DBMS_ADDM.DELETE ('my_partial_analysis_mode_task');
END

14.2.9 DELETE_FINDING_DIRECTIVE Procedure

This procedure deletes a finding directive.

Syntax

DBMS_ADDM.DELETE_FINDING_DIRECTIVE (
  task_name           IN VARCHAR2,
  dir_name            IN VARCHAR2);

Parameters

Table 14-10  DELETE_FINDING_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it is a system directive.</td>
</tr>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
</tbody>
</table>

14.2.10 DELETE_PARAMETER_DIRECTIVE Procedure

This procedure deletes a parameter directive. This removes a specific system directive for parameters. Subsequent ADDM tasks are not affected by this directive.

Syntax

DBMS_ADDM.DELETE_PARAMETER_DIRECTIVE (
  task_name           IN VARCHAR2,
  dir_name            IN VARCHAR2);

Parameters

Table 14-11  DELETE_PARAMETER_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it is a system directive.</td>
</tr>
</tbody>
</table>
Table 14-11  (Cont.) DELETE_PARAMETER_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
</tbody>
</table>

Examples

```sql
BEGIN
    DBMS_ADDM.DELETE_PARAMETER_DIRECTIVE (NULL,'my Parameter directive');
END;
```

14.2.11 DELETE_SEGMENT_DIRECTIVE Procedure

This procedure deletes a segment directive.

Syntax

```sql
DBMS_ADDM.DELETE_SEGMENT_DIRECTIVE (  
    task_name IN VARCHAR2,  
    dir_name IN VARCHAR2);  
```

Parameters

Table 14-12  DELETE_SEGMENT_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it is a system directive.</td>
</tr>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
</tbody>
</table>

14.2.12 DELETE_SQL_DIRECTIVE Procedure

This procedure deletes a SQL directive.

Syntax

```sql
DBMS_ADDM.DELETE_SQL_DIRECTIVE (  
    task_name IN VARCHAR2,  
    dir_name IN VARCHAR2);  
```

Parameters

Table 14-13  DELETE_SQL_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it is a system directive.</td>
</tr>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
</tbody>
</table>
14.2.13 GET_ASH_QUERY Function

The function returns a string containing the SQL text of an ASH query identifying the rows in ASH with impact for the finding.

For most types of findings this identifies the exact rows in ASH corresponding to the finding. For some types of findings the query is an approximation and should not be used for exact identification of the finding's impact or the finding's specific activity.

Syntax

```sql
DBMS_ADDM.GET_ASH_QUERY (  
    task_name           IN   VARCHAR2,  
    finding_id          IN   NUMBER)  
RETURN VARCHAR2;
```

Parameters

Table 14-14  GET_ASH_QUERY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>finding</td>
<td>ID of the finding within the task</td>
</tr>
</tbody>
</table>

Return Values

A VARCHAR containing an ASH query identifying the rows in ASH with impact for the finding

14.2.14 GET_REPORT Function

This function retrieves the default text report of an executed ADDM task.

Syntax

```sql
DBMS_ADDM.GET_REPORT (  
    task_name           IN VARCHAR2)  
RETURN CLOB;
```

Parameters

Table 14-15  GET_REPORT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

Examples

Set long 1000000
Set pagesize 50000
SELECT DBMS_ADDM.GET_REPORT('my_partial_analysis_mode_task') FROM DUAL;
14.2.15 INSERT_FINDING_DIRECTIVE Procedure

This procedure creates a directive to limit reporting of a specific finding type. The directive can be created for a specific task (only when the task is in INITIAL status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

```sql
DBMS_ADDM.INSERT_FINDING_DIRECTIVE (  
    task_name             IN VARCHAR2,  
    dir_name              IN VARCHAR2,  
    finding_name          IN VARCHAR2,  
    min_active_sessions   IN NUMBER := 0,  
    min_perc_impact       IN NUMBER := 0);  
```

Parameters

Table 14-16  INSERT_FINDING_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it applies to all subsequently created ADDM Tasks.</td>
</tr>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
<tr>
<td>finding_name</td>
<td>Name of an ADDM finding to which this directive applies. All valid finding names appear in the NAME column of view DBA_ADVISOR_FINDING_NAMES.</td>
</tr>
<tr>
<td>min_active_sessions</td>
<td>Minimal number of active sessions for the finding. If a finding has less than this number, it is filtered from the ADDM result.</td>
</tr>
<tr>
<td>min_perc_impact</td>
<td>Minimal number for the &quot;percent impact&quot; of the finding relative to total database time in the analysis period. If the finding's impact is less than this number, it is filtered from the ADDM result.</td>
</tr>
</tbody>
</table>

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for 'Undersized SGA' findings. The result of `GET_REPORT` shows only an 'Undersized SGA' finding if the finding is responsible for at least 2 average active sessions during the analysis period, and this constitutes at least 10% of the total database time during that period.

```sql
var tname VARCHAR2(60);
BEGIN
    DBMS_ADDM.INSERT_FINDING_DIRECTIVE(
    NULL,  
    'Undersized SGA directive',  
    'Undersized SGA',  
    2,  
    10);  
    :tname := 'my_instance_analysis_mode_task';  
    DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing 'Undersized SGA' findings regardless of the directive:
14.2.16 INSERT_PARAMETER_DIRECTIVE Procedure

This procedure creates a directive to prevent ADDM from creating actions to alter the value of a specific system parameter. The directive can be created for a specific task (only when the task is in INITIAL status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

```sql
DBMS_ADDM.INSERT_PARAMETER_DIRECTIVE (  
    task_name             IN VARCHAR2,  
    dir_name              IN VARCHAR2,  
    parameter_name        IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it applies to all subsequently created ADDM Tasks.</td>
</tr>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>Specifies the parameter to use. Valid parameter names appear in V$PARAMETER.</td>
</tr>
</tbody>
</table>

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for all actions that recommend modifying the parameter 'sga_target'. The result of GET_REPORT does not show these actions.

```sql
var tname varchar2(60);
BEGIN
    DBMS_ADDM.INSERT_PARAMETER_DIRECTIVE(
        NULL,  
        'my Parameter directive',  
        'sga_target');
    :tname := 'my_instance_analysis_mode_task';
    DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```sql
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

14.2.17 INSERT_SEGMENT_DIRECTIVE Procedure

This procedure creates a directive to prevent ADDM from creating actions to "run Segment Advisor" for specific segments. The directive can be created for a specific task (only when the...
task is in INITIAL status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE (  
  task_name             IN VARCHAR2,  
  dir_name              IN VARCHAR2,  
  owner_name            IN VARCHAR2,  
  object_name           IN VARCHAR2 := NULL,  
  sub_object_name       IN VARCHAR2 := NULL);

DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE (  
  task_name             IN VARCHAR2,  
  dir_name              IN VARCHAR2,  
  object_number         IN NUMBER);

Parameters

Table 14-18  INSERT_SEGMENT_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it applies to all subsequently created ADDM Tasks.</td>
</tr>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
<tr>
<td>owner_name</td>
<td>Specifies the owner of the segment/s to be filtered. A wildcard is allowed in the same syntax used for “like” constraints.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the main object to be filtered. Again, wildcards are allowed. The default value of NULL is equivalent to a value of ’%’.</td>
</tr>
<tr>
<td>sub_object_name</td>
<td>Name of the part of the main object to be filtered. This could be a partition name, or even sub partitions (separated by a ’.’). Again, wildcards are allowed. The default value of NULL is equivalent to a value of ’%’.</td>
</tr>
<tr>
<td>object_number</td>
<td>Object number of the SEGMENT that this directive is to filter, found in views DBA_OBJECTS or DBA_SEGMENTS</td>
</tr>
</tbody>
</table>

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for all segments that belong to user SCOTT. The result of GET_REPORT does not show actions for running Segment advisor for segments that belong to SCOTT.

```sql
var tname VARCHAR2(60);
BEGIN
  DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE(NULL,  
    'my Segment directive',  
    'SCOTT');
  :tname := 'my_instance_analysis_mode_task';  
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;

To see a report containing all actions regardless of the directive:

SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```
14.2.18 INSERT_SQL_DIRECTIVE Procedure

This procedure creates a directive to limit reporting of actions on specific SQL. The directive can be created for a specific task (only when the task is in INITIAL status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

```sql
DBMS_ADDM.INSERT_SQL_DIRECTIVE (
    task_name             IN VARCHAR2,
    dir_name              IN VARCHAR2,
    sql_id                IN VARCHAR2,
    min_active_sessions   IN NUMBER := 0,
    min_response_time     IN NUMBER := 0);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task this directive applies to. If the value is NULL, it applies to all subsequently created ADDM Tasks.</td>
</tr>
<tr>
<td>dir_name</td>
<td>Name of the directive. All directives must be given unique names.</td>
</tr>
<tr>
<td>sql_id</td>
<td>Identifies which SQL statement to filter. A valid value contains exactly 13 characters from '0' to '9' and 'a' to 'z'.</td>
</tr>
<tr>
<td>min_active_sessions</td>
<td>Minimal number of active sessions for the SQL. If a SQL action has less than this number, it is filtered from the ADDM result.</td>
</tr>
<tr>
<td>min_response_time</td>
<td>Minimal value for response time of the SQL (in microseconds). If the SQL had lower response time, it is filtered from the ADDM result.</td>
</tr>
</tbody>
</table>

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for SQL with id 'abcd123456789'. The result of GET_REPORT shows only actions for that SQL (actions to tune the SQL, or to investigate application using it) if the SQL is responsible for at least 2 average active sessions during the analysis period, and the average response time was at least 1 second.

```sql
var tname VARCHAR2(60);
BEGIN
    DBMS_ADDM.INSERT_SQL_DIRECTIVE(
        NULL,
        'my SQL directive',
        'abcd123456789',
        2,
        1000000);
    :tname := 'my_instance_analysis_mode_task';
    DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```sql
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```
14.2.19 REAL_TIME_ADDM_REPORT Function

This function produces a real-time ADDM report for ADDM-related activity for the last five minutes. In an Oracle Real Application Clusters (Oracle RAC) environment, the function assumes that executing SQL over GV$ is possible.

Syntax

DBMS_ADDM.REAL_TIME_ADDM_REPORT ()
RETURN CLOB;

Return Values

CLOB containing a real-time ADDM report
DBMS_ADVANCED_REWRITE contains interfaces for advanced query rewrite users. Using this package, you can create, drop, and maintain functional equivalence declarations for query rewrite.

See Also:
Oracle Database Data Warehousing Guide for more information about query rewrite

This chapter contains the following topics:
- DBMS_ADVANCED_REWRITE Security Model
- Summary of DBMS_ADVANCED_REWRITE Subprograms

15.1 DBMS_ADVANCED_REWRITE Security Model

Default privileges are not granted to anyone for access to DBMS_ADVANCED_REWRITE Security Model procedures. To gain access to these procedures, you must connect as SYSDBA and explicitly grant execute access to the desired database administrators.

You can control security on this package by granting the EXECUTE privilege to selected database administrators or roles. For example, the user er can be given access to use this package by the following statement, executed as SYSDBA:

GRANT EXECUTE ON DBMS_ADVANCED_REWRITE TO er;

You may want to write a separate cover package on top of this package for restricting the alert names used. Instead of granting the EXECUTE privilege on the DBMS_ADVANCED_REWRITE package directly, you can then grant it to the cover package.

In addition, similar to the privilege required for regular materialized views, the user should be granted the privilege to create an equivalence. For example, the user er can be granted this privilege by executing the following statement as SYSDBA:

GRANT CREATE MATERIALIZED VIEW TO er;

15.2 Summary of DBMS_ADVANCED_REWRITE Subprograms

This table lists the DBMS_ADVANCED_REWRITE subprograms and briefly describes them.
Table 15-1  DBMS_ADVANCED_REWRITE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_REWRITE_EQUIVALENCE Procedure</td>
<td>Changes the mode of the rewrite equivalence declaration to the mode you specify</td>
</tr>
<tr>
<td>BUILD_SAFE_REWRITE_EQUIVALENCE Procedure</td>
<td>Enables the rewrite of top-level materialized views using submaterialized views. Oracle Corporation does not recommend you directly use this procedure</td>
</tr>
<tr>
<td>DECLARE_REWRITE_EQUIVALENCE Procedures</td>
<td>Creates a declaration indicating that source_stmt is functionally equivalent to destination_stmt for as long as the equivalence declaration remains enabled, and that destination_stmt is more favorable in terms of performance</td>
</tr>
<tr>
<td>DROP_REWRITE_EQUIVALENCE Procedure</td>
<td>Drops the specified rewrite equivalence declaration</td>
</tr>
<tr>
<td>VALIDATE_REWRITE_EQUIVALENCE Procedure</td>
<td>Validates the specified rewrite equivalence declaration using the same validation method as described with the validate parameter</td>
</tr>
</tbody>
</table>

15.2.1 ALTER_REWRITE_EQUIVALENCE Procedure

This table list the all the package subprograms in alphabetical order.

Syntax

```sql
DBMS_ADVANCED_REWRITE.ALTER_REWRITE_EQUIVALENCE (name VARCHAR2, rewrite_mode VARCHAR2);
```

Parameters

**Table 15-2  ALTER_REWRITE_EQUIVALENCE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A name for the equivalence declaration to alter. The name can be of the form owner.name, where owner complies with the rules for a schema name, and name complies with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is altered in the current schema. The invoker must have the appropriate alter materialized view privileges to alter an equivalence declaration outside their own schema.</td>
</tr>
</tbody>
</table>
Table 15-2   (Cont.) ALTER_REWRITE_EQUIVALENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rewrite_mode</td>
<td>The following modes are supported, in increasing order of power:</td>
</tr>
<tr>
<td></td>
<td>disabled: Query rewrite does not use the equivalence declaration. Use this mode to temporarily disable use of the rewrite equivalence declaration.</td>
</tr>
<tr>
<td></td>
<td>text_match: Query rewrite uses the equivalence declaration only in its text match modes. This mode is useful for simple transformations.</td>
</tr>
<tr>
<td></td>
<td>general: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. However, query</td>
</tr>
<tr>
<td></td>
<td>rewrite makes no attempt to rewrite the specified destination_query.</td>
</tr>
<tr>
<td></td>
<td>recursive: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. Moreover, query</td>
</tr>
<tr>
<td></td>
<td>rewrite further attempts to rewrite the specified destination_query for further performance enhancements whenever it uses the equivalence declaration.</td>
</tr>
<tr>
<td></td>
<td>Oracle recommends you use the least powerful mode that is sufficient to solve your performance problem.</td>
</tr>
</tbody>
</table>

15.2.2 BUILD_SAFE_REWRITE_EQUIVALENCE Procedure

This procedure enables the rewrite and refresh of top-level materialized views using submaterialized views. It is provided for the exclusive use by scripts generated by the DBMS_ADVISOR.TUNE_MVIEW procedure.

It is required to enable query rewrite and fast refresh when DBMS_ADVISOR.TUNE_MVIEW decomposes a materialized view into a top-level materialized view and one or more submaterialized views.

Oracle does not recommend you directly use the BUILD_SAFE_REWRITE_EQUIVALENCE procedure. You should use either the DBMS_ADVISOR.TUNE_MVIEW or the DBMS_ADVANCED_REWRITE.CREATE_REWRITE_EQUIVALENCE procedure as appropriate.

15.2.3 DECLARE_REWRITE_EQUIVALENCE Procedures

This procedure creates a declaration indicating that source_stmt is functionally equivalent to destination_stmt for as long as the equivalence declaration remains enabled, and that destination_stmt is more favorable in terms of performance.

The scope of the declaration is system wide. The query rewrite engine uses such declarations to perform rewrite transformations in QUERY_REWRITE_INTEGRITY = trusted and stale_tolerated modes.

Because the underlying equivalences between the source and destination statements cannot be enforced by the query rewrite engine, queries can be only rewritten in trusted and stale_tolerated integrity modes.

**Syntax**

```sql
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (|
    name VARCHAR2,
```
source_stmt VARCHAR2,
destination_stmt VARCHAR2,
validate BOOLEAN := TRUE,
rewrite_mode VARCHAR2 := 'TEXT_MATCH');

DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
  name VARCHAR2,
  source_stmt CLOB,
  destination_stmt CLOB,
  validate BOOLEAN := TRUE,
  rewrite_mode VARCHAR2 := 'TEXT_MATCH');

Parameters

Table 15-3 DECLARE_REWRITE_EQUIVALENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A name for the equivalence declaration. The name can be of the form owner.name, where owner complies with the rules for a schema name, and name compiles with the rules for a table name. Alternatively, a simple name that compiles with the rules for a table name can be specified. In this case, the rewrite equivalence is created in the current schema. The invoker must have the appropriate CREATE MATERIALIZED VIEW privileges to alter an equivalence declaration.</td>
</tr>
<tr>
<td>source_stmt</td>
<td>A sub-SELECT expression in either VARCHAR2 or CLOB format. This is the query statement that is the target of optimization.</td>
</tr>
<tr>
<td>destination_stmt</td>
<td>A sub-SELECT expression in either VARCHAR2 or CLOB format.</td>
</tr>
<tr>
<td>validate</td>
<td>A Boolean indicating whether to validate that the specified source_stmt is functionally equivalent to the specified destination_stmt. If validate is specified as TRUE, DECLARE_REWRITE_EQUIVALENCE evaluates the two sub-SELECTs and compares their results. If the results are not the same, DECLARE_REWRITE_EQUIVALENCE does not create the rewrite equivalence and returns an error condition. If FALSE, DECLARE_REWRITE_EQUIVALENCE does not validate the equivalence.</td>
</tr>
</tbody>
</table>
Table 15-3  (Cont.) DECLARE_REWRITE_EQUIVALENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rewrite_mode</td>
<td>The following modes are supported, in increasing order of power:</td>
</tr>
<tr>
<td></td>
<td>· disabled: Query rewrite does not use the equivalence declaration. Use this mode to temporarily disable use of the rewrite equivalence declaration.</td>
</tr>
<tr>
<td></td>
<td>· text_match: Query rewrite uses the equivalence declaration only in its text match modes. This mode is useful for simple transformations.</td>
</tr>
<tr>
<td></td>
<td>· general: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. However, query rewrite makes no attempt to rewrite the specified destination_query.</td>
</tr>
<tr>
<td></td>
<td>· recursive: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. Moreover, query rewrite further attempts to rewrite the specified destination_query for further performance enhancements whenever it uses the equivalence declaration.</td>
</tr>
<tr>
<td></td>
<td>Oracle recommends you use the least powerful mode that is sufficient to solve your performance problem.</td>
</tr>
</tbody>
</table>

Exceptions

Table 15-4  DECLARE_REWRITE_EQUIVALENCE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-30388</td>
<td>Name of the rewrite equivalence is not specified</td>
</tr>
<tr>
<td>ORA-30391</td>
<td>The specified rewrite equivalence does not exist</td>
</tr>
<tr>
<td>ORA-30392</td>
<td>The checksum analysis for the rewrite equivalence failed</td>
</tr>
<tr>
<td>ORA-30393</td>
<td>A query block in the statement did not write</td>
</tr>
<tr>
<td>ORA-30396</td>
<td>Rewrite equivalence procedures require the COMPATIBLE parameter to be set to 10.1 or greater</td>
</tr>
</tbody>
</table>

Usage Notes

Query rewrite using equivalence declarations occurs simultaneously and in concert with query rewrite using materialized views. The same query rewrite engine is used for both. The query rewrite engine uses the same rewrite rules to rewrite queries using both equivalence declarations and materialized views. Because the rewrite equivalence represents a specific rewrite crafted by a sophisticated user, the query rewrite engine gives priority to rewrite equivalences over materialized views when it is possible to perform a rewrite with either a materialized view or a rewrite equivalence. For this same reason, the cost-based optimizer (specifically, cost-based rewrite) will not choose an unrewritten query plan over a query plan that is rewritten to use a rewrite equivalence even if the cost of the un-rewritten plan appears more favorable. Query rewrite matches properties of the incoming request query against the equivalence declaration's source_stmt or the materialized view's defining statement, respectively, and derives an equivalent relational expression in terms of the equivalence declaration's destination_stmt or the materialized view's container table, respectively.
15.2.4 DROP_REWRITE_EQUIVALENCE Procedure

This procedure drops the specified rewrite equivalence declaration.

Syntax

DBMS_ADVANCED_REWRITE.DROP_REWRITE_EQUIVALENCE (name VARCHAR2);

Parameters

Table 15-5  DROP_REWRITE_EQUIVALENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A name for the equivalence declaration to drop. The name can be of the form owner.name, where owner complies with the rules for a schema name, and name complies with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is dropped in the current schema. The invoker must have the appropriate drop materialized view privilege to drop an equivalence declaration outside their own schema.</td>
</tr>
</tbody>
</table>

15.2.5 VALIDATE_REWRITE_EQUIVALENCE Procedure

This procedure validates the specified rewrite equivalence declaration.

It uses the same validation method as described with the VALIDATE parameter in "VALIDATE_REWRITE_EQUIVALENCE Procedure".

Syntax

DBMS_ADVANCED_REWRITE.VALIDATE_REWRITE_EQUIVALENCE (name VARCHAR2);

Parameters

Table 15-6  VALIDATE_REWRITE_EQUIVALENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A name for the equivalence declaration to validate. The name can be of the form owner.name, where owner complies with the rules for a schema name, and name complies with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is validated in the current schema. The invoker must have sufficient privileges to execute both the source_stmt and destination_stmt of the specified equivalence declaration.</td>
</tr>
</tbody>
</table>
DBMS_ADVISOR is part of the server manageability suite of advisors, a set of expert systems that identifies and helps resolve performance problems relating to database server components.

Some advisors have their own packages. For these advisors, Oracle recommends that you use the advisor-specific package rather than DBMS_ADVISOR. Each of the following advisors has its own package, tailored to its specific functionality:

- Automatic Database Diagnostic Monitor (DBMS_ADDM)
- SQL Performance Analyzer (DBMS_SQLPA)
- SQL Repair Advisor (DBMS_SQLDIAG)
- SQL Tuning Advisor (DBMS_SQLTUNE)
- Compression Advisor (DBMS_COMPRESSION.GET_COMPRESSION_RATIO)

SQL Access Advisor and Segment Advisor are the only advisors with common use cases for DBMS_ADVISOR. Undo Advisor and Compression Advisor do not support DBMS_ADVISOR subprograms.

This chapter contains the following topics:

- DBMS_ADVISOR Deprecated Subprograms
- DBMS_ADVISOR Security Model
- Summary of DBMS_ADVISOR Subprograms

See Also:

- Oracle Database Administrator’s Guide to learn about Segment Advisor
- Oracle Database 2 Day + Performance Tuning Guide to learn how to use SQL Access Advisor in Enterprise Manager
- Oracle Database SQL Tuning Guide to learn more about SQL Access Advisor

16.1 DBMS_ADVISOR Deprecated Subprograms

The section lists programs that are deprecated with Oracle Database 11g.

Note:

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.
The following subprograms are deprecated:

- ADD_SQLWKLD_REF Procedure
- CREATE_SQLWKLD Procedure
- DELETE_SQLWKLD Procedure
- DELETE_SQLWKLD_REF Procedure
- DELETE_SQLWKLD_STATEMENT Procedure
- IMPORT_SQLWKLD_SCHEMA Procedure
- IMPORT_SQLWKLD_SQLCACHE Procedure
- IMPORT_SQLWKLD_STS Procedure
- IMPORT_SQLWKLD_SUMADV Procedure
- IMPORT_SQLWKLD_USER Procedure
- RESET_SQLWKLD Procedure
- SET_SQLWKLD_PARAMETER Procedure
- UPDATE_SQLWKLD_ATTRIBUTES Procedure
- UPDATE_SQLWKLD_STATEMENT Procedure

### 16.2 DBMS_ADVISOR Security Model

The **ADVISOR** privilege is required to use the DBMS_ADVISOR package.

### 16.3 Summary of DBMS_ADVISOR Subprograms

This topic lists and describes the subprograms in the DBMS_ADVISOR package.

In the following table, the **Used in** column lists advisors relevant for each subprogram, but excludes ADDM, SQL Performance Analyzer, SQL Repair Advisor, and SQL Tuning Advisor because these advisors have their own packages.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Used in</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_SQLWKLD_REF Procedure</td>
<td>Adds a workload reference to an Advisor task (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>ADD_SQLWKLD_STATEMENT Procedure</td>
<td>Adds a single statement to a workload</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>ADD_STS_REF Procedure</td>
<td>Establishes a link between the current SQL Access Advisor task and a SQL tuning set</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>CANCEL_TASK Procedure</td>
<td>Cancels a currently executing task operation</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>COPY_SQLWKLD_TO_STS Procedure</td>
<td>Copies the contents of a SQL workload object to a SQL tuning set</td>
<td>SQL Access Advisor</td>
</tr>
</tbody>
</table>
## Table 16-1  (Cont.) DBMS_ADVISOR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Used in</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_FILE Procedure</td>
<td>Creates an external file from a PL/SQL CLOB variable, which is useful for creating scripts and reports</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>CREATE_OBJEKT Procedure</td>
<td>Creates a new task object</td>
<td>Segment Advisor</td>
</tr>
<tr>
<td>CREATE_SQLWK KLD Procedure</td>
<td>Creates a new workload object (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>CREATE_TASK Procedures</td>
<td>Creates a new Advisor task in the repository</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>DELETE_SQLWK KLD Procedure</td>
<td>Deletes an entire workload object (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>DELETE_SQLWK KLD_REF Procedure</td>
<td>Deletes an entire workload object (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>DELETE_SQLWK KLD_STATEMENT Procedure</td>
<td>Deletes one or more statements from a workload (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>DELETE_STS_REF Procedure</td>
<td>Removes a link between the current SQL Access Advisor task and a SQL tuning set object</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>DELETE_TASK Procedure</td>
<td>Deletes the specified task from the repository</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>EXECUTE_TASK Procedure</td>
<td>Executes the specified task</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>GET_REC_ATTRIBUTES Procedure</td>
<td>Retrieves specific recommendation attributes from a task</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>GET_TASK_REPORT Function</td>
<td>Creates and returns a report for the specified task</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>GET_TASK_SCR IPT Function</td>
<td>Creates and returns an executable SQL script of the Advisor task’s recommendations in a buffer</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>IMPLEMENT_TASK Procedure</td>
<td>Implements the recommendations for a task</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>IMPORT_SQLWK LD_SCHEMA Procedure</td>
<td>Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>IMPORT_SQLWK LD_SQLCACHE Procedure</td>
<td>Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>IMPORT_SQLWK LD_STS Procedure</td>
<td>Imports data from a SQL tuning set into a SQL workload data object (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>IMPORT_SQLWK LD_SUMADV Procedure</td>
<td>Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>IMPORT_SQLWK LD_USER Procedure</td>
<td>Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
</tbody>
</table>
Table 16-1  (Cont.) DBMS_ADVISOR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Used in</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT_TASK Procedure</td>
<td>Stops a currently executing task, ending its operations as it would at a normal exit, so that the recommendations are visible</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>MARK_RECOMENDATION Procedure</td>
<td>Sets the annotation_status for a particular recommendation</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>QUICK_TUNE Procedure</td>
<td>Performs an analysis on a single SQL statement</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>RESET_SQLWKLD Procedure</td>
<td>Resets a workload to its initial starting point (Caution: Deprecated Subprogram)</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>RESET_TASK Procedure</td>
<td>Resets a task to its initial state</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>SET_DEFAULT_SQLWKLD_PAR AMETER Procedure</td>
<td>Imports data into a workload from schema evidence</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>SET_DEFAULT_TASK_PARAM ETER Procedure</td>
<td>Modifies a default task parameter</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>SET_SQLWKLD_PARAMETER Procedure</td>
<td>Sets the value of a workload parameter</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>SET_TASK_PARAMETER Procedure</td>
<td>Sets the specified task parameter value</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
<tr>
<td>TUNE_MVIEW Procedure</td>
<td>Shows how to decompose a materialized view into two or more materialized views or to restate the materialized view in a way that is more advantageous for fast refresh and query rewrite</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>UPDATE_OBJECT Procedure</td>
<td>Updates a task object</td>
<td>Segment Advisor</td>
</tr>
<tr>
<td>UPDATE_REC_ATTRIBUTES Procedure</td>
<td>Updates an existing recommendation for the specified task</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>UPDATE_SQLWKLD_ATTRIBU TES Procedure</td>
<td>Updates a workload object</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>UPDATE_SQLWKLD_STATEMENT Procedure</td>
<td>Updates one or more SQL statements in a workload</td>
<td>SQL Access Advisor</td>
</tr>
<tr>
<td>UPDATE_TASK_ATTRIBUTES Procedure</td>
<td>Updates a task's attributes</td>
<td>Segment Advisor, SQL Access Advisor</td>
</tr>
</tbody>
</table>
16.3.1 ADD_SQLWKLD_REF Procedure

This procedure establishes a link between the current SQL Access Advisor task and a SQL Workload object.

Note:
This procedure is deprecated starting in Oracle Database 11g.

The link allows an advisor task to access interesting data for doing an analysis. The link also provides a stable view of the data. Once a connection between a SQL Access Advisor task and a SQL Workload object is made, the workload is protected from removal or modification.

Users should use **ADD_STS_REF** instead of **ADD_SQLWKLD_REF** for all SQL tuning set-based advisor runs. This function is only provided for backward compatibility.

Syntax

```sql
DBMS_ADVISOR.ADD_SQLWKLD_REF (
    task_name              IN VARCHAR2,
    workload_name          IN VARCHAR2,
    is_sts                 IN NUMBER := 0);
```

Parameters

Table 16-2  ADD_SQLWKLD_REF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The SQL Access Advisor task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>workload_name</td>
<td>The name of the workload object to be linked. Once a object has been linked to a task, it becomes read-only and cannot be deleted. There is no limit to the number of links to workload objects. To remove the link to the workload object, use the procedure <strong>DELETE_REFERENCE</strong>.</td>
</tr>
<tr>
<td>is_sts</td>
<td>Indicates the type of workload source. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• 0 - SQL workload object</td>
</tr>
<tr>
<td></td>
<td>• 1 - SQL tuning set</td>
</tr>
</tbody>
</table>

Examples

```sql
DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
```
16.3.2 ADD_SQLWKLD_STATEMENT Procedure

This procedure adds a single statement to the specified workload.

Note:
This procedure is deprecated starting in Oracle Database 11g.

Syntax

```
DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT (  
  workload_name          IN VARCHAR2,  
  module                 IN VARCHAR2,  
  action                 IN VARCHAR2,  
  cpu_time               IN NUMBER := 0,  
  elapsed_time           IN NUMBER := 0,  
  disk_reads             IN NUMBER := 0,  
  buffer_gets            IN NUMBER := 0,  
  rows_processed         IN NUMBER := 0,  
  optimizer_cost         IN NUMBER := 0,  
  executions             IN NUMBER := 1,  
  priority               IN NUMBER := 2,  
  last_execution_date    IN DATE := 'SYSDATE',  
  stat_period            IN NUMBER := 0,  
  username               IN VARCHAR2,  
  sql_text               IN CLOB);  
```

Parameters

Table 16-3    ADD_SQLWKLD_STATEMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>module</td>
<td>An optional business application module that will be associated with the SQL statement.</td>
</tr>
<tr>
<td>action</td>
<td>An optional application action that will be associated with the SQL statement.</td>
</tr>
<tr>
<td>cpu_time</td>
<td>The total CPU time in seconds that is consumed by the SQL statement.</td>
</tr>
<tr>
<td>elapsed_time</td>
<td>The total elapsed time in seconds that is consumed by the SQL statement.</td>
</tr>
<tr>
<td>disk_reads</td>
<td>The total disk-read operations that are consumed by the SQL statement.</td>
</tr>
<tr>
<td>buffer_gets</td>
<td></td>
</tr>
<tr>
<td>rows_processed</td>
<td></td>
</tr>
<tr>
<td>optimizer_cost</td>
<td></td>
</tr>
<tr>
<td>executions</td>
<td></td>
</tr>
<tr>
<td>priority</td>
<td></td>
</tr>
<tr>
<td>last_execution_date</td>
<td>The date of the last execution of the SQL statement.</td>
</tr>
<tr>
<td>stat_period</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td></td>
</tr>
<tr>
<td>sql_text</td>
<td></td>
</tr>
</tbody>
</table>
Table 16-3  (Cont.) ADD_SQLWKLD_STATEMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_gets</td>
<td>The total buffer-get operations that are consumed by the SQL statement.</td>
</tr>
<tr>
<td>rows_processed</td>
<td>The average number of rows processed by the SQL statement.</td>
</tr>
<tr>
<td>optimizer_cost</td>
<td>The optimizer's calculated cost value.</td>
</tr>
<tr>
<td>executions</td>
<td>The total execution count by the SQL statement. This value should be greater than zero.</td>
</tr>
<tr>
<td>priority</td>
<td>The relative priority of the SQL statement. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.</td>
</tr>
<tr>
<td>last_execution_date</td>
<td>The date and time at which the SQL statement last executed. If the value is NULL, then the current date and time will be used.</td>
</tr>
<tr>
<td>stat_period</td>
<td>Time interval in seconds from which statement statistics were calculated.</td>
</tr>
<tr>
<td>username</td>
<td>The Oracle user name that executed the SQL statement. Because a user name is an Oracle identifier, the username value must be entered exactly as it is stored in the server. For example, if the user SCOTT is the executing user, then you must provide the user identifier SCOTT in all uppercase letters. It will not recognize the user scott as a match for SCOTT.</td>
</tr>
<tr>
<td>sql_text</td>
<td>The complete SQL statement. To increase the quality of a recommendation, the SQL statement should not contain bind variables.</td>
</tr>
</tbody>
</table>

Usage Notes

You cannot modify or delete a workload when it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure for directions on setting a task to its initial state.

The ADD_SQLWKLD_STATEMENT procedure accepts several parameters that may be ignored by the caller. The database only uses the disk_reads, buffer_gets, and optimizer_cost parameters to sort workload data when actual analysis occurs. Therefore, actual values are only necessary when the order_list task parameter references a particular statistic.

To determine what statistics to provide when adding a new SQL statement to a workload, examine or set the task parameter order_list. The order_list parameter accepts any combination of the keys:

- cpu_time
- elapsed_time
- buffer_gets
- optimizer_cost
- disk_reads
- executions
- priority

The optimizer_cost key, which is a typical setting of priority, indicates that SQL Access Advisor sorts the workload data by priority and optimizer_cost, and processes the highest
cost statements first. Any statements that you add to the workload must include appropriate priority and optimizer_cost values. All other statistics can be defaulted or set to zero.

For the statistical keys referenced by the order_list task parameter, the actual parameter values should be reasonably accurate since they will be compared to other statements in the workload. If the caller is unable to estimate values, then choose values that would determine its importance relative to other statements in the workload. For example, if the current statement is considered the most critical query in your business, then an appropriate value would be anything greater than all other values for the same statistic found in the workload.

Examples

DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY',
    'ROLLUP',
    100, 400, 5041, 103, 640445, 680000, 2,
    1, SYSDATE, 1, 'SH', 'SELECT AVG(amount_sold) FROM sh.sales');
END;
/

16.3.3 ADD_STS_REF Procedure

This procedure establishes a link between the current SQL Access Advisor task and a SQL tuning set.

The link enables an advisor task to access data for the purpose of doing an analysis. The link also provides a stable view of the data. Once a connection between a SQL Access Advisor task and a SQL tuning set is made, the STS is protected from removal or modification.

Use ADD_STS_REF for any STS-based advisor runs. The older method of using ADD_SQLWKLD_REF with parameter IS_STS=1 is only supported for backward compatibility. Furthermore, the ADD_STS_REF function accepts a SQL tuning set owner name, whereas ADD_SQLWKLD_REF does not.

Syntax

DBMS_ADVISOR.ADD_STS_REF(
  task_name       IN VARCHAR2 NOT NULL,
  sts_owner       IN VARCHAR2,
  workload_name   IN VARCHAR2 NOT NULL);
### Parameters

**Table 16-4  ADD_STS_REF Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The SQL Access Advisor task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>sts_owner</td>
<td>The owner of the SQL tuning set. The value of this parameter may be NULL, in which case the advisor assumes the SQL tuning set to be owned by the currently logged-in user.</td>
</tr>
<tr>
<td>workload_name</td>
<td>The name of the workload to be linked. A workload consists of one or more SQL statements, plus statistics and attributes that fully describe each statement. The database stores a workload as a SQL tuning set. After a workload has been linked to a task, it becomes read-only and cannot be deleted. There is no limit to the number of links to workloads. To remove the link to the workload, use the procedure DBMS_ADVISOR.DELETE_STS_REF.</td>
</tr>
</tbody>
</table>

**Examples**

```sql
DBMS_ADVISOR.ADD_STS_REF ('My Task', 'SCOTT', 'My Workload');
```

### 16.3.4 CANCEL_TASK Procedure

This procedure causes a currently executing operation to terminate.

This call performs a soft interrupt. It will not break into a low-level database access call like a hard interrupt such as Ctrl-C. The SQL Access Advisor periodically checks for soft interrupts and acts appropriately. As a result, this operation may take a few seconds to respond to a call.

**Syntax**

```sql
DBMS_ADVISOR.CANCEL_TASK (  
    task_name IN  VARCHAR2);  
```

**Parameters**

**Table 16-5  CANCEL_TASK Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>A valid Advisor task name that uniquely identifies an existing task.</td>
</tr>
</tbody>
</table>

**Usage Notes**

A cancel command restores the task to its condition prior to the start of the cancelled operation. Therefore, a cancelled task or data object cannot be resumed.

Because all Advisor task procedures are synchronous, to cancel an operation, you must use a separate database session.
Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.CANCEL_TASK('My Task');
END;
/

16.3.5 COPY_SQLWKLD_TO_STS Procedure

This procedure copies the contents of a SQL workload object to a SQL tuning set.

Syntax

To use this procedure, the caller must have privileges to create and modify a SQL tuning set.

\[
\text{DBMS_ADVISOR.COPY_SQLWKLD_TO_STS (}
\text{    workload_name IN VARCHAR2,}
\text{    sts_name IN VARCHAR2,}
\text{    import_mode IN VARCHAR2 := 'NEW');}
\]

Parameters

Table 16-6  COPY_SQLWKLD_TO_STS Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The SQL Workload object name to copy.</td>
</tr>
<tr>
<td>sts_name</td>
<td>The SQL tuning set name into which the SQL Workload object will be copied.</td>
</tr>
</tbody>
</table>
Usage Notes

To use this procedure, the caller must have privileges to create and modify a SQL tuning set.

Examples

BEGIN
  DBMS_ADVISOR.COPY_SQLWKLD_TO_STS('MY_OLD_WORKLOAD', 'MY_NEW_STS', 'NEW');
END;
/

16.3.6 CREATE_FILE Procedure

This procedure creates an external file from a PL/SQL CLOB variable, which is used for creating scripts and reports.

Syntax

DBMS_ADVISOR.CREATE_FILE (
  buffer       IN  CLOB,
  location     IN  VARCHAR2,
  filename     IN  VARCHAR2);

Parameters

Table 16-7  CREATE_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>A CLOB buffer containing report or script information.</td>
</tr>
<tr>
<td>location</td>
<td>The name of the directory that will contain the output file. You must use the alias as defined by the CREATE_DIRECTORY statement. The Advisor translates the alias into the actual directory location.</td>
</tr>
</tbody>
</table>
Table 16-7  (Cont.) CREATE_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the output file. The file name can only contain the name and an optional file type of the form filename.filetype.</td>
</tr>
</tbody>
</table>

Usage Notes

You must embed all formatting within the CLOB.

The database restricts file access within stored procedures. This means that file locations and names must adhere to the known file permissions in the server.

Examples

CREATE DIRECTORY MY_DIR as '/homedir/user4/gssmith';
GRANT READ,WRITE ON DIRECTORY MY_DIR TO PUBLIC;

DECLARE
  v_task_id NUMBER;
  v_task_name VARCHAR2(30);
  v_workload_name VARCHAR2(30);
BEGIN
  v_task_name := 'My Task';
  v_workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(
    advisor_name => DBMS_ADVISOR.SQLACCESS_ADVISOR,
    task_id      => v_task_id,
    task_name    => v_task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(
    workload_name => v_workload_name,
    description   => 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(
    task_name     => v_task_name,
    workload_name => v_workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(
    workload_name       => v_workload_name,
    module              => 'MONTHLY',
    action              => 'ROLLUP',
    cpu_time            => 100,
    elapsed_time        => 400,
    disk_reads          => 5041,
    buffer_gets         => 103,
    rows_processed      => 640445,
    optimizer_cost      => 680000,
    executions          => 2,
    priority            => 1,
    last_execution_date => SYSDATE,
    stat_period         => 1,
    username            => 'SH',
    sql_text            => 'SELECT AVG(amount_sold) FROM sh.sales');
DBMS_ADVISOR.EXECUTE_TASK(v_task_name);
DBMS_ADVISOR.CREATE_FILE(
    buffer => DBMS_ADVISOR.GET_TASK_SCRIPT(v_task_name)
    , location => 'MY_DIR'
    , filename => 'script.sql' )
END;
/

16.3.7 CREATE_OBJECT Procedure

This procedure creates a new task object.

Syntax

DBMS_ADVISOR.CREATE_OBJECT (    task_name IN VARCHAR2,
    object_type IN VARCHAR2,
    attr1 IN VARCHAR2 := NULL,
    attr2 IN VARCHAR2 := NULL,
    attr3 IN VARCHAR2 := NULL,
    attr4 IN CLOB := NULL,
    attr5 IN VARCHAR2 := NULL,
    object_id OUT NUMBER,
    attr6 IN VARCHAR2 := NULL,
    attr7 IN VARCHAR2 := NULL,
    attr8 IN VARCHAR2 := NULL,
    attr9 IN VARCHAR2 := NULL,
    attr10 IN VARCHAR2 := NULL);

Parameters

Table 16-8 CREATE_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>A valid Advisor task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>object_type</td>
<td>Specifies the external object type.</td>
</tr>
<tr>
<td>attr1</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr2</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr3</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr4</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr5</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>object_id</td>
<td>The advisor-assigned object identifier.</td>
</tr>
<tr>
<td>attr6</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr7</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr8</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr9</td>
<td>Advisor-specific data.</td>
</tr>
<tr>
<td>attr10</td>
<td>Advisor-specific data.</td>
</tr>
</tbody>
</table>

The attribute parameters have different values depending upon the object type. See Oracle Database Administrator's Guide for details regarding these parameters and object types.
Return Values

Returns the new object identifier.

Usage Notes

Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level. If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or LOB column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

See Oracle Database Administrator’s Guide for further information regarding the Segment Advisor.

Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    obj_id NUMBER;
BEGIN
    task_name := 'My Task';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.CREATE_OBJECT (task_name,'SQL',NULL,NULL,NULL,
        'SELECT * FROM SH.SALES',obj_id);
END;
/

16.3.8 CREATE_SQLWKLD Procedure

This procedure creates a new private SQL Workload object for the user.

A SQL Workload object manages a SQL workload on behalf of the SQL Access Advisor. A SQL Workload object must exist prior to performing any other SQL Workload operations, such as importing or updating SQL statements.

Note:

This procedure is deprecated starting in Oracle Database 11g.

Syntax

DBMS_ADVISOR.CREATE_SQLWKLD ( {     workload_name IN OUT VARCHAR2,    description IN VARCHAR2 := NULL,    template IN VARCHAR2 := NULL,    is_template IN VARCHAR2 := 'FALSE');
Parameters

### Table 16-9 CREATE_SQLWKLD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>A name that uniquely identifies the created workload. If not specified, the system will generate a unique name. Names can be up to 30 characters long.</td>
</tr>
<tr>
<td>description</td>
<td>Specifies an optional workload description. Descriptions can be up to 256 characters.</td>
</tr>
<tr>
<td>template</td>
<td>An optional SQL Workload name of an existing workload data object or data object template.</td>
</tr>
<tr>
<td>is_template</td>
<td>An optional value that enables you to set the newly created workload as a template. Valid values are TRUE and FALSE.</td>
</tr>
</tbody>
</table>

Return Values

The SQL Access Advisor returns a unique workload object identifier number that must be used for subsequent activities within the new SQL Workload object.

Usage Notes

By default, workload objects are created using built-in default settings. To create a workload using the parameter settings of an existing workload or workload template, the user may specify an existing workload name.

After a SQL Workload object is present, it can then be referenced by one or more SQL Access Advisor tasks using the `ADD_SQLWKLD_REF` procedure.

Examples

```sql
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
END;
/
```

### 16.3.9 CREATE_TASK Procedures

This procedure creates a new Advisor task in the repository.

**Syntax**

```sql
DBMS_ADVISOR.CREATE_TASK (advisor_name IN VARCHAR2,
                           task_id OUT NUMBER,
                           task_name IN OUT VARCHAR2,
                           task_desc IN VARCHAR2 := NULL,
                           template IN VARCHAR2 := NULL,
                           is_template IN VARCHAR2 := 'FALSE',
```
DBMS_ADVISOR.CREATE_TASK (
    advisor_name IN VARCHAR2,
    task_name IN VARCHAR2,
    task_desc IN VARCHAR2 := NULL,
    template IN VARCHAR2 := NULL,
    is_template IN VARCHAR2 := 'FALSE',
    how_created IN VARCHAR2 := NULL);

DBMS_ADVISOR.CREATE_TASK (
    parent_task_name IN VARCHAR2,
    rec_id IN NUMBER,
    task_id OUT NUMBER,
    task_name IN OUT VARCHAR2,
    task_desc IN VARCHAR2,
    template IN VARCHAR2);

Parameters

Table 16-10 CREATE_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>advisor_name</td>
<td>Specifies the unique advisor name as defined in the view DBA_ADVISOR_DEFINITIONS.</td>
</tr>
<tr>
<td>task_id</td>
<td>A number that uniquely identifies the created task. The number is generated by the procedure and returned to the user.</td>
</tr>
<tr>
<td>task_name</td>
<td>Specifies a new task name. Names must be unique among all tasks for the user.</td>
</tr>
<tr>
<td></td>
<td>When using the second form of the CREATE_TASK syntax listed above (with OUT), a unique name can be generated. Names can be up to 30 characters long.</td>
</tr>
<tr>
<td>task_desc</td>
<td>Specifies an optional task description. Descriptions can be up to 256 characters in length.</td>
</tr>
<tr>
<td>template</td>
<td>An optional task name of an existing task or task template. To specify built-in SQL Access Advisor templates, use the template name as described earlier.</td>
</tr>
<tr>
<td>is_template</td>
<td>An optional value that allows the user to set the newly created task as template. Valid values are: TRUE and FALSE.</td>
</tr>
<tr>
<td>how_created</td>
<td>An optional value that identifies how the source was created.</td>
</tr>
</tbody>
</table>

Return Values

Returns a unique task ID number and a unique task name if one is not specified.

Usage Notes

A task must be associated with an advisor, and once the task has been created, it is permanently associated with the original advisor. By default, tasks are created using built-in default settings. To create a task using the parameter settings of an existing task or task template, the user may specify an existing task name.

For the SQL Access Advisor, use the identifier DBMS_ADVISOR.SQLACCESS_ADVISOR as the advisor_name.
The SQL Access Advisor provides three built-in task templates, using the following constants:

- **DBMS_ADVISOR.SQLACCESS_OLTP**
  Parameters are preset to favor an OLTP application environment.

- **DBMS_ADVISOR.SQLACCESS_WAREHOUSE**
  Parameters are preset to favor a data warehouse application environment.

- **DBMS_ADVISOR.SQLACCESS_GENERAL**
  Parameters are preset to favor a hybrid application environment where both OLTP and data warehouse operations may occur. For the SQL Access Advisor, this is the default template.

**Examples**

```sql
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
END;
/
```

### 16.3.10 DELETE_SQLWKLD Procedure

This procedure deletes an existing SQL Workload object from the repository.

**Note:**

This procedure is deprecated starting in Oracle Database 11g.

**Syntax**

```sql
DBMS_ADVISOR.DELETE_SQLWKLD (workload_name IN VARCHAR2);
```

**Parameters**

**Table 16-11  DELETE_SQLWKLD Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload object name that uniquely identifies an existing workload. The wildcard <code>%</code> is supported as a WORKLOAD_NAME. The rules of use are identical to the LIKE operator. For example, to delete all tasks for the current user, use the wildcard <code>%</code> as the WORKLOAD_NAME. If a wildcard is provided, the DELETE_SQLWKLD operation will not delete any workloads marked as READ_ONLY or TEMPLATE.</td>
</tr>
</tbody>
</table>
Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the \texttt{RESET\_TASK Procedure} to set a task to its initial state.

Examples

\begin{verbatim}
DECLARE
    workload_name VARCHAR2(30);
BEGIN
    workload_name := 'My Workload';
    DBMS_ADVISOR.CREATE\_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.DELETE\_SQLWKLD(workload_name);
END;
/
\end{verbatim}

16.3.11 \texttt{DELETE\_SQLWKLD\_REF Procedure}

This procedure removes a link between the current SQL Access task and a SQL Workload data object.

\begin{quote}
\textbf{Note:}

This procedure is deprecated starting in Oracle Database 11g.
\end{quote}

Use \texttt{DELETE\_STS\_REF} instead of \texttt{DELETE\_SQLWKLD\_REF} for all SQL tuning set-based advisor runs. This function is only provided for backward compatibility.

Syntax

\begin{verbatim}
DBMS\_ADVISOR\_DELETE\_SQLWKLD\_REF (
    task_name              IN VARCHAR2,
    workload_name          IN VARCHAR2,
    is_sts                 IN NUMBER :=0);
\end{verbatim}

Parameters

\textbf{Table 16-12} \texttt{DELETE\_SQLWKLD\_REF Procedure Parameters}

\begin{tabular}{|l|p{10cm}|}
\hline
\textbf{Parameter} & \textbf{Description} \\
\hline
\texttt{task\_name} & The SQL Access task name that uniquely identifies an existing task. \\
\texttt{workload\_name} & The name of the workload object to be unlinked. The wildcard \% is supported as a \texttt{workload\_name}. The rules of use are identical to the \texttt{LIKE} operator. For example, to remove all links to workload objects, use the wildcard \% as the \texttt{workload\_name}. \\
\texttt{is\_sts} & Indicates the type of workload source. Possible values are: \\
& \begin{itemize}
\item 0 - SQL workload object \\
\item 1 - SQL tuning set 
\end{itemize} \\
\hline
\end{tabular}
Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
    DBMS_ADVISOR.DELETE_SQLWKLD_REF(task_name, workload_name);
END;
/

16.3.12 DELETE_SQLWKLD_STATEMENT Procedure

This procedure deletes one or more statements from a workload.

Note:
This procedure has been deprecated.

Syntax

DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (  
    workload_name        IN VARCHAR2,  
    sql_id               IN NUMBER);  

DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (  
    workload_name        IN VARCHAR2,  
    search               IN VARCHAR2,  
    deleted              OUT NUMBER);  

Parameters

Table 16-13   DELETE_SQLWKLD_STATEMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>sql_id</td>
<td>The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant ADVISOR_ALL.</td>
</tr>
<tr>
<td>search</td>
<td>Disabled.</td>
</tr>
<tr>
<td>deleted</td>
<td>Returns the number of statements deleted by the searched deleted operation.</td>
</tr>
</tbody>
</table>

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the RESET_TASK Procedure to set a task to its initial state.
Examples

DECLARE
  workload_name VARCHAR2(30);
  deleted NUMBER;
  id NUMBER;
BEGIN
  workload_name := 'My Workload';
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'YEARLY', 'ROLLUP',
                                       100, 400, 5041, 103, 640445, 680000, 2,
                                       1, SYSDATE, 1, 'SH', 'SELECT AVG(amount_sold)'
                                       FROM sh.sales);

  SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
       WHERE workload_name = 'My Workload';

  DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT(workload_name, id);
END;
/

16.3.13 DELETE_STS_REF Procedure

This procedure removes a link between the current SQL Access Advisor task and a SQL tuning set.

Use DELETE_STS_REF for any STS-based advisor runs. The older method of using DELETE_SQLWKLD_REF with parameter IS_STS=1 is only supported for backward compatibility. Furthermore, the DELETE_STS_REF function accepts an STS owner name, whereas DELETE_SQLWKLD_REF does not.

Syntax

DBMS_ADVISOR.DELETE_STS_REF (  
  task_name       IN VARCHAR2 NOT NULL,
  sts_owner       IN VARCHAR2,
  workload_name   IN VARCHAR2 NOT NULL);

Parameters

Table 16-14  DELETE_STS_REF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The SQL Access Advisor task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>sts_owner</td>
<td>The owner of the SQL tuning set. The value of this parameter may be NULL, in which case the advisor assumes the SQL tuning set to be owned by the currently logged-in user.</td>
</tr>
</tbody>
</table>
Table 16-14  (Cont.) DELETE_STS_REF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| workload_name   | The name of the workload to be unlinked. A workload consists of one or more SQL statements, plus statistics and attributes that fully describe each statement. The database stores a workload as a SQL tuning set.  
The wildcard `%` is supported as a workload name. The rules of use are identical to the SQL LIKE operator. For example, to remove all links to SQL tuning set objects, use the wildcard `%` as the STS_NAME. |

Examples

```
DBMS_ADVISOR.DELETE_STS_REF ('My task', 'SCOTT', 'My workload');
```

16.3.14 DELETE_TASK Procedure

This procedure deletes an existing task from the repository.

Syntax

```
DBMS_ADVISOR.DELETE_TASK (  
    task_name    IN VARCHAR2);
```

Parameters

Table 16-15  DELETE_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| task_name | A single Advisor task name that will be deleted from the repository. The wildcard `%` is supported as a TASK_NAME. The rules of use are identical to the LIKE operator. For example, to delete all tasks for the current user, use the wildcard `%` as the TASK_NAME.  
If a wildcard is provided, the DELETE_TASK operation will not delete any tasks marked as READ_ONLY or TEMPLATE. |

Examples

```
DECLARE  
    task_id NUMBER;  
    task_name VARCHAR2(30);  
BEGIN  
    task_name := 'My Task';  
    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);  
    DBMS_ADVISOR.DELETE_TASK(task_name);  
END;  
/`
```
16.3.15 EXECUTE_TASK Procedure

This procedure performs the Advisor analysis or evaluation for the specified task. The procedure is overloaded.

The execution-related arguments are optional and you do not need to set them for advisors that do not allow their tasks to be executed multiple times.

Advisors can execute a task multiple times and use the results for further processing and analysis.

Syntax

DBMS_ADVISOR.EXECUTE_TASK (  
  task_name          IN VARCHAR2);  

DBMS_ADVISOR.EXECUTE_TASK (  
  task_name         IN VARCHAR2,  
  execution_type    IN VARCHAR2             := NULL,  
  execution_name    IN VARCHAR2             := NULL,  
  execution_params  IN dbms_advisor.argList := NULL,  
  execution_desc    IN VARCHAR2             := NULL,  
  RETURN VARCHAR2;  

Parameters

Table 16-16  EXECUTE_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The task name that uniquely identifies an existing task.</td>
</tr>
</tbody>
</table>
| execution_type| The type of action to be performed by the function. If NULL, it will default to the value of the DEFAULT_EXECUTION_TYPE parameter. As an example, the SQL Performance Analyzer accepts the following possible values:  
  - EXPLAIN PLAN: Generate an explain plan for a SQL statement. This is similar to an EXPLAIN PLAN command. The resulting plans will be stored in the advisor framework in association with the task.  
  - TEST EXECUTE: Test execute the SQL statement and collect its execute plan and statistics. The resulting plans and statistics are stored in the advisor framework.  
  - ANALYZE PERFORMANCE: Analyze and compare two versions of SQL performance data. The performance data is generated by test executing a SQL statement or generating its explain plan. |
| execution_name| A name to qualify and identify an execution. If not specified, it will be generated by the Advisor and returned by function. |
| execution_params| A list of parameters (name, value) for the specified execution. Note that execution parameters are real task parameters, but they affect only the execution they are specified for. As an example, consider the following:  
  DBMS_ADVISOR.ARGLIST('time_limit', 12, 'username', 'hr') |
Table 16-16  (Cont.) EXECUTE_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execution_desc</td>
<td>A 256-length string describing the execution.</td>
</tr>
</tbody>
</table>

**Usage Notes**

Task execution is a synchronous operation. Control will not be returned to the caller until the operation has completed, or a user-interrupt was detected.

Upon return, you can check the DBA_ADVISOR_LOG table for the execution status.

**Examples**

```sql
DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
    DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/```

### 16.3.16 GET_REC_ATTRIBUTES Procedure

This procedure retrieves a specified attribute of a new object as recommended by Advisor analysis.

**Syntax**

```sql
DBMS_ADVISOR.GET_REC_ATTRIBUTES (  
    workload_name IN VARCHAR2,  
    rec_id IN NUMBER,  
    action_id IN NUMBER,  
    attribute_name IN VARCHAR2,  
    value OUT VARCHAR2,  
    owner_name IN VARCHAR2 := NULL);  
```

**Parameters**

Table 16-17  GET_REC_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>rec_id</td>
<td>The Advisor-generated identifier number that is assigned to the recommendation.</td>
</tr>
</tbody>
</table>
Table 16-17  (Cont.) GET_REC_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action_id</td>
<td>The Advisor-generated action identifier that is assigned to the particular command.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Specifies the attribute to change.</td>
</tr>
<tr>
<td>value</td>
<td>The buffer to receive the requested attribute value.</td>
</tr>
<tr>
<td>owner_name</td>
<td>Optional owner name of the target task. This permits access to task data not owned by the current user.</td>
</tr>
</tbody>
</table>

Return Values

The requested attribute value is returned in the `VALUE` argument.

Examples

```
DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
    attribute VARCHAR2(100);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';
    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                        100,400,5041,103,640445,680000,2,
                                        1,SYSDATE,1,'SH','SELECT
                                        AVG(amount_sold)
                                        FROM sh.sales WHERE promo_id =
                                        10');
    DBMS_ADVISOR.EXECUTE_TASK(task_name);
    DBMS_ADVISOR.GET_REC_ATTRIBUTES(task_name, 1, 1, 'NAME', attribute);
END;
/
```

16.3.17 GET_TASK_REPORT Function

This function creates and returns a report for the specified task.

Syntax

```
DBMS_ADVISOR.GET_TASK_REPORT (  
    task_name  IN VARCHAR2,         
    type       IN VARCHAR2 := 'TEXT',  
    level      IN VARCHAR2 := 'TYPICAL',  
    section    IN VARCHAR2 := 'ALL',  
```
The SQL script generated by the GET_TASK_SCRIPT function contains all of the accepted recommendations from the specified task.

Syntax

```sql
DBMS_ADVISOR.GET_TASK_SCRIPT (    task_name IN VARCHAR2,    type IN VARCHAR2 := 'IMPLEMENTATION',    rec_id IN NUMBER := NULL,    act_id IN NUMBER := NULL,    owner_name IN VARCHAR2 := NULL,    execution_name IN VARCHAR2 := NULL,    object_id IN NUMBER := NULL)    RETURN CLOB;
```

16.3.18 GET_TASK_SCRIPT Function

Returns the buffer receiving the script.

**Table 16-19  GET_TASK_SCRIPT Function Parameters**

| Parameter         | Description                                                                 |
|-------------------|スタンダード(必要) |
| task_name         | The task name that uniquely identifies an existing task.                     |
| type              | The only valid value is 'IMPLEMENTATION'.                                    |
| rec_id            | An identifier of a specific execution of the task.                          |
| act_id            | An identifier of an advisor object that can be targeted by the script.       |
| owner_name        | Owner of the task. If specified, the system will check to see if the current user has read privileges to the task data. |
| execution_name    | An identifier of a specific execution of the task.                          |
| object_id         | An identifier of an advisor object that can be targeted by the script.       |
Table 16-19  (Cont.) GET_TASK_SCRIPT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>Specifies the type of script to generate. The possible values are IMPLEMENTATION and UNDO.</td>
</tr>
<tr>
<td><strong>rec_id</strong></td>
<td>An optional recommendation identifier number that can be used to extract a subset of the implementation script. A zero or the value DBMS_ADVISOR.ADVISOR_ALL indicates all accepted recommendations would be included. The default is to include all accepted recommendations for the task.</td>
</tr>
<tr>
<td><strong>act_id</strong></td>
<td>Optional action identifier number that can be used to extract a single action as a DDL command. A zero or the value DBMS_ADVISOR.ADVISOR_ALL indicates all actions for the recommendation would be included. The default is to include all actions for a recommendation.</td>
</tr>
<tr>
<td><strong>owner_name</strong></td>
<td>An optional task owner name.</td>
</tr>
<tr>
<td><strong>execution_name</strong></td>
<td>An identifier of a specific execution of the task. It is needed only for advisors that allow their tasks to be executed multiple times.</td>
</tr>
<tr>
<td><strong>object_id</strong></td>
<td>An identifier of an advisor object that can be targeted by the script.</td>
</tr>
</tbody>
</table>

Return Values

Returns the script as a CLOB buffer.

Usage Notes

Though the script is ready to execute, Oracle recommends that the user review the script for acceptable locations for new materialized views and indexes.

For a recommendation to appear in a generated script, it must be marked as accepted.

Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
    buf CLOB;
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                         100, 400, 5041, 103, 640445, 680000, 2, 1, SYSDATE, 1, 'SH', 'SELECT AVG(amount_sold) FROM sh.sales');
16.3.19 IMPLEMENT_TASK Procedure

This procedure implements the recommendations of the specified Advisor task.

Syntax

DBMS_ADVISOR.IMPLEMENT_TASK (
    task_name          IN VARCHAR2,
    rec_id             IN NUMBER := NULL,
    exit_on_error      IN BOOLEAN := NULL);

Parameters

Table 16-20 IMPLEMENT_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the task.</td>
</tr>
<tr>
<td>rec_id</td>
<td>An optional recommendation ID.</td>
</tr>
<tr>
<td>exit_on_error</td>
<td>An optional Boolean to exit on the first error.</td>
</tr>
</tbody>
</table>

16.3.20 IMPORT_SQLWKLD_SCHEMA Procedure

This procedure constructs and loads a SQL workload based on schema evidence. The workload is also referred to as a hypothetical workload.

Note:

This procedure is deprecated starting in Oracle Database 11g.

Syntax

DBMS_ADVISOR.IMPORT_SQLWKLD_SCHEMA (
    workload_name         IN VARCHAR2,
    import_mode           IN VARCHAR2 := 'NEW',
    priority              IN NUMBER := 2,
    saved_rows            OUT NUMBER,
    failed_rows           OUT NUMBER);
### Parameters

#### Table 16-21 IMPORT_SQLWKLD_SCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>import_mode</td>
<td>Specifies the action to be taken when storing the workload. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• APPEND Indicates that the collected workload will be added to any existing workload in the task.</td>
</tr>
<tr>
<td></td>
<td>• NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</td>
</tr>
<tr>
<td></td>
<td>• REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is NEW.</td>
</tr>
<tr>
<td>priority</td>
<td>Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.</td>
</tr>
<tr>
<td>failed_rows</td>
<td>Returns the number or rows that were not saved due to syntax or validation errors.</td>
</tr>
<tr>
<td>saved_rows</td>
<td>Returns the number of rows actually saved in the repository.</td>
</tr>
</tbody>
</table>

#### Return Values

This call returns the number of rows saved and failed as output parameters.

#### Usage Notes

To successfully import a hypothetical workload, the target schemas must contain dimensions.

If the `VALID_TABLE_LIST` parameter is not set, the search space may become very large and require a significant amount of time to complete. Oracle recommends that you limit your search space to specific set of tables.

If a task contains valid recommendations from a prior run, adding or modifying task will mark the task as invalid, preventing the viewing and reporting of potentially valuable recommendation data.

#### Examples

```sql
DECLARE
   workload_name VARCHAR2(30);
   saved NUMBER;
   failed NUMBER;
BEGIN
   workload_name := 'My Workload';

   DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
```
16.3.21 IMPORT_SQLWKLD_SQLCACHE Procedure

This procedure creates a SQL workload from the current contents of the server's SQL cache.

Note:
This procedure is deprecated starting in Oracle Database 11g.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE (  
  workload_name      IN VARCHAR2,  
  import_mode        IN VARCHAR2 := 'NEW',  
  priority           IN NUMBER := 2,  
  saved_rows         OUT NUMBER,  
  failed_rows        OUT NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| workload_name | The workload object name that uniquely identifies an existing workload.  
| import_mode  | Specifies the action to be taken when storing the workload. Possible values are: |
|              | • APPEND Indicates that the collected workload will be added to any existing workload in the task. |
|              | • NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. |
|              | • REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. |
| priority     | Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following 1-HIGH, 2-MEDIUM, or 3-LOW. |
| saved_rows   | Returns the number of rows saved as output parameters.  
| failed_rows  | Returns the number of rows that were not saved due to syntax or validation errors. |

Return Values

This call returns the number of rows saved and failed as output parameters.
Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See **RESET_TASK Procedure** to set a task to its initial state.

Examples

```sql
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');

  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.\%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE(workload_name, 'REPLACE', 1, saved, failed);
END;
/
```

16.3.22 IMPORT_SQLWKLD_STS Procedure

This procedure loads a SQL workload from an existing SQL tuning set. A SQL tuning set is typically created from the server workload repository using various time and data filters.

**Note:**

This procedure is deprecated starting in Oracle Database 11g.

Syntax

```sql
DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
  workload_name     IN VARCHAR2,
  sts_name          IN VARCHAR2,
  import_mode       IN VARCHAR2 := 'NEW',
  priority          IN NUMBER := 2,
  saved_rows        OUT NUMBER,
  failed_rows       OUT NUMBER);
```

```sql
DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
  workload_name     IN VARCHAR2,
  sts_owner         IN VARCHAR2,
  sts_name          IN VARCHAR2,
```
import_mode           IN VARCHAR2 := 'NEW',
priority              IN NUMBER := 2,
saved_rows            OUT NUMBER,
failed_rows           OUT NUMBER);

Parameters

Table 16-23    IMPORT_SQLWKLD_STS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>sts_owner</td>
<td>The optional owner of the SQL tuning set.</td>
</tr>
<tr>
<td>sts_name</td>
<td>The name of an existing SQL tuning set workload from which the data will be imported. If the sts_owner value is not provided, the owner will default to the current user.</td>
</tr>
</tbody>
</table>
| import_mode | Specifies the action to be taken when storing the workload. Possible values are:  
  • APPEND Indicates that the collected workload will be added to any existing workload in the task.  
  • NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.  
  • REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.  
  The default value is NEW. |
| priority    | Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW. The default value is 2. |
| saved_rows  | Returns the number of rows actually saved in the repository. |
| failed_rows | Returns the number of rows that were not saved due to syntax or validation errors. |

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

DECLARE
    workload_name VARCHAR2(30);
saved NUMBER;
failed NUMBER;
BEGIN
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH .%');
DBMS_ADVISOR.IMPORT_SQLWKLD_STS(workload_name, 'MY_SQLSET', 'REPLACE', 1,
    saved, failed);
END;
/

16.3.23 IMPORT_SQLWKLD_SUMADV Procedure

This procedure collects a SQL workload from a Summary Advisor workload.

This procedure is intended to assist Oracle9i Database Summary Advisor users in the migration to SQL Access Advisor.

**Note:**
This procedure is deprecated starting in Oracle Database 11g.

**Syntax**

\[
\text{DBMS_ADVISOR.IMPORT_SQLWKLD_SUMADV (}
\text{    workload_name} \quad \text{IN VARCHAR2,}
\text{    import_mode} \quad \text{IN VARCHAR2 := 'NEW',}
\text{    priority} \quad \text{IN NUMBER := 2,}
\text{    sumadv_id} \quad \text{IN NUMBER,}
\text{    saved_rows} \quad \text{OUT NUMBER,}
\text{    failed_rows} \quad \text{OUT NUMBER);}\n\]

**Parameters**

**Table 16-24** IMPORT_SQLWKLD_SUMADV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>import_mode</td>
<td>Specifies the action to be taken when storing the workload. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• APPEND Indicates that the collected workload will be added to any existing workload in the task.</td>
</tr>
<tr>
<td></td>
<td>• NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</td>
</tr>
<tr>
<td></td>
<td>• REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.</td>
</tr>
<tr>
<td></td>
<td>The default value is NEW.</td>
</tr>
</tbody>
</table>
Table 16-24  (Cont.) IMPORT_SQLWKLD_SUMADV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>Specifies the default application priority for each statement that is saved in the workload object. If a Summary Advisor workload statement contains a priority of zero, the default priority will be applied. If the workload statement contains a valid priority, then the Summary Advisor priority will be converted to a comparable SQL Access Advisor priority. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.</td>
</tr>
<tr>
<td>sumadv_id</td>
<td>Specifies the Summary Advisor workload identifier number.</td>
</tr>
<tr>
<td>saved_rows</td>
<td>Returns the number of rows actually saved in the repository.</td>
</tr>
<tr>
<td>failed_rows</td>
<td>Returns the number of rows that were not saved due to syntax or validation errors.</td>
</tr>
</tbody>
</table>

**Return Values**

This call returns the number of rows saved and failed as output parameters.

**Usage Notes**

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See `RESET_TASK Procedure` to set a task to its initial state.

**Examples**

```sql
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
  sumadv_id NUMBER;
BEGIN
  workload_name := 'My Workload';
  sumadv_id := 394;
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SUMADV(workload_name, 'REPLACE', 1, sumadv_id, saved, failed);
END;
/```
16.3.24 IMPORT_SQLWKLD_USER Procedure

This procedure collects a SQL workload from a specified user table.

Note:
This procedure is deprecated starting in Oracle Database 11g.

Syntax

```sql
DBMS_ADVISOR.IMPORT_SQLWKLD_USER (
    workload_name         IN VARCHAR2,
    import_mode           IN VARCHAR2 := 'NEW',
    owner_name            IN VARCHAR2,
    table_name            IN VARCHAR2,
    saved_rows            OUT NUMBER,
    failed_rows           OUT NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>import_mode</td>
<td>Specifies the action to be taken when storing the workload. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• APPEND Indicates that the collected workload will be added to any existing workload in the task.</td>
</tr>
<tr>
<td></td>
<td>• NEW Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</td>
</tr>
<tr>
<td></td>
<td>• REPLACE Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.</td>
</tr>
<tr>
<td></td>
<td>The default value is NEW.</td>
</tr>
<tr>
<td>owner_name</td>
<td>Specifies the owner name of the table or view from which workload data will be collected.</td>
</tr>
<tr>
<td>table_name</td>
<td>Specifies the name of the table or view from which workload data will be collected.</td>
</tr>
<tr>
<td>saved_rows</td>
<td>Returns the number of rows actually saved in the workload object.</td>
</tr>
<tr>
<td>failed_rows</td>
<td>Returns the number of rows that were not saved due to syntax or validation errors.</td>
</tr>
</tbody>
</table>

Return Values

This call returns the number of rows saved and failed as output parameters.
Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

DECLARE
    workload_name VARCHAR2(30);
    saved NUMBER;
    failed NUMBER;
BEGIN
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');

    DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
    DBMS_ADVISOR.IMPORT_SQLWKLD_USER(workload_name, 'REPLACE', 'SH',
                                      'USER_WORKLOAD', saved, failed);
END;
/

16.3.25 INTERRUPT_TASK Procedure

This procedure stops a currently executing task.

The task will end its operations as it would at a normal exit. The user will be able to access any recommendations that exist to this point.

Syntax

DBMS_ADVISOR.INTERRUPT_TASK (task_name IN VARCHAR2);

Parameters

Table 16-26  INTERRUPT_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>A single Advisor task name that will be interrupted.</td>
</tr>
</tbody>
</table>

Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
    DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/

While this session is executing its task, you can interrupt the task from a second session using the following statement:

BEGIN
    DBMS_ADVISOR.INTERRUPT_TASK('My Task');
END;
/

16.3.26 MARK_RECOMMENDATION Procedure

This procedure marks a recommendation for import or implementation.

Syntax

DBMS_ADVISOR.MARK_RECOMMENDATION (  
    task_name          IN VARCHAR2
    id                 IN NUMBER,
    action             IN VARCHAR2);

Parameters

Table 16-27  MARK_RECOMMENDATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task.</td>
</tr>
<tr>
<td>id</td>
<td>The recommendation identifier number assigned by the Advisor.</td>
</tr>
<tr>
<td>action</td>
<td>The recommendation action setting. The possible actions are:</td>
</tr>
<tr>
<td></td>
<td>• ACCEPT Marks the recommendation as accepted. With this setting, the recommendation will appear in implementation and undo scripts.</td>
</tr>
<tr>
<td></td>
<td>• IGNORE Marks the recommendation as ignore. With this setting, the recommendation will not appear in an implementation or undo script.</td>
</tr>
<tr>
<td></td>
<td>• REJECT Marks the recommendation as rejected. With this setting, the recommendation will not appear in any implementation or undo scripts.</td>
</tr>
</tbody>
</table>

Usage Notes

For a recommendation to be implemented, it must be marked as accepted. By default, all recommendations are considered accepted and will appear in any generated scripts.
Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
    attribute VARCHAR2(100);
    rec_id NUMBER;
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                         100, 400, 5041, 103, 640445, 680000, 2,
                                         1, SYSDATE, 1, 'SH', 'SELECT
                                         AVG(amount_sold) FROM sh.sales WHERE promo_id = 10');
    DBMS_ADVISOR.EXECUTE_TASK(task_name);

    rec_id := 1;
    DBMS_ADVISOR.MARK_RECOMMENDATION(task_name, rec_id, 'REJECT');
END;
/

16.3.27 QUICK_TUNE Procedure

This procedure performs an analysis and generates recommendations for a single SQL statement.

This provides a shortcut method of all necessary operations to analyze the specified SQL statement. The operation creates a task using the specified task name. The task will be created using a specified Advisor task template. Finally, the task will be executed and the results will be saved in the repository.

Syntax

DBMS_ADVISOR.QUICK_TUNE (advisor_name IN VARCHAR2,
                          task_name IN VARCHAR2,
                          attr1 IN CLOB,
                          attr2 IN VARCHAR2 := NULL,
                          attr3 IN NUMBER := NULL,
                          template IN VARCHAR2 := NULL,
                          implement IN BOOLEAN := FALSE,
                          description IN VARCHAR2 := NULL);
### Parameters

#### Table 16-28 QUICK_TUNE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>advisor_name</td>
<td>Name of the Advisor that will perform the analysis.</td>
</tr>
<tr>
<td>task_name</td>
<td>Name of the task.</td>
</tr>
<tr>
<td>attr1</td>
<td>Advisor-specific attribute in the form of a CLOB variable.</td>
</tr>
<tr>
<td>attr2</td>
<td>Advisor-specific attribute in the form of a VARCHAR2 variable.</td>
</tr>
<tr>
<td>attr3</td>
<td>Advisor-specific attribute in the form of a NUMBER.</td>
</tr>
<tr>
<td>template</td>
<td>Name of an existing task or template from which the initial settings need to be copied.</td>
</tr>
<tr>
<td>implement</td>
<td>Flag specifying whether to implement the task.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the task.</td>
</tr>
</tbody>
</table>

#### Usage Notes

If indicated by the user, the final recommendations can be implemented by the procedure.

The task will be created using either a specified SQL Access task template or the built-in default template of SQLACCESS_GENERAL. The workload will only contain the specified statement, and all task parameters will be defaulted.

attr1 must be the single SQL statement to tune. For the SQL Access Advisor, attr2 is the user who would execute the single statement. If omitted, the current user will be used.

#### Examples

```sql
DECLARE
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.QUICK_TUNE(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_name,
                          'SELECT AVG(amount_sold) FROM sh.sales WHERE promo_id=10');
END;
/
```

### 16.3.28 RESET_SQLWKLD Procedure

This procedure resets a workload to its initial starting point.

Resetting the workload has the effect of removing all journal and log messages, and recalculating necessary volatility and usage statistics.
**Note:**

This procedure is deprecated starting in Oracle Database 11g.

Syntax

```sql
DBMS_ADVISOR.RESET_SQLWKLD (  
    workload_name    IN VARCHAR2);
```

Parameters

**Table 16-29  RESET_SQLWKLD Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The SQL Workload object name that uniquely identifies an existing workload.</td>
</tr>
</tbody>
</table>

Usage Notes

RESET_SQLWKLD should be executed after any workload adjustments such as adding or removing SQL statements.

Examples

```sql
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP', 100,400,5041,103,640445,680000,2,1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
        FROM sh.sales WHERE promo_id = 10');

  DBMS_ADVISOR.RESET_SQLWKLD(workload_name);
END;
/
```

**16.3.29 RESET_TASK Procedure**

This procedure re-initializes the metadata for the specified task. The task status will be set to INITIAL.

Syntax

```sql
DBMS_ADVISOR.RESET_TASK (  
    task_name    IN VARCHAR2);
```
Parameters

Table 16-30  RESET_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The task name that uniquely identifies an existing task.</td>
</tr>
</tbody>
</table>

Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
    DBMS_ADVISOR.EXECUTE_TASK(task_name);
    DBMS_ADVISOR.RESET_TASK(task_name);
END;
/

16.3.30 SET_DEFAULT_SQLWKLD_PARAMETER Procedure

This procedure modifies the default value for a user parameter within a SQL Workload object or SQL Workload object template.

A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting. When a default value is changed for a parameter, workload objects will inherit the new value when they are created.

>Note:

This procedure is deprecated starting in Oracle Database 11g.

Syntax

DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (parameter IN VARCHAR2,
value IN VARCHAR2);

DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (parameter IN VARCHAR2,
value IN NUMBER);
Parameters

Table 16-31  SET_DEFAULT_SQLWKLD_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>The name of the data parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the workload object type, but not necessarily unique to all workload object types. Various object types may use the same parameter name for different purposes.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the specified parameter. The value can be specified as a string or a number. If the value is DBMS_ADVISOR.DEFAULT, the value will be reset to the default value.</td>
</tr>
</tbody>
</table>

Usage Notes

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

Examples

```
BEGIN
  DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER('VALID_TABLE_LIST','SH.%');
END;
/
```

16.3.31 SET_DEFAULT_TASK_PARAMETER Procedure

This procedure modifies the default value for a user parameter within a task or a template.

A user parameter is a simple variable that stores various attributes that affect various Advisor operations. When a default value is changed for a parameter, tasks will inherit the new value when they are created.

A default task is different from a regular task. The default value is the initial value that will be inserted into a newly created task, while setting a task parameter with SET_TASK_PARAMETER sets the local value only. Thus, SET_DEFAULT_TASK_PARAMETER has no effect on an existing task.

Syntax

```
DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (  
  advisor_name        IN VARCHAR2
  parameter           IN VARCHAR2,
  value               IN VARCHAR2);
```

```
DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (  
  advisor_name        IN VARCHAR2
  parameter           IN VARCHAR2,
  value               IN NUMBER);
```
Parameters

Table 16-32  SET_DEFAULT_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>advisor_name</td>
<td>Specifies the unique advisor name as defined in the view DBA_ADVISOR_DEFINITIONS.</td>
</tr>
<tr>
<td>parameter</td>
<td>The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the specified task parameter. The value can be specified as a string or a number.</td>
</tr>
</tbody>
</table>

Examples

BEGIN

DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER(DBMS_ADVISOR.SQLACCESS_ADVISOR, 'VALID_TABLE_LIST', 'SH.%');

END;

/  

16.3.32 SET_SQLWKLD_PARAMETER Procedure

This procedure modifies a user parameter within a SQL Workload object or SQL Workload object template.

A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

Note:

This procedure is deprecated starting in Oracle Database 11g.

Syntax

```
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (  
    workload_name IN VARCHAR2,  
    parameter     IN VARCHAR2,  
    value         IN VARCHAR2);

DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (  
    workload_name IN VARCHAR2,  
    parameter     IN VARCHAR2,  
    value         IN NUMBER);
```
 Parameters

**Table 16-33  SET_SQLWKLD_PARAMETER Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The SQL Workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>parameter</td>
<td>The name of the data parameter to be modified. Parameter names are not case sensitive.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the specified parameter. The value can be specified as a string or a number. If the value is <code>DBMS_ADVISOR.DEFAULT</code>, the value will be reset to the default value.</td>
</tr>
</tbody>
</table>

**Usage Notes**

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

**Examples**

```sql
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name, 'VALID_TABLE_LIST','SH.%');
END;
/
```

### 16.3.33 SET_TASK_PARAMETER Procedure

This procedure modifies a user parameter within an Advisor task or a template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

**Syntax**

```sql
DBMS_ADVISOR.SET_TASK_PARAMETER (  
  task_name   IN VARCHAR2  
  parameter   IN VARCHAR2,  
  value       IN VARCHAR2);  

DBMS_ADVISOR.SET_TASK_PARAMETER (  
  task_name   IN VARCHAR2  
  parameter   IN VARCHAR2,  
  value       IN NUMBER);  
```
Parameters

Table 16-34  SET_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The Advisor task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>parameter</td>
<td>The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the specified task parameter. The value can be specified as a string or a number. If the value is DEFAULT, the value will be reset to the default value.</td>
</tr>
</tbody>
</table>

Usage Notes

A task cannot be modified unless it is in its initial state. See RESET_TASK Procedure to set a task to its initial state. See your Advisor-specific documentation for further information on using this procedure.

SQL Access Advisor Task Parameters

Table 16-35 lists SQL Access Advisor task parameters.
Table 16-35  SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYSIS_SCOPE</td>
<td>A comma-separated list that specifies the tuning artifacts to consider during analysis. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>- <strong>ALL</strong> Short name for specifying INDEX, MVIEW, TABLE, and PARTITION.</td>
</tr>
<tr>
<td></td>
<td>- <strong>EVALUATION</strong> Causes a read-only evaluation of the specified workload. No new recommendations will be made. Can only be specified alone.</td>
</tr>
<tr>
<td></td>
<td>- <strong>INDEX</strong> Allows the SQL Access Advisor to recommend index structure changes.</td>
</tr>
<tr>
<td></td>
<td>- <strong>MVIEW</strong> Allows the SQL Access Advisor to recommend materialized view and log changes.</td>
</tr>
<tr>
<td></td>
<td>- <strong>PARTITION</strong> Allows the SQL Access Advisor to recommend partition options. Use this in conjunction with the INDEX, MVIEW, and TABLE options.</td>
</tr>
<tr>
<td></td>
<td>- <strong>TABLE</strong> Allows the SQL Access Advisor to make base-table recommendations. In this release, the only base-table recommendation is partitioning.</td>
</tr>
<tr>
<td></td>
<td>Using the new keywords, the following combinations are valid:</td>
</tr>
<tr>
<td></td>
<td>- INDEX</td>
</tr>
<tr>
<td></td>
<td>- MVIEW</td>
</tr>
<tr>
<td></td>
<td>- INDEX, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- INDEX, MVIEW, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- INDEX, TABLE, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- MVIEW, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- MVIEW, TABLE, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- INDEX, MVIEW, TABLE, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- TABLE, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- TABLE, PARTITION</td>
</tr>
<tr>
<td></td>
<td>- EVALUATION</td>
</tr>
<tr>
<td></td>
<td>The default value is <strong>INDEX</strong>. The data type is STRINGLIST.</td>
</tr>
<tr>
<td>CREATION_COST</td>
<td>When set to <strong>true</strong> (default), the SQL Access Advisor will weigh the cost of creation of the access structure (index or materialized view) against the frequency of the query and potential improvement in the query execution time. When set to <strong>false</strong>, the cost of creation is ignored. The data type is STRING.</td>
</tr>
<tr>
<td>DAYS_TO_EXPIRE</td>
<td>Specifies the expiration time in days for the current SQL Access Advisor task. Once the task expires, it will become a candidate for removal by an automatic purge operation. The value is relative to the last modification date. Once the task expires, it becomes a candidate for removal by an automatic purge operation. The value is relative to the last modification date. The data type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>The possible values are:</td>
</tr>
<tr>
<td></td>
<td>- an integer in the range of 0 to 2147483647</td>
</tr>
<tr>
<td></td>
<td>- ADVISOR_UNLIMITED</td>
</tr>
<tr>
<td></td>
<td>- ADVISOR_UNUSED</td>
</tr>
<tr>
<td></td>
<td>The default value is <strong>30</strong>.</td>
</tr>
<tr>
<td>DEF_EM_TEMPLATE</td>
<td>Contains the default task or template name from which the Enterprise Manager SQL Access Advisor Wizard reads its initial values. The data type is STRING.</td>
</tr>
<tr>
<td></td>
<td>The default value is <strong>SQLACCESS_EMTASK</strong>.</td>
</tr>
</tbody>
</table>
Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_INDEX_OWNER</td>
<td>Specifies the default owner for new index recommendations. When a script is created, this value will be used to qualify the index name. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· Existing schema name. Quoted identifiers are supported.</td>
</tr>
<tr>
<td></td>
<td>· ADVISOR_UNUSED</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is STRING.</td>
</tr>
<tr>
<td>DEF_INDEX_TABLESPACE</td>
<td>Specifies the default tablespace for new index recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· Existing tablespace name. Quoted identifiers are supported.</td>
</tr>
<tr>
<td></td>
<td>· ADVISOR_UNUSED No tablespace clause will be present in the script for indexes.</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is STRING.</td>
</tr>
<tr>
<td>DEF_MVIEW_OWNER</td>
<td>Specifies the default owner for new materialized view recommendations. When a script is created, this value will be used to qualify the materialized view name. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· Existing schema name. Quoted identifiers are supported.</td>
</tr>
<tr>
<td></td>
<td>· ADVISOR_UNUSED</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is STRING.</td>
</tr>
<tr>
<td>DEF_MVIEW_TABLESPACE</td>
<td>Specifies the default tablespace for new materialized view recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· Existing tablespace name. Quoted identifiers are supported.</td>
</tr>
<tr>
<td></td>
<td>· ADVISOR_UNUSED No tablespace clause will be present in the script for materialized view logs.</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is STRING.</td>
</tr>
<tr>
<td>DEF_MVLOG_TABLSPACE</td>
<td>Specifies the default tablespace for new materialized view log recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· Existing tablespace name. Quoted identifiers are supported.</td>
</tr>
<tr>
<td></td>
<td>· ADVISOR_UNUSED No tablespace clause will be present in the script for materialized view logs.</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is STRING.</td>
</tr>
<tr>
<td>DEF_PARTITION_TABLESPACE</td>
<td>Specifies the default tablespace for new partitioning recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· Existing tablespace name. Quoted identifiers are supported.</td>
</tr>
<tr>
<td></td>
<td>· ADVISOR_UNUSED No tablespace clause will be present in the script for materialized views.</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is STRING.</td>
</tr>
</tbody>
</table>
Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DML_VOLATILITY</td>
<td>When set to TRUE, the SQL Access Advisor will consider the impact of index maintenance and materialized view refresh in determining the recommendations. It will limit the access structure recommendations involving columns or tables that are frequently updated. For example, if there are too many DML statements on a column, then it may favor a B-tree index over a bitmap index on that column. For this process to be effective, the workload must include DML (insert/update/delete/merge/direct path inserts) statements that represent the update behavior of the application. The data type is STRING. See the related parameter refresh_mode.</td>
</tr>
</tbody>
</table>
| END_TIME        | Specifies an end time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYYY HH24:MI:SS, where:  
• DD is the numeric date  
• MM is the numeric month  
• YYYY is the numeric year  
• HH is the hour in 24 hour format  
• MI is the minute  
• SS is the second  
The data type is STRING. |
| EVALUATION_ONLY | This parameter is maintained for backward compatibility. All values will be translated and placed into the ANALYSIS_SCOPE task parameter. If set to TRUE, causes SQL Access Advisor to analyze the workload, but only comment on how well the current configuration is supporting it. No tuning recommendations will be generated. Possible values are:  
• FALSE  
• TRUE  
The default value is FALSE. The data type is STRING. |
| EXECUTION_TYPE  | This parameter is maintained for backward compatibility. All values will be translated and placed into the ANALYSIS_SCOPE task parameter. The translated values are:  
• FULL => FULL  
• INDEX_ONLY => INDEX  
• MVIEW_ONLY => MVIEW  
• MVIEW_LOG_ONLY => MVIEW LOG_ONLY  
The type of recommendations that is desired. Possible values:  
• FULL All supported recommendation types will be considered.  
• INDEX_ONLY The SQL Access Advisor will only consider index solutions as recommendations.  
• MVIEW_ONLY The SQL Access Advisor will consider materialized view and materialized view log solutions as recommendations.  
• MVIEW_LOG_ONLY The SQL Access Advisor will only consider materialized view log solutions as recommendations.  
The default value is FULL. The data type is STRINGLIST. |
### Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| IMPLEMENT_EXIT_ON_ERROR    | When performing an IMPLEMENT_TASK operation, this parameter will control behavior when an action fails to implement. If set to TRUE, IMPLEMENT_TASK will stop on the first unexpected error. The possible values are:  
  - TRUE  
  - FALSE  
The default value is TRUE. The data type is STRING. |
| INDEX_NAME_TEMPLATE         | Specifies the method by which new index names are formed. If the TASK_ID is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the TASK_ID in the template. Once formatted, the maximum size of a name is 30 characters. Valid keywords are:  
  - Any literal value up to 22 characters.  
  - TABLE Causes the parent table name to be substituted into the index name. If the name is too long, it will be trimmed to fit.  
  - TASK_ID Causes the current task identifier number to be inserted in hexadecimal form.  
  - SEQ Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token.  
The default template is table_IDX$_$task_idsequence. The data type is STRING. |
| INVALID_ACTION_LIST         | Contains a fully qualified list of actions that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.  
An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness.  
During a task execution, if a SQL statement's action matches a name in the action list, it will not be processed by the task. An action name is case sensitive. The possible values are:  
  - single action  
  - comma-delimited action list  
  - ADVISOR_UNUSED  
The default value is ADVISOR_UNUSED. The data type is STRINGLIST. |
### Table 16-35 (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| INVALID_MODULE_LIST   | Contains a fully qualified list of modules that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.  
A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness.  
During a task execution, if a SQL statement's module matches a name in the list, it will not be processed by the task. A module name is case sensitive.  
The possible values are:  
- single application  
- comma-delimited module list  
- ADVISOR_UNUSED  
The default value is ADVISOR_UNUSED. The data type is STRINGLIST. |
| INVALID_SQLSTRING_LIST| Contains a fully qualified list of text strings that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted values are supported.  
A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness.  
During a task execution, if a SQL statement contains a string in the SQL string list, it will not be processed by the task.  
The possible values are:  
- single string  
- comma-delimited string list  
- ADVISOR_UNUSED  
The default value is ADVISOR_UNUSED. The data type is STRINGLIST. |
| INVALID_USERNAME_LIST | Contains a fully qualified list of user names that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.  
During a task execution, if a SQL statement's user name matches a name in the user name list, it will not be processed by the task. A user name is not case sensitive unless it is quoted.  
The possible values are:  
- single user name  
- comma-delimited user name list  
- ADVISOR_UNUSED  
The default value is ADVISOR_UNUSED. The data type is STRINGLIST. |
Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOURNALING</td>
<td>Controls the logging of messages to the journal (DBA_ADVISOR_JOURNAL and USER_ADVISOR_JOURNAL views). The higher the setting, the more information is logged to the journal. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• UNUSED: no journal messages</td>
</tr>
<tr>
<td></td>
<td>• FATAL: explanation of fatal conditions</td>
</tr>
<tr>
<td></td>
<td>• ERROR: explanation of errors</td>
</tr>
<tr>
<td></td>
<td>• WARNING: explanation of warnings</td>
</tr>
<tr>
<td></td>
<td>• INFORMATION: information message</td>
</tr>
<tr>
<td></td>
<td>• INFORMATION2: common information</td>
</tr>
<tr>
<td></td>
<td>• INFORMATION3: common information</td>
</tr>
<tr>
<td></td>
<td>• INFORMATION4: common information</td>
</tr>
<tr>
<td></td>
<td>• INFORMATION5: common information</td>
</tr>
<tr>
<td></td>
<td>• INFORMATION6: common information</td>
</tr>
<tr>
<td></td>
<td>Each journal value represents all recorded messages at that level or lower. For example, when choosing WARNING, all messages marked WARNING as well as ERROR and FATAL will be recorded in the repository. INFORMATION6 represents the most thorough message recording and UNUSED is the least. The default value is INFORMATION. The data type is NUMBER.</td>
</tr>
<tr>
<td>LIMITED_PARTITION_SCHEMES</td>
<td>User can suggest that the Partition Expert cut off the number of partitioning schemes to investigate. This can help with cutting down the run time of the advisor. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• An integer in the range of 1 to 10</td>
</tr>
<tr>
<td></td>
<td>• ADVISOR_UNUSED</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is NUMBER.</td>
</tr>
<tr>
<td>MAX_NUMBER_PARTITIONS</td>
<td>Limits the number of partitions the advisor will recommend for any base table, index, or materialized view. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• An integer in the range of 1 to 4294967295</td>
</tr>
<tr>
<td></td>
<td>• ADVISOR_UNLIMITED</td>
</tr>
<tr>
<td></td>
<td>• ADVISOR_UNUSED</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNLIMITED. The data type is NUMBER.</td>
</tr>
<tr>
<td>MODE</td>
<td>Specifies the mode by which Access Advisor will operate during an analysis. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• LIMITED Indicates the Advisor will attempt to a quick job by limiting the search-space of candidate recommendations, and correspondingly, the results may be of a low quality.</td>
</tr>
<tr>
<td></td>
<td>• COMPREHENSIVE Indicates the Advisor will search a large pool of candidates that may take long to run, but the resulting recommendations will be of the highest quality.</td>
</tr>
<tr>
<td></td>
<td>The default value is COMPREHENSIVE. The data type is STRING.</td>
</tr>
</tbody>
</table>
### Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| MVIEW_NAME_TEMPLATE     | Specifies the method by which new materialized view names are formed. If the TASK_ID is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the TASK_ID in the template. The format is any combination of keyword tokens and literals. However, once formatted, the maximum size of a name is 30 characters. Valid tokens are:  
  - Any literal value up to 22 characters.  
  - TASK_ID Causes the current task identifier number to be inserted in hexadecimal form.  
  - SEQ Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token.  
The default template is: MV$$_task_idsequence. The data type is STRING. |
| ORDER_LIST              | This parameter has been deprecated.  
Contains the primary natural order in which the Access Advisor processes workload elements during the analysis operation. To determine absolute natural order, Access Advisor sorts the workload using ORDER_LIST values. A comma must separate multiple order keys. Possible values are:  
  - BUFFER_GETS Sets the order using the SQL statement's buffer-get count value.  
  - CPU_TIME Sets the order using the SQL statement's CPU time value.  
  - DISK_READS Sets the order using the SQL statement's disk-read count value.  
  - ELAPSED_TIME Sets the order using the SQL statement's elapsed time value.  
  - EXECUTIONS Sets the order using the SQL statement's execution frequency value.  
  - OPTIMIZER_COST Sets the order using the SQL statement's optimizer cost value.  
  - I/O Sets the order using the SQL statement's I/O count value.  
  - PRIORITY Sets the order using the user-supplied business priority value.  
All values are accessed in descending order, where a high value is considered more interesting than a low value. The default value is PRIORITY, OPTIMIZER_COST. The data type is STRINGLIST. |
### Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| PARTITION_NAME_TEMPLATE    | Specifies the method by which new partition names are formed. The format is any combination of keyword tokens and literals. However, once formatted, the maximum size of a name is 30 characters. Valid tokens are:  
  - Any literal value up to 22 characters.  
  - `table` - Causes the parent table name to be substituted into the partition name. If the name is too long, it will be trimmed to fit.  
  - `task_id` - Causes the current task identifier number to be inserted in hexadecimal form.  
  - `sequence` - Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token.  
  
The default template is `PTN$$_table_task_idsequence`. The data type is `STRING`. |
| PARTITIONING_GOAL          | Specifies the approach used to make partitioning recommendations. One possible value is `PERFORMANCE`, which is the default. The data type is `STRING`. |
| PARTITIONING_TYPES         | Specifies the type of partitioning used. Possible values are `RANGE` and `HASH`. The data type is `STRING`. |
| RANKING_MEASURE            | Contains the primary natural order in which the SQL Access Advisor processes workload elements during the analysis operation. To determine absolute natural order, SQL Access Advisor sorts the workload using `RANKING_MEASURE` values. A comma must separate multiple order keys. Possible values are:  
  - `BUFFER_GETS` Sets the order using the SQL statement's buffer-get count value.  
  - `CPU_TIME` Sets the order using the SQL statement's CPU time value.  
  - `DISK_READS` Sets the order using the SQL statement's disk-read count value.  
  - `ELAPSED_TIME` Sets the order using the SQL statement's elapsed time value.  
  - `EXECUTIONS` Sets the order using the SQL statement's elapsed time value.  
  - `OPTIMIZER_COST` Sets the order using the SQL statement's optimizer cost value.  
  - `PRIORITY` Sets the order using the user-supplied business priority value.  
  All values are accessed in descending order, where a high value is considered more interesting than a low value.  
  
The default value is `PRIORITY, OPTIMIZER_COST`. The data type is `STRINGLIST`. |
| RECOMMEND_MV_EXACT_TEXT_M  | When considering candidate materialized views, exact text match solutions will only be included if this parameter contains `TRUE`. The possible values are:  
  - `TRUE`  
  - `FALSE`  
  The default value is `TRUE`. The data type is `STRING`. |
### Table 16-35 (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| RECOMMENDED_TABLESPACES | Allows the SQL Access Advisor to recommend optimal tablespaces for any partitioning scheme. If this is not set, the SQL Access Advisor will simply recommend a partitioning method but give no advice on physical storage. Possible values are:  
- TRUE  
- FALSE (the default)  
The data type is STRING. |
| REFRESH_MODE       | Specifies whether materialized views are refreshed ON_DEMAND or ON_COMMIT. This will be used to weigh the impact of materialized view refresh when the parameter dml_volatility is set to TRUE. Possible values are:  
- ON_DEMAND  
- ON_COMMIT  
The default value is ON_DEMAND. The data type is STRING. |
| REPORT_DATE_FORMAT | This is the default date and time formatting template. The default format is DD/MM/YYYYHH24:MI. The data type is STRING. |
| SHOW_RETAINS       | Controls the display of RETAIN actions within an implementation script and the SQL Access Advisor wizard. The possible values are:  
- TRUE  
- FALSE  
The default value is TRUE. The data type is STRING. |
| SQL_LIMIT          | Specifies the number of SQL statements to be analyzed. The SQL_LIMIT filter is applied after all other filters have been applied. For example, if only statements referencing the table hr.employees are to be accepted, the SQL_LIMIT value will be only apply to those statements. When used in conjunction with the parameter ORDER_LIST, SQL Access Advisor will process the most interesting SQL statements by ordering the statements according to the specified sort keys. The possible values are:  
- An integer in the range of 1 to 2147483647  
- ADVISOR_UNLIMITED  
- ADVISOR_UNUSED  
The default value is ADVISOR_UNUSED. The data type is NUMBER. |
| START_TIME         | Specifies a start time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYYY HH24:MI:SS, where:  
- DD is the numeric date  
- MM is the numeric month  
- YYYY is the numeric year  
- HH is the hour in 24 hour format  
- MI is the minute  
- SS is the second  
The data type is STRING. |
Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORAGE_CHANGE</td>
<td>Contains the amount of space adjustment that can be consumed by SQL Access Advisor recommendations. Zero or negative values are only permitted if the workload scope is marked as FULL. When the SQL Access Advisor produces a set of recommendations, the resultant physical structures must be able to fit into the budgeted space. A space budget is computed by adding the STORAGE_CHANGE value to the space quantity currently used by existing access structures. A negative STORAGE_CHANGE value may force SQL Access Advisor to remove existing structures in order to shrink space demand. Possible values:</td>
</tr>
<tr>
<td></td>
<td>- Any valid integer including negative values, zero and positive values. The default value is ADVISOR_UNLIMITED. The data type is NUMBER.</td>
</tr>
<tr>
<td>TIME_LIMIT</td>
<td>Specifies the time in minutes that the SQL Access Advisor can use to perform an analysis operation. If the SQL Access Advisor reaches a specified recommendation quality or all input data has been analyzed, processing will terminate regardless of any remaining time. Possible values:</td>
</tr>
<tr>
<td></td>
<td>- An integer in the range of 1 to 10,000</td>
</tr>
<tr>
<td></td>
<td>- ADVISOR_UNLIMITED</td>
</tr>
<tr>
<td></td>
<td>The default value is 720 (12 hours). The data type is NUMBER. Note that specifying ADVISOR_UNLIMITED has the same effect as setting the parameter to the maximum of 10,000 (about one week). The SQL Access Advisor will never run for more than 10,000 minutes.</td>
</tr>
<tr>
<td>VALID_ACTION_LIST</td>
<td>Contains a fully qualified list of actions that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During a task execution, if a SQL statement's action does not match a name in the action list, it will not be processed by the task. An action name is case sensitive. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>- single action</td>
</tr>
<tr>
<td></td>
<td>- comma-delimited action list</td>
</tr>
<tr>
<td></td>
<td>- ADVISOR_UNUSED</td>
</tr>
<tr>
<td></td>
<td>The default value is ADVISOR_UNUSED. The data type is STRINGLIST.</td>
</tr>
</tbody>
</table>
### Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| VALID_MODULE_LIST    | Contains a fully qualified list of application modules that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During a task execution, if a SQL statement's module does not match a name in the module list, it will not be processed by the task. A module name is case sensitive. The possible values are:  
  - single application  
  - comma-delimited module list  
  - ADVISOR_UNUSED  
  The default value is ADVISOR_UNUSED. The data type is STRINGLIST. |
| VALID_SQLSTRING_LIST | Contains a fully qualified list of text strings that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness. During a task execution, if a SQL statement does not contain string in the SQL string list, it will not be processed by the task. The possible values are:  
  - single string  
  - comma-delimited string list  
  - ADVISOR_UNUSED  
  The default value is ADVISOR_UNUSED. The data type is STRINGLIST. |
| VALID_TABLE_LIST     | Contains a fully qualified list of tables that are eligible for tuning. The list elements are comma-delimited, and quoted identifiers are supported. Wildcard specifications are supported for tables. The default value is all tables within the user's scope are eligible for tuning. Supported wildcard character is %. A % wildcard matches any set of consecutive characters. When a SQL statement is processed, it will not be accepted unless at least one referenced table is specified in the valid table list. If the list is unused, then all table references within a SQL statement are considered valid. The valid syntax for a table reference is:  
  - schema.table  
  - schema  
  - schema.% (equivalent to schema)  
  - comma-delimited action list  
  - ADVISOR_UNUSED  
  The possible values are:  
  - single table reference  
  - comma-delimited reference list  
  - ADVISOR_UNUSED  
  The default value is ADVISOR_UNUSED. The data type is TABLELIST. |
Table 16-35  (Cont.) SQL Access Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| VALID_USERNAME_LIST        | Contains a fully qualified list of user names that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. During a task execution, if a SQL statement's user name does not match a name in the user name list, it will not be processed by the task. A user name is not case sensitive unless it is quoted. The possible values are:  
  - single user name  
  - comma-delimited user name list  
  - ADVISOR_UNUSED  
  The default value is ADVISOR_UNUSED. The data type is STRINGLIST. |
| WORKLOAD_SCOPE             | Describes the level of application coverage the workload represents. Possible values are FULL and PARTIAL. FULL Should be used if the workload contains all interesting application SQL statements for the targeted tables. PARTIAL (default) Should be used if the workload contains anything less than a full representation of the interesting application SQL statements for the targeted tables. The data type is STRING. |

Segment Advisor Parameters

Table 16-36 lists the input task parameters that can be set in the Segment Advisor using the SET_TASK_PARAMETER procedure.

Table 16-36  Segment Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| MODE         | The data to use for analysis. The default value is COMPREHENSIVE, and the possible values are:  
  - LIMITED: Analysis restricted to statistics available in the Automatic Workload Repository  
  - COMPREHENSIVE: Analysis based on sampling and Automatic Workload Repository statistics |
| TIME_LIST     | The time limit for which the Advisor should run. It is specified in seconds, and the default and possible values are UNLIMITED.                                                                                 |
| RECOMMEND_ALL | Whether to generate recommendations for all segments. The default value is TRUE. If set to TRUE, it generates recommendations all segments specified by the user. If set to FALSE, it generates recommendations for only those objects that are eligible for shrink. |

Examples

DECLARE  
  task_id NUMBER;  
  task_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
    task_name);
    DBMS_ADVISOR.SET_TASK_PARAMETER(task_name, 'VALID_TABLELIST',
    'SH.%,SCOTT.EMP');
END;
/

Undo Advisor Task Parameters

Table 16-37 lists the input task parameters that can be set in the Undo Advisor using the
SET_TASK_PARAMETER procedure.

Table 16-37 Undo Advisor Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_OBJECTS</td>
<td>The undo tablespace of the system. There is no default value, and the possible value is UNDO_TBS.</td>
</tr>
<tr>
<td>START_SNAPSHOT</td>
<td>The starting time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value and the possible values are the valid snapshot numbers in the AWR repository.</td>
</tr>
<tr>
<td>END_SNAPSHOT</td>
<td>The ending time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value and the possible values are the valid snapshot numbers in the AWR repository.</td>
</tr>
<tr>
<td>BEGIN_TIME_SEC</td>
<td>The number of seconds between the beginning time of the period and now. Describes a period of time for the system to perform analysis. BEGIN_TIME_SEC should be greater than END_TIME_SEC. There is no default value and the possible values are any positive integer.</td>
</tr>
<tr>
<td>END_TIME_SEC</td>
<td>The number of seconds between the ending time of the period and now. END_TIME_SEC should be less than BEGIN_TIME_SEC. There is no default value and the possible values are any positive integer.</td>
</tr>
</tbody>
</table>

Examples

DECLARE
    tname  VARCHAR2(30);
    oid    NUMBER;
BEGIN
    DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor Task');
    DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null',
    oid);
    DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
    DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
    DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
    DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'INSTANCE', 1);
    DBMS_ADVISOR.EXECUTE_TASK(tname);
END;
/
Automatic Database Diagnostic Monitor (ADDM) Task Parameters

Table 16-38 lists the input task parameters that can be set in ADDM using the `SET_TASK_PARAMETER` procedure.

Table 16-38  ADDM Task Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_SNAPSHOT</td>
<td>The starting time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value, and the possible values are the valid snapshot numbers in the AWR repository.</td>
</tr>
<tr>
<td>END_SNAPSHOT</td>
<td>The ending time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value, and the possible values are the valid snapshot numbers in the AWR repository.</td>
</tr>
<tr>
<td>DB_ID</td>
<td>The database for <code>START_SNAPSHOT</code> and <code>END_SNAPSHOT</code>. The default value is the current database ID.</td>
</tr>
<tr>
<td>INSTANCE</td>
<td>The instance for <code>START_SNAPSHOT</code> and <code>END_SNAPSHOT</code>. The default value is 0 or UNUSED, and the possible values are all positive integers. By default, all instances are analyzed.</td>
</tr>
<tr>
<td>INSTANCES</td>
<td>If the <code>INSTANCE</code> parameter has been set, <code>INSTANCES</code> is ignored. The default value is UNUSED, and the possible values are comma-separated list of instance numbers (for example, &quot;1, 3, 5&quot;). By default, all instances are analyzed.</td>
</tr>
<tr>
<td>DBIOEXPECTED</td>
<td>The average time to read the database block in microseconds. The default value is 10 milliseconds, and the possible values are system-dependent.</td>
</tr>
</tbody>
</table>

Examples

The following creates and executes an ADDM task for the current database and an AWR snapshot range between 19 and 26. Note that this example will analyze all instances, whether you have only one or an Oracle RAC database.

```sql
DECLARE
tid NUMBER;
tname VARCHAR2(30) := 'ADDM_TEST';
BEGIN
  DBMS_ADVISOR.CREATE_TASK('ADDM', tid, tname, 'my test');
  DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', '19');
  DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', '26');
  DBMS_ADVISOR.EXECUTE_TASK(tname);
END;
/`
SQL Tuning Advisor Task Parameters

See the DBMS_SQLTUNE package and Oracle Database SQL Tuning Guide for more information.

16.3.34 TUNE_MVIEW Procedure

This procedure shows how to decompose a materialized view into multiple views and to restate the materialized view to be optimized for fast refresh and query rewrite. It also shows how to fix materialized view logs and to enable query rewrite.

Syntax

```sql
DBMS_ADVISOR.TUNE_MVIEW (  
    task_name IN OUT VARCHAR2,  
    mv_create_stmt IN [CLOB | VARCHAR2]);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The task name for querying the results in a catalog view. If not specified, the database generates a task name, and then returns.</td>
</tr>
<tr>
<td>mv_create_stmt</td>
<td>The original materialized view creation statement.</td>
</tr>
</tbody>
</table>

Usage Notes

Executing TUNE_MVIEW generates two sets of output results: one for the implementation, and the other for undoing the implementation. The output is accessible through USER_TUNE_MVIEW and DBA_TUNE_MVIEW views. You can also use DBMS_ADVISOR.GET_TASK_SCRIPT and DBMS_ADVISOR.CREATE_FILE to print the TUNE_MVIEW results into a script file for later execution.

Table 16-40 USER_TUNE_MVIEW and DBA_TUNE_MVIEW Views

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER</td>
<td>The name of the materialized view owner.</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>The name of the task. This name serves as a key to access the set of recommendations.</td>
</tr>
<tr>
<td>SCRIPT_TYPE</td>
<td>Recommendation ID that indicates whether the row is for the IMPLEMENTATION or UNDO script.</td>
</tr>
</tbody>
</table>
Table 16-40  (Cont.) USER_TUNE_MVIEW and DBA_TUNE_MVIEW Views

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_ID</td>
<td>Action ID used as the command order number.</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>For TUNE_MVIEW output, this column represents the following statements, and includes statement properties such as REFRESH and REWRITE options:</td>
</tr>
<tr>
<td></td>
<td>· CREATE MATERIALIZED VIEW LOG</td>
</tr>
<tr>
<td></td>
<td>· ALTER MATERIALIZED VIEW LOG FORCE</td>
</tr>
<tr>
<td></td>
<td>· [CREATE</td>
</tr>
</tbody>
</table>

Examples

The following example shows how to use TUNE_MVIEW to optimize a CREATE MATERIALIZED VIEW statement:

DECLARE
    v_tname VARCHAR2(30);
BEGIN
    v_tname := 'mview_task';
    DBMS_ADVISOR.TUNE_MVIEW(
        task_name      => v_tname,
        mv_create_stmt => 'CREATE MATERIALIZED VIEW LOG ON "OE"."ORDERS" WITH ROWID AS SELECT * FROM orders');
END;

You can view the results by querying USER_TUNE_MVIEW or DBA_TUNE_MVIEW as the following example (sample output included):

```
SET LINESIZE 120
COL TASK_NAME FORMAT a20
COL STATEMENT FORMAT a40

SELECT *
FROM USER_TUNE_MVIEW
WHERE TASK_NAME='mview_task'
AND SCRIPT_TYPE='IMPLEMENTATION';
```

<table>
<thead>
<tr>
<th>TASK_NAME</th>
<th>ACTION_ID</th>
<th>SCRIPT_TYPE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>mview_task</td>
<td>1</td>
<td>IMPLEMENTATION</td>
<td>CREATE MATERIALIZED VIEW LOG ON &quot;OE&quot;.&quot;ORDERS&quot; WITH ROWID</td>
</tr>
<tr>
<td>mview_task</td>
<td>2</td>
<td>IMPLEMENTATION</td>
<td>ALTER MATERIALIZED VIEW LOG FORCE ON &quot;OE&quot;.&quot;ORDERS&quot; ADD ROWID</td>
</tr>
<tr>
<td>mview_task</td>
<td>3</td>
<td>IMPLEMENTATION</td>
<td>CREATE MATERIALIZED VIEW OE.OMV REFRESH</td>
</tr>
</tbody>
</table>
Alternatively, you can save the output results in an external script file as in the following example:

```sql
CREATE DIRECTORY TUNE_RESULTS_DIR AS '/tmp';
GRANT READ, WRITE ON DIRECTORY TUNE_RESULTS_DIR TO PUBLIC;
BEGIN
    DBMS_ADVISOR.CREATE_FILE(
        buffer => DBMS_ADVISOR.GET_TASK_SCRIPT( task_name => 'mview_task')
        , location => 'TUNE_RESULTS_DIR'
        , filename => 'mview_create.sql'
    );
END;
```

The preceding statement will save the results in `/tmp/mview_create.sql`.

See Also:

Oracle Database SQL Tuning Guide for more information about using the TUNE_MVIEW procedure

### 16.3.35 UPDATE_OBJECT Procedure

This procedure updates an existing task object.

Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level.

**Syntax**

```
DBMS_ADVISOR.UPDATE_OBJECT ( 
    task_name       IN VARCHAR2
    object_id       IN NUMBER,
    attr1           IN VARCHAR2 := NULL,
    attr2           IN VARCHAR2 := NULL,
    attr3           IN VARCHAR2 := NULL,
    attr4           IN CLOB := NULL,
    attr5           IN VARCHAR2 := NULL);
```

**Parameters**

**Table 16-41  UPDATE_OBJECT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>A valid advisor task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>object_id</td>
<td>The advisor-assigned object identifier.</td>
</tr>
</tbody>
</table>
Table 16-41  (Cont.) UPDATE_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr1</td>
<td>Advisor-specific data. If set to NULL, there will be no effect on the target object.</td>
</tr>
<tr>
<td>attr2</td>
<td>Advisor-specific data. If set to NULL, there will be no effect on the target object.</td>
</tr>
<tr>
<td>attr3</td>
<td>Advisor-specific data. If set to NULL, there will be no effect on the target object.</td>
</tr>
<tr>
<td>attr4</td>
<td>Advisor-specific data. If set to NULL, there will be no effect on the target object.</td>
</tr>
<tr>
<td>attr5</td>
<td>Advisor-specific data. If set to NULL, there will be no effect on the target object.</td>
</tr>
</tbody>
</table>

The attribute parameters have different values depending upon the object type. See Oracle Database Administrator's Guide for details regarding these parameters and object types.

Usage Notes

If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or lob column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

Examples

```sql
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  obj_id NUMBER;
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_OBJECT (task_name,'SQL',NULL,NULL,NULL,'SELECT * FROM SH.SALES',obj_id);
  DBMS_ADVISOR.UPDATE_OBJECT (task_name, obj_id,NULL,NULL,NULL,'SELECT count(*) FROM SH.SALES');
END;
/
```

See Also:

Oracle Database Administrator’s Guide for further information regarding the Segment Advisor
16.3.36 UPDATE_REC_ATTRIBUTES Procedure

This procedure updates the owner, name, and tablespace for a recommendation.

Syntax

DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES (  
    task_name            IN VARCHAR2  
    rec_id               IN NUMBER,  
    action_id            IN NUMBER,  
    attribute_name       IN VARCHAR2,  
    value                IN VARCHAR2);  

Parameters

Table 16-42 UPDATE_REC_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>rec_id</td>
<td>The Advisor-generated identifier number that is assigned to the recommendation.</td>
</tr>
<tr>
<td>action_id</td>
<td>The Advisor-generated action identifier that is assigned to the particular command.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Name of the attribute to be changed. The valid values are:</td>
</tr>
<tr>
<td></td>
<td>- owner The new owner of the object.</td>
</tr>
<tr>
<td></td>
<td>- name The new name of the object.</td>
</tr>
<tr>
<td></td>
<td>- tablespace The new tablespace for the object.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies the new value for the recommendation attribute.</td>
</tr>
</tbody>
</table>

Usage Notes

Recommendation attributes cannot be modified unless the task has successfully executed.

Examples

DECLARE  
    task_id NUMBER;  
    task_name VARCHAR2(30);  
    workload_name VARCHAR2(30);  
    attribute VARCHAR2(100);  
BEGIN  
    task_name := 'My Task';  
    workload_name := 'My Workload';  

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);  
    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');  
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);  
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',  
        100,400,5041,103,640445,680000,2,
1,SYSDATE,1,'SH','SELECT AVG(amount_sold) FROM sh.sales WHERE promo_id = 10');
   DBMS_ADVISOR.EXECUTE_TASK(task_name);
attribute := 'SH';
   DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES(task_name, 1, 3, 'OWNER',
   attribute);
END;
/

16.3.37 UPDATE_SQLWKLD_ATTRIBUTES Procedure

This procedure changes various attributes of a SQL Workload object or template.

**Note:**

This procedure is deprecated starting in Oracle Database 11g.

**Syntax**

```
DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES (workload_name        IN VARCHAR2,
                                             new_name             IN VARCHAR2 := NULL,
                                             description          IN VARCHAR2 := NULL,
                                             read_only            IN VARCHAR2 := NULL,
                                             is_template          IN VARCHAR2 := NULL,
                                             how_created          IN VARCHAR2 := NULL);
```

**Parameters**

Table 16-43 UPDATE_SQLWKLD_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>new_name</td>
<td>The new workload object name. If the value is NULL or contains the value ADVISOR_UNUSED, the workload will not be renamed. A task name can be up to 30 characters long.</td>
</tr>
<tr>
<td>description</td>
<td>A new workload description. If the value is NULL or contains the value ADVISOR_UNUSED, the description will not be changed. Names can be up to 256 characters long.</td>
</tr>
<tr>
<td>read_only</td>
<td>Set to TRUE so it cannot be changed.</td>
</tr>
<tr>
<td>is_template</td>
<td>TRUE if workload is to be used as a template.</td>
</tr>
</tbody>
</table>
Table 16-43  (Cont.) UPDATE_SQLWKLD_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>how_created</td>
<td>Indicates a source application name that initiated the workload creation. If the value is NULL or contains the value ADVISOR_UNUSED, the source will not be changed.</td>
</tr>
</tbody>
</table>

Examples

```sql
DECLARE
    workload_name VARCHAR2(30);
BEGIN
    workload_name := 'My Workload';
    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                        100,400,5041,103,640445,680000,2,
                                        1,SYSDATE,1,'SH','SELECT
                                        AVG(amount_sold) FROM sh.sales WHERE promo_id = 10');
    DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES(workload_name,'New workload name');
END;
/```

16.3.38 UPDATE_SQLWKLD_STATEMENT Procedure

This procedure updates an existing SQL statement in a specified SQL workload.

**Note:**

This procedure is deprecated starting in Oracle Database 11g.

Syntax

```sql
DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (workload_name IN VARCHAR2,
                                        sql_id IN NUMBER,
                                        application IN VARCHAR2 := NULL,
                                        action IN VARCHAR2 := NULL,
                                        priority IN NUMBER := NULL,
                                        username IN VARCHAR2 := NULL);
```

```sql
DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (workload_name IN VARCHAR2,
                                        search IN VARCHAR2,
                                        updated OUT NUMBER,
                                        application IN VARCHAR2 := NULL,
                                        action IN VARCHAR2 := NULL,
                                        ...
DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (workload_name IN VARCHAR2,
                                        search IN VARCHAR2,
                                        updated OUT NUMBER,
                                        application IN VARCHAR2 := NULL,
                                        action IN VARCHAR2 := NULL,
```

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
priority IN NUMBER := NULL,
username IN VARCHAR2 := NULL);

Parameters

Table 16-44 UPDATE_SQLWKLD_STATEMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload_name</td>
<td>The SQL Workload object name that uniquely identifies an existing workload.</td>
</tr>
<tr>
<td>sql_id</td>
<td>The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant DBMS_ADVISOR.ADVISOR_ALL.</td>
</tr>
<tr>
<td>updated</td>
<td>Returns the number of statements changed by a searched update.</td>
</tr>
<tr>
<td>application</td>
<td>Specifies a business application name that will be associated with the SQL statement. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository.</td>
</tr>
<tr>
<td>action</td>
<td>Specifies the application action for the statement. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository.</td>
</tr>
<tr>
<td>priority</td>
<td>The relative priority of the SQL statement. The value must be one of the following: 1 - HIGH, 2 - MEDIUM, or 3 - LOW. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository.</td>
</tr>
<tr>
<td>username</td>
<td>The Oracle user name that executed the SQL statement. If the value is NULL or contains the value ADVISOR_UNUSED, then the column will not be updated in the repository. Because a user name is an Oracle identifier, the username value must be entered exactly like it is stored in the database. For example, if the user SCOTT is the executing user, then you must provide the user identifier SCOTT in all uppercase letters. The database does not recognize the user scott as a match for SCOTT.</td>
</tr>
<tr>
<td>search</td>
<td>Disabled.</td>
</tr>
</tbody>
</table>

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

DECLARE
  workload_name VARCHAR2(30);
  updated NUMBER;
  id NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY',
'ROLLUP',
100,400,5041,103,640445,680000,2,
1,SYSDATE,1,'SH','SELECT
AVG(amount_sold)
FROM sh.sales WHERE promo_id = 10');

SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
WHERE workload_name = 'My Workload';

DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT(workload_name, id);
END;
/

16.3.39 UPDATE_TASK_ATTRIBUTES Procedure

This procedure changes various attributes of a task or a task template.

Syntax

DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES (task_name          IN VARCHAR2
new_name           IN VARCHAR2 := NULL,
description        IN VARCHAR2 := NULL,
read_only          IN VARCHAR2 := NULL,
is_template        IN VARCHAR2 := NULL,
how_created        IN VARCHAR2 := NULL);

Parameters

Table 16-45  UPDATE_TASK_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The Advisor task name that uniquely identifies an existing task.</td>
</tr>
<tr>
<td>new_name</td>
<td>The new Advisor task name. If the value is NULL or contains the value ADVISOR_UNUSED, the task will not be renamed. A task name can be up to 30 characters long.</td>
</tr>
<tr>
<td>description</td>
<td>A new task description. If the value is NULL or contains the value ADVISOR_UNUSED, the description will not be changed. Names can be up to 256 characters long.</td>
</tr>
<tr>
<td>read_only</td>
<td>Sets the task to read-only. Possible values are: TRUE and FALSE. If the value is NULL or contains the value ADVISOR_UNUSED, the setting will not be changed.</td>
</tr>
<tr>
<td>is_template</td>
<td>Marks the task as a template. Physically, there is no difference between a task and a template; however, a template cannot be executed. Possible values are: TRUE and FALSE. If the value is NULL or contains the value ADVISOR_UNUSED, the setting will not be changed.</td>
</tr>
<tr>
<td>how_created</td>
<td>Indicates a source application name that initiated the task creation. If the value is NULL or contains the value ADVISOR_UNUSED, the source will not be changed.</td>
</tr>
</tbody>
</table>
Examples

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES(task_name,'New Task Name');
    DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES('New Task Name',NULL,'New description');
END;
/
DBMS_ALERT supports asynchronous notification of database events (alerts). By appropriate use of this package and database triggers, an application can notify itself whenever values of interest in the database are changed.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Restrictions
- Exceptions
- Operational Notes
- Examples
- Summary of DBMS_ALERT Subprograms

### 17.1 DBMS_ALERT Overview

This scenario describes a possible use of the DBMS_ALERT package.

Suppose a graphics tool is displaying a graph of some data from a database table. The graphics tool can, after reading and graphing the data, wait on a database alert (WAITONE) covering the data just read. The tool automatically wakes up when the data is changed by any other user. All that is required is that a trigger be placed on the database table, which performs a signal (SIGNAL) whenever the trigger is fired.

### 17.2 DBMS_ALERT Security Model

Security on this package can be controlled by granting EXECUTE on this package to selected users or roles. You might want to write a cover package on top of this one that restricts the alert names used. EXECUTE privilege on this cover package can then be granted rather than on this package.

### 17.3 DBMS_ALERT Constants

The DBMS_ALERT package uses the constants listed and described in this topic.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXWAIT</td>
<td>INTEGER</td>
<td>86400000</td>
<td>The maximum time to wait for an alert (1000 days which is essentially forever).</td>
</tr>
</tbody>
</table>
17.4 DBMS_ALERT Restrictions

Because database alerters issue commits, they cannot be used with Oracle Forms. For more information on restrictions on calling stored procedures while Oracle Forms is active, refer to your Oracle Forms documentation.

17.5 DBMS_ALERT Exceptions

DBMS_ALERT raises the application error -20000 on error conditions.

Table 17-2 shows the messages and the procedures that can raise them.

17.6 DBMS_ALERT Operational Notes

This topic lists notes related to general and specific applications. Also, a list of DBMS_ALERT error messages is provided.

- Alerts are transaction-based. This means that the waiting session is not alerted until the transaction signalling the alert commits. There can be any number of concurrent signalers of a given alert, and there can be any number of concurrent waiters on a given alert.
- A waiting application is blocked in the database and cannot do any other work.
- An application can register for multiple events and can then wait for any of them to occur using the WAITANY procedure.
- An application can also supply an optional timeout parameter to the WAITONE or WAITANY procedures. A timeout of 0 returns immediately if there is no pending alert.
- The signalling session can optionally pass a message that is received by the waiting session.
- Alerts can be signalled more often than the corresponding application wait calls. In such cases, the older alerts are discarded. The application always gets the latest alert (based on transaction commit times).
- If the application does not require transaction-based alerts, the DBMS_PIPE package may provide a useful alternative.

See Also:

- DBMS_PIPE

- If the transaction is rolled back after the call to SIGNAL, no alert occurs.
- It is possible to receive an alert, read the data, and find that no data has changed. This is because the data changed after the prior alert, but before the data was read for that prior alert.
- Usually, Oracle is event-driven; this means that there are no polling loops. There are two cases where polling loops can occur:
Shared mode. If your database is running in shared mode, a polling loop is required to check for alerts from another instance. The polling loop defaults to one second and can be set by the **SET_DEFAULTS** procedure.

**WAITANY** procedure. If you use the **WAITANY** procedure, and if a signalling session does a signal but does not commit within one second of the signal, a polling loop is required so that this uncommitted alert does not camouflage other alerts. The polling loop begins at a one second interval and exponentially backs off to 30-second intervals.

**Table 17-2 DBMS_ALERT Error Messages**

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORU-10001 lock request error, status: N</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10015 error: N waiting for pipe status</td>
<td>WAITANY</td>
</tr>
<tr>
<td>ORU-10016 error: N sending on pipe 'X'</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10017 error: N receiving on pipe 'X'</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10019 error: N on lock request</td>
<td>WAIT</td>
</tr>
<tr>
<td>ORU-10020 error: N on lock request</td>
<td>WAITANY</td>
</tr>
<tr>
<td>ORU-10021 lock request error; status: N</td>
<td>REGISTER</td>
</tr>
<tr>
<td>ORU-10022 lock request error, status: N</td>
<td>SIGNAL</td>
</tr>
<tr>
<td>ORU-10023 lock request error; status N</td>
<td>WAITONE</td>
</tr>
<tr>
<td>ORU-10024 there are no alerts registered</td>
<td>WAITANY</td>
</tr>
<tr>
<td>ORU-10025 lock request error; status N</td>
<td>REGISTER</td>
</tr>
<tr>
<td>ORU-10037 attempting to wait on uncommitted signal from same session</td>
<td>WAITONE</td>
</tr>
</tbody>
</table>

**17.7 DBMS_ALERT Examples**

In this example, suppose that you want to graph average salaries by department, for all employees. Your application needs to know whenever EMP is changed.

Your application would look similar to this code:

```sql
DBMS_ALERT.REGISTER('emp_table_alert');
<<readagain>>:
/* ... read the emp table and graph it */
  DBMS_ALERT.WAITONE('emp_table_alert', :message, :status);
  if status = 0 then goto <<readagain>>; else
    /* ... error condition */
```

The **EMP table** would have a trigger similar to this:

```sql
CREATE TRIGGER emptrig AFTER INSERT OR UPDATE OR DELETE ON emp
BEGIN
  DBMS_ALERT.SIGNAL('emp_table_alert', 'message_text');
END;
```

When the application is no longer interested in the alert, it makes this request:

```sql
DBMS_ALERT.REMOVE('emp_table_alert');
```
This reduces the amount of work required by the alert signaller. If a session exits (or
dies) while registered alerts exist, the alerts are eventually cleaned up by future users
of this package.

The example guarantees that the application always sees the latest data, although it
may not see every intermediate value.

### 17.8 Summary of DBMS_ALERT Subprograms

This table lists the DBMS_ALERT subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER Procedure</td>
<td>Receives messages from an alert</td>
</tr>
<tr>
<td>REMOVE Procedure</td>
<td>Disables notification from an alert</td>
</tr>
<tr>
<td>REMOVEALL Procedure</td>
<td>Removes all alerts for this session from the registration list</td>
</tr>
<tr>
<td>SET_DEFAULTS Procedure</td>
<td>Sets the polling interval</td>
</tr>
<tr>
<td>SIGNAL Procedure</td>
<td>Signals an alert (send message to registered sessions)</td>
</tr>
<tr>
<td>WAITANY Procedure</td>
<td>Waits <code>timeout</code> seconds to receive alert message from an alert registered for session</td>
</tr>
<tr>
<td>WAITONE Procedure</td>
<td>Waits <code>timeout</code> seconds to receive message from named alert</td>
</tr>
</tbody>
</table>

#### 17.8.1 REGISTER Procedure

This procedure lets a session register interest in an alert.

**Syntax**

```sql
DBMS_ALERT.REGISTER (  
    name      IN  VARCHAR2,
    cleanup   IN  BOOLEAN DEFAULT TRUE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert in which this session is interested</td>
</tr>
<tr>
<td>cleanup</td>
<td>Specifies whether to perform cleanup of any extant orphaned pipes used by the DBMS_ALERT package. This cleanup is only performed on the first call to REGISTER for each package instantiation. The default for the parameter is TRUE.</td>
</tr>
</tbody>
</table>
Usage Notes

A session can register interest in an unlimited number of alerts. Alerts should be deregistered when the session no longer has any interest, by calling REMOVE.

17.8.2 REMOVE Procedure

This procedure enables a session that is no longer interested in an alert to remove that alert from its registration list. Removing an alert reduces the amount of work done by signalers of the alert.

Syntax

DBMS_ALERT.REMOVE (name IN VARCHAR2);

Parameters

Table 17-5    REMOVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert (case-insensitive) to be removed from registration list.</td>
</tr>
</tbody>
</table>

Usage Notes

Removing alerts is important because it reduces the amount of work done by signalers of the alert. If a session dies without removing the alert, that alert is eventually (but not immediately) cleaned up.

17.8.3 REMOVEALL Procedure

This procedure removes all alerts for this session from the registration list. You should do this when the session is no longer interested in any alerts.

This procedure is called automatically upon first reference to this package during a session. Therefore, no alerts from prior sessions which may have terminated abnormally can affect this session.

This procedure always performs a commit.

Syntax

DBMS_ALERT.REMOVEALL;
17.8.4 SET_DEFAULTS Procedure

In case a polling loop is required, use the SET_DEFAULTS procedure to set the polling interval.

Syntax

```
DBMS_ALERT.SET_DEFAULTS (sensitivity  IN  NUMBER);
```

Parameters

Table 17-6  SET_DEFAULTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensitivity</td>
<td>Polling interval, in seconds, to sleep between polls. The default interval is five seconds.</td>
</tr>
</tbody>
</table>

17.8.5 SIGNAL Procedure

This procedure signals an alert. The effect of the SIGNAL call only occurs when the transaction in which it is made commits. If the transaction rolls back, SIGNAL has no effect.

All sessions that have registered interest in this alert are notified. If the interested sessions are currently waiting, they are awakened. If the interested sessions are not currently waiting, they are notified the next time they do a wait call.

Multiple sessions can concurrently perform signals on the same alert. Each session, as it signals the alert, blocks all other concurrent sessions until it commits. This has the effect of serializing the transactions.

Syntax

```
DBMS_ALERT.SIGNAL (name     IN  VARCHAR2,
                    message  IN  VARCHAR2);
```

Parameters

Table 17-7  SIGNAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert to signal.</td>
</tr>
<tr>
<td>message</td>
<td>Message, of 1800 bytes or less, to associate with this alert.</td>
</tr>
<tr>
<td></td>
<td>This message is passed to the waiting session. The waiting session might be able to avoid reading the database after the alert occurs by using the information in the message.</td>
</tr>
</tbody>
</table>
17.8.6 WAITANY Procedure

Call this procedure to wait for an alert to occur for any of the alerts for which the current session is registered.

Syntax

```sql
DBMS_ALERT.WAITANY (
    name OUT VARCHAR2,
    message OUT VARCHAR2,
    status OUT INTEGER,
    timeout IN NUMBER DEFAULT MAXWAIT);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Returns the name of the alert that occurred.</td>
</tr>
<tr>
<td>message</td>
<td>Returns the message associated with the alert.</td>
</tr>
<tr>
<td></td>
<td>This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITANY, the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.</td>
</tr>
<tr>
<td>status</td>
<td>Values returned:</td>
</tr>
<tr>
<td></td>
<td>0 - alert occurred</td>
</tr>
<tr>
<td></td>
<td>1 - timeout occurred</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum time to wait for an alert.</td>
</tr>
<tr>
<td></td>
<td>If no alert occurs before timeout seconds, this returns a status of 1.</td>
</tr>
</tbody>
</table>

Usage Notes

An implicit COMMIT is issued before this procedure is executed. The same session that waits for the alert may also first signal the alert. In this case remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.

Exceptions

-20000, ORU-10024: there are no alerts registered.

17.8.7 WAITONE Procedure

This procedure waits for a specific alert to occur.

An implicit COMMIT is issued before this procedure is executed. A session that is the first to signal an alert can also wait for the alert in a subsequent transaction. In this case, remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.
Syntax

```sql
DBMS_ALERT.GETONE (
    name      IN   VARCHAR2,
    message   OUT  VARCHAR2,
    status    OUT  INTEGER,
    timeout   IN   NUMBER DEFAULT MAXWAIT);
```

Parameters

Table 17-9  WAITONE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the alert to wait for.</td>
</tr>
<tr>
<td>message</td>
<td>Returns the message associated with the alert.</td>
</tr>
<tr>
<td></td>
<td>This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITONE, the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.</td>
</tr>
<tr>
<td>status</td>
<td>Values returned:</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum time to wait for an alert.</td>
</tr>
</tbody>
</table>
The `DBMS_APP_CONT` package provides an interface to determine if the in-flight transaction on a now unavailable session committed or not, and if the last call on that session completed or not.

**See Also:**

`Oracle Database Development Guide` for explanations of application continuity and Transaction Guard, and the relationship between these two features:

- "Using Transaction Guard"

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_APP_CONT Subprograms

### 18.1 DBMS_APP_CONT Overview

The DBMS_APP_CONT package can be used to solve this example issue.

**Problem Description**

One of the fundamental problems for recovering applications after an outage is that the commit message that is sent back to the client is not durable. If there is a break between the client and the server, the client sees an error message indicating that the communication failed. This error does not inform the application whether the submission executed any commit operations or if a procedural call, ran to completion executing all expected commits and session state changes or failed part way through or yet worse, is still running disconnected from the client.

**GET_LTXID_OUTCOME**

The purpose of the `GET_LTXID_OUTCOME Procedure` is to determine if the in-flight transaction on a now unavailable session completed or not. It is used when the original session returned an error due to unavailability. Situations that can cause such session unavailability may occur at the session, instance, server, or network, and result from planned or unplanned outages. When such an outage occurs, the application receives a disconnection error. Such an error provides no insight as to whether the transaction committed. It also does not reveal what the application might have been expecting from that commit if it had returned.
18.2 DBMS_APP_CONT Security Model

Applications must have the EXECUTE privilege on the DBMS_APP_CONT package.

To grant this privilege, ask your database administrator to run the following SQL statement:

```
GRANT execute on DBMS_APP_CONT to application user;
```

18.3 Summary of DBMS_APP_CONT Subprograms

The DBMS_APP_CONT package contains the GET_LTXID_OUTCOME Procedure.

Table 18-1  DBMS_APP_CONT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_LTXID_OUTCOME Procedure</td>
<td>Lets customer applications and third party application servers determine the transactional status of the last session when that session becomes unavailable.</td>
</tr>
</tbody>
</table>

18.3.1 ADD_SQL_CONNECTION_TEST Procedure

This procedure adds a new connection test that is used during draining sessions before planned maintenance begins. Use this procedure when the SQL connection test is not covered by standard tests. The test is enabled when added. If the optional service name qualifier is provided, the test only applies only to that service name.

Syntax

```
DBMS_SESSION.ADD_SQL_CONNECTION_TEST (  
  connection_test          IN VARCHAR2  
  service_name             IN VARCHAR2   DEFAULT NULL);  
```

Parameters

Table 18-2  ADD_SQL_CONNECTION_TEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION_TEST</td>
<td>The SQL text used to test and drain connections.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>Optional service name qualifier.</td>
</tr>
</tbody>
</table>

Usage Notes

The ADD_SQL_CONNECTION_TEST Procedure adds a connection test for the purpose of draining sessions before planned maintenance begins. The connection test is used by
the application to test connections that are marked for draining. Sessions are set for draining at stop and relocate operations for services or PDBs. When set the RDBMS closes the connection while draining so the application sees no errors during planned maintenance. You can enter as many CONNECTION TESTs as needed. They are used only during planned maintenance. The tests apply to all RAC instances.

Check online documentation for latest updates on service qualifier availability.

Added connection can be viewed by querying the view `DBA_CONNECTION_TESTS`.

This procedure is owned by `SYS` and is granted to users for execution at `CDB$ROOT` or `PDB` levels, or when not multitenant, at dictionary level.

### 18.3.2 DELETE_SQL_CONNECTION_TEST Procedure

This procedure deletes a connection test that is no longer needed for planned draining. Removing a test applies immediately to all RAC instances where the PDB is open.

**Syntax**

```
DBMS_SESSION.DELETE_SQL_CONNECTION_TEST (
    connection_test          IN VARCHAR2,
    service_name             IN VARCHAR2   DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION_TEST</td>
<td>The SQL text used to test and drain connections.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>Optional service name qualifier.</td>
</tr>
<tr>
<td></td>
<td>If the optional SERVICE_NAME qualifier is provided, only the test for that service name is deleted.</td>
</tr>
</tbody>
</table>

**Usage Notes**

If you are not certain if a test should be deleted, you can disable the test using `DISABLE_CONNECTION_TEST` Procedure. Only custom SQL tests can be deleted. Predefined tests cannot be deleted. Check for latest updates on service qualifier availability.

This procedure is owned by `SYS` at `CDB$ROOT` or `PDB` level, or `SYS` for when not multitenant.

Connection tests and their status can be checked by querying the view `DBA_CONNECTION_TESTS`.

### 18.3.3 DISABLE_CONNECTION_TEST Procedure

This procedure disables usage of a connection test during draining of sessions. Disabling a test applies immediately to all RAC instances where the PDB is open.

**Syntax**

```
DBMS_SESSION.DISABLE_CONNECTION_TEST ( connection_test_type IN VARCHAR2, 
                              connection_test IN VARCHAR2,
                              service_name IN VARCHAR2 DEFAULT NULL)
```

**Usage Notes**

This procedure is owned by `SYS` at `CDB$ROOT` or `PDB` level, or `SYS` for when not multitenant.
Parameters

Table 18-4  DISABLE_CONNECTION_TEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION_TEST_TYPE</td>
<td>The permitted values are:</td>
</tr>
<tr>
<td></td>
<td>•  DBMS_SESSION.SQL_TEST</td>
</tr>
<tr>
<td></td>
<td>•  DBMS_SESSION.PING_TEST</td>
</tr>
<tr>
<td></td>
<td>•  DBMS_SESSION.ENDREQUEST_TEST</td>
</tr>
<tr>
<td>CONNECTION_TEST</td>
<td>The SQL text used to test and drain connections.</td>
</tr>
<tr>
<td></td>
<td>This parameter is allowed only if the value of CONNECTION_TEST_TYPE is SQL_TEST.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>Optional service name qualifier. If the optional service name qualifier is provided, only the test for that service name is enabled. A disable at service name level takes precedence over an enable at PDB level. That is the PDB can be enabled, and the service disabled.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is owned by SYS and is granted to users for execution at CDB$ROOT or PDB levels, or when not multitenant, at dictionary level.

Connection tests and their status can be checked by querying the view DBA_CONNECTION_TESTS.

18.3.4 ENABLE_CONNECTION_TEST Procedure

This procedure enables usage of a connection test for draining database sessions before planned maintenance. Enabling a test applies immediately to all RAC instances where the PDB is open.

Syntax

```
DBMS_SESSION.ENABLE_CONNECTION_TEST (  
    connection_test_type IN VARCHAR2,  
    connection_test    IN VARCHAR2,  
    service_name       IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 18-5  ENABLE_CONNECTION_TEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| CONNECTION_TEST_TYPE | The connection type used when managing connection tests for draining before planned maintenance. See ADD, DELETE, ENABLE, DISABLE procedures for connection tests. The permitted values are:  
  - DBMS_SESSION.SQL_TEST  
  - DBMS_SESSION.PING_TEST  
  - DBMS_SESSION.ENDREQUEST_TEST |
| CONNECTION_TEST      | The SQL text used to test and drain connections at the RDBMS before planned maintenance starts. This parameter is allowed only if the value of CONNECTION_TEST_TYPE is SQL_TEST. |
| SERVICE_NAME         | Optional service name qualifier. If the optional service name qualifier is provided, only the test for that service name is enabled. An enable at service name level overrides any higher-level disables. That is, the PDB can be disabled, and the service enabled. |

Usage Notes

- This procedure is owned by SYS and is granted to users for execution at CDB$ROOT or PDB levels, or when not multitenant, at dictionary level.
- ENABLE_CONNECTION_TEST enables a connection test for draining sessions during planned maintenance. The enable operation applies to all RAC instances where the PDB is open. It persists across database restarts.
- This procedure is owned by SYS and is granted to users for execution at CDB$ROOT or PDB levels, or when not multitenant, at dictionary level.

18.3.5 GET_LTXID_OUTCOME Procedure

This procedure lets customer applications and third party application servers determine the transactional status of the last session when that session becomes unavailable.

Syntax

```sql
DBMS_APP_CONT.GET_LTXID_OUTCOME (  
  client_ltxid IN RAW,  
  committed OUT BOOLEAN,  
  user_call_completed OUT BOOLEAN)
```
Parameters

Table 18-6 GET_LTXID_OUTCOME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_ltxid</td>
<td>Client-side logical transaction ID. Obtain the LTXID from the previous failed session using the client driver provided APIs - getLTXID for JDBC, and LogicalTransactionId for ODP.net., and OCI_ATTR_GET with LTXID for OCI.</td>
</tr>
<tr>
<td>committed</td>
<td>Returns TRUE if the transaction with the named logical LTXID has COMMITTED. Returns FALSE if the logical LTXID has not COMMITTED. When returning FALSE, the procedure blocks the LTXID from further use so that there is no possibility of previous in-flight work committing this LTXID.</td>
</tr>
<tr>
<td>user_call_completed</td>
<td>Whether all information has been returned to the client. Examples of such messages are the number of rows processed when using autocommit or commit on success, parameter and function results when calling PL/SQL, or PL/SQL with more work to do after the COMMIT. Applications that expect to use data returned from the commit in order to function correctly must look at this second parameter.</td>
</tr>
</tbody>
</table>

Exceptions

Table 18-7 GET_LTXID_OUTCOME Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-14950 - SERVER_AHEAD</td>
<td>The server is ahead so the transaction is both an old transaction and one which has already committed. This is an error as the application is passing an older LTXID that is not the last used for that session. The purpose of GET_LTXID_OUTCOME is to return the current transaction outcome for that session after a recoverable outage.</td>
</tr>
<tr>
<td>ORA-14951 - CLIENT_AHEAD</td>
<td>The client is ahead of the server. This can happen if the server has been flashed backed, recovered using media recovery, or is a standby that has opened earlier with data loss.</td>
</tr>
<tr>
<td>ORA-14906 - SAME_SESSION</td>
<td>Executing GET_LTXID_OUTCOME is not supported on the session owning the LTXID as it blocks further processing on that session after a recoverable outage.</td>
</tr>
<tr>
<td>ORA-14909 - COMMIT_BLOCKED</td>
<td>Your session has been blocked from committing by another user with the same username using GET_LTXID_OUTCOME. GET_LTXID_OUTCOME should only be called on dead sessions. Please check with your application administrator.</td>
</tr>
<tr>
<td>ORA-14952 - ERROR</td>
<td>The outcome cannot be determined. During processing an error happened. The error stack shows the error detail.</td>
</tr>
</tbody>
</table>
19

DBMS_APPLICATION_INFO

Application developers can use the DBMS_APPLICATION_INFO package with Oracle Trace and the SQL trace facility to record names of executing modules or transactions in the database for later use when tracking the performance of various modules and debugging.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Summary of DBMS_APPLICATION_INFO Subprograms

19.1 DBMS_APPLICATION_INFO Overview

Registering the application allows system administrators and performance tuning specialists to track performance by module. System administrators can also use this information to track resource use by module. When an application registers with the database, its name and actions are recorded in the $SESSION and $SQLAREA views.

19.2 DBMS_APPLICATION_INFO Security Model

No further privileges are required. The DBMSAPIN.SQL script is already run as a part of standard database creation.

Note:

The public synonym for DBMS_APPLICATION_INFO is not dropped before creation so that you can redirect the public synonym to point to your own package.

19.3 DBMS_APPLICATION_INFO Operational Notes

Your applications should set the name of the module and name of the action automatically each time a user enters that module. The module name could be the name of a form in an Oracle Forms application, or the name of the code segment in an Oracle Precompilers application. The action name should usually be the name or description of the current transaction within a module.

If you want to gather your own statistics based on module, you can implement a wrapper around this package by writing a version of this package in another schema that first gathers statistics and then calls the SYS version of the package. The public synonym for DBMS_APPLICATION_INFO can then be changed to point to the DBA's version of the package.
19.4 Summary of DBMS_APPLICATION_INFO Subprograms

This table lists and describes the DBMS_APPLICATION_INFO package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_CLIENT_INFO Procedure</td>
<td>Reads the value of the client_info field of the current session</td>
</tr>
<tr>
<td>READ_MODULE Procedure</td>
<td>Reads the values of the module and action fields of the current session</td>
</tr>
<tr>
<td>SET_ACTION Procedure</td>
<td>Sets the name of the current action within the current module</td>
</tr>
<tr>
<td>SET_CLIENT_INFO Procedure</td>
<td>Sets the client_info field of the session</td>
</tr>
<tr>
<td>SET_MODULE Procedure</td>
<td>Sets the name of the module that is currently running to a new module</td>
</tr>
<tr>
<td>SET_SESSION_LONGOPS Procedure</td>
<td>Sets a row in the V$SESSION_LONGOPS table</td>
</tr>
</tbody>
</table>

19.4.1 READ_CLIENT_INFO Procedure

This procedure reads the value of the client_info field of the current session.

Syntax

DBMS_APPLICATION_INFO.READ_CLIENT_INFO (client_info OUT VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_info</td>
<td>Last client information value supplied to the SET_CLIENT_INFO procedure.</td>
</tr>
</tbody>
</table>

19.4.2 READ_MODULE Procedure

This procedure reads the values of the module and action fields of the current session.

Syntax

DBMS_APPLICATION_INFO.READ_MODULE (module_name OUT VARCHAR2, action_name OUT VARCHAR2);
Parameters

Table 19-3  READ_MODULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module_name</td>
<td>Last value that the module name was set to by calling SET_MODULE.</td>
</tr>
<tr>
<td>action_name</td>
<td>Last value that the action name was set to by calling SET_ACTION or SET_MODULE.</td>
</tr>
</tbody>
</table>

Usage Notes

Module and action names for a registered application can be retrieved by querying V$SQLAREA or by calling the READ_MODULE procedure. Client information can be retrieved by querying the V$SESSION view, or by calling the READ_CLIENT_INFO Procedure.

Examples

The following sample query illustrates the use of the MODULE and ACTION column of the V$SQLAREA.

```sql
SELECT sql_text, disk_reads, module, action
FROM v$sqlarea
WHERE module = 'add_employee';
```

<table>
<thead>
<tr>
<th>SQL_TEXT DISK_READS MODULE ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>INSERT INTO emp 1 add_employee insert into emp</td>
</tr>
<tr>
<td>(ename, empno, sal, mgr, job, hiredate, comm, deptno)</td>
</tr>
<tr>
<td>VALUES</td>
</tr>
<tr>
<td>(name, next.emp_seq, manager, title, SYSDATE, commission, department)</td>
</tr>
<tr>
<td>1 row selected.</td>
</tr>
</tbody>
</table>

19.4.3 SET_ACTION Procedure

This procedure sets the name of the current action within the current module.

Syntax

```sql
DBMS_APPLICATION_INFO.SET_ACTION ( action_name IN VARCHAR2);
```

Parameters

Table 19-4  SET_ACTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action_name</td>
<td>The name of the current action within the current module. When the current action terminates, call this procedure with the name of the next action if there is one, or NULL if there is not. Names longer than 32 bytes are truncated.</td>
</tr>
</tbody>
</table>
Usage Notes

The action name should be descriptive text about the current action being performed. You should probably set the action name before the start of every transaction.

Set the transaction name to NULL after the transaction completes, so that subsequent transactions are logged correctly. If you do not set the transaction name to NULL, subsequent transactions may be logged with the previous transaction's name.

Example

The following is an example of a transaction that uses the registration procedure:

```sql
CREATE OR REPLACE PROCEDURE bal_tran (amt IN NUMBER(7,2)) AS
BEGIN
  -- balance transfer transaction
  DBMS_APPLICATION_INFO.SET_ACTION(
    action_name => 'transfer from chk to sav');
  UPDATE chk SET bal = bal + :amt
    WHERE acct# = :acct;
  UPDATE sav SET bal = bal - :amt
    WHERE acct# = :acct;
  COMMIT;
  DBMS_APPLICATION_INFO.SET_ACTION(null);
END;
```

19.4.4 SET_CLIENT_INFO Procedure

This procedure supplies additional information about the client application.

Syntax

```sql
DBMS_APPLICATION_INFO.SET_CLIENT_INFO (
  client_info IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_info</td>
<td>Supplies any additional information about the client application. This information is stored in the V$SESSION view. Information exceeding 64 bytes is truncated.</td>
</tr>
</tbody>
</table>

Note:

CLIENT_INFO is readable and writable by any user. For storing secured application attributes, you can use the application context feature.
19.4.5 SET_MODULE Procedure

This procedure sets the name of the current application or module.

Syntax

```
DBMS_APPLICATION_INFO.SET_MODULE (
    module_name IN VARCHAR2,
    action_name IN VARCHAR2);
```

Parameters

Table 19-6  SET_MODULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module_name</td>
<td>Name of module that is currently running. When the current module terminates, call this procedure with the name of the new module if there is one, or NULL if there is not. Names longer than 48 bytes are truncated.</td>
</tr>
<tr>
<td>action_name</td>
<td>Name of current action within the current module. If you do not want to specify an action, this value should be NULL. Names longer than 32 bytes are truncated.</td>
</tr>
</tbody>
</table>

Usage Notes

Example

```
CREATE or replace PROCEDURE add_employee(  
    name VARCHAR2,  
    salary NUMBER,  
    manager NUMBER,  
    title VARCHAR2,  
    commission NUMBER,  
    department NUMBER) AS  
BEGIN  
    DBMS_APPLICATION_INFO.SET_MODULE(  
        module_name => 'add_employee',  
        action_name => 'insert into emp');  
    INSERT INTO emp  
    (ename, empno, sal, mgr, job, hiredate, comm, deptno)  
    VALUES (name, emp_seq.nextval, salary, manager, title, SYSDATE,  
            commission, department);  
    DBMS_APPLICATION_INFO.SET_MODULE(null,null);  
END;
```

19.4.6 SET_SESSION_LONGOPS Procedure

This procedure sets a row in the V$SESSION_LONGOPS view. This is a view that is used to indicate the on-going progress of a long running operation. Some Oracle functions, such as
parallel execution and Server Managed Recovery, use rows in this view to indicate the status of, for example, a database backup.

Applications may use the `SET_SESSION_LONGOPS` procedure to advertise information on the progress of application specific long running tasks so that the progress can be monitored by way of the `V$SESSION_LONGOPS` view.

**Syntax**

```sql
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS (  
    rindex      IN OUT BINARY_INTEGER,  
    slno        IN OUT BINARY_INTEGER,  
    op_name     IN     VARCHAR2       DEFAULT NULL,  
    target      IN     BINARY_INTEGER DEFAULT 0,  
    context     IN     BINARY_INTEGER DEFAULT 0,  
   sofar        IN     NUMBER         DEFAULT 0,  
    totalwork   IN     NUMBER         DEFAULT 0,  
    target_desc IN     VARCHAR2       DEFAULT 'unknown target',  
    units       IN     VARCHAR2       DEFAULT NULL)
```

```sql
set_session_longops_nohint constant BINARY_INTEGER := -1;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rindex</td>
<td>A token which represents the <code>v$session_longops</code> row to update. Set this to <code>set_session_longops_nohint</code> to start a new row. Use the returned value from the prior call to reuse a row.</td>
</tr>
<tr>
<td>slno</td>
<td>Saves information across calls to <code>set_session_longops</code>: It is for internal use and should not be modified by the caller.</td>
</tr>
<tr>
<td>op_name</td>
<td>Specifies the name of the long running task. It appears as the OPNAME column of <code>v$session_longops</code>. The maximum length is 64 bytes.</td>
</tr>
<tr>
<td>target</td>
<td>Specifies the object that is being worked on during the long running operation. For example, it could be a table ID that is being sorted. It appears as the TARGET column of <code>v$session_longops</code>.</td>
</tr>
<tr>
<td>context</td>
<td>Any number the client wants to store. It appears in the CONTEXT column of <code>v$session_longops</code>.</td>
</tr>
<tr>
<td>sofar</td>
<td>Any number the client wants to store. It appears in the SOFAR column of <code>v$session_longops</code>. This is typically the amount of work which has been done so far.</td>
</tr>
<tr>
<td>totalwork</td>
<td>Any number the client wants to store. It appears in the TOTALWORK column of <code>v$session_longops</code>. This is typically an estimate of the total amount of work needed to be done in this long running operation.</td>
</tr>
<tr>
<td>target_desc</td>
<td>Specifies the description of the object being manipulated in this long operation. This provides a caption for the target parameter. This value appears in the TARGET_DESC field of <code>v$session_longops</code>. The maximum length is 32 bytes.</td>
</tr>
</tbody>
</table>
Table 19-7  (Cont.) SET_SESSION_LONGOPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>units</td>
<td>Specifies the units in which sofar and totalwork are being represented. It appears as the UNITS field of v$session_longops. The maximum length is 32 bytes.</td>
</tr>
</tbody>
</table>

Example

This example performs a task on 10 objects in a loop. As the example completes each object, Oracle updates V$SESSION_LONGOPS on the procedure’s progress.

```sql
DECLARE
  rindex    BINARY_INTEGER;
  slno      BINARY_INTEGER;
  totalwork number;
  sofar     number;
  obj       BINARY_INTEGER;
BEGIN
  rindex := dbms_application_info.set_session_longops_nohint;
  sofar := 0;
  totalwork := 10;
  WHILE sofar < 10 LOOP
    -- update obj based on sofar
    -- perform task on object target
    sofar := sofar + 1;
    dbms_application_info.set_session_longops(rindex, slno, "Operation X", obj, 0, sofar, totalwork, "table", "tables");
  END LOOP;
END;
```
The **DBMS_APPLY_ADM** package provides subprograms to configure and manage Oracle Streams apply processes, XStream outbound servers, and XStream inbound servers.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Summary of DBMS_APPLY_ADM Subprograms

**See Also:**

- *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator’s Guide* for more information about this package and apply processes
- *Oracle Database XStream Guide* for more information about XStream outbound servers and inbound servers

### 20.1 DBMS_APPLY_ADM Overview

The **DBMS_APPLY_ADM** package provides interfaces to start, stop, and configure Oracle Streams apply processes, XStream outbound servers, and XStream inbound servers.

This package includes subprograms for configuring apply handlers, setting enqueue destinations for messages, and specifying execution directives for messages. This package also provides administrative subprograms that set the instantiation SCN for objects at a destination database. This package also includes subprograms for managing apply errors.

XStream inbound servers and outbound servers can be used in an XStream configuration in a multitenant container database (CDB). A CDB is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs).

**Note:**

- For simplicity, this chapter refers to apply processes, XStream outbound servers, and XStream inbound servers as **apply components**. This chapter identifies a specific type of apply component when necessary.
- Using XStream requires purchasing a license for the Oracle GoldenGate product.
20.2 DBMS_APPLY_ADM Security Model

Security on this package can be controlled by either granting EXECUTE on this package to selected users or roles, or by granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

When the DBMS_APPLY_ADM package is used to manage an Oracle Streams configuration, it requires that the user is granted the privileges of an Oracle Streams administrator.

When the DBMS_APPLY_ADM package is used to manage an XStream configuration, it requires that the user is granted the privileges of an XStream administrator.

Note:
The user must be granted additional privileges to perform some administrative tasks using the subprograms in this package, such as setting an apply user. If additional privileges are required for a subprogram, then the privileges are documented in the section that describes the subprogram.

See Also:
- Oracle Streams Concepts and Administration for information about configuring an Oracle Streams administrator
- Oracle Database XStream Guide for information about configuring an XStream administrator
20.3 DBMS\_APPLY\_ADM Deprecated Subprograms

The `NONE` value for the `commit_serialization` apply component parameter is deprecated. It is replaced by the `DEPENDENT\_TRANSACTIONS` value.

**Note:**

Oracle recommends that you do not use deprecated apply component parameter values. Support for deprecated features is for backward compatibility only.

**See Also:**

- SET\_PARAMETER Procedure

20.4 Summary of DBMS\_APPLY\_ADM Subprograms

This table lists and describes the `DBMS\_APPLY\_ADM` subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_STMT_HANDLER Procedure</td>
<td>Adds a statement DML handler for a specified operation on a specified database object to a single apply component or to all apply components in the database</td>
</tr>
<tr>
<td>ALTER_APPLY Procedure</td>
<td>Alters an apply component</td>
</tr>
<tr>
<td>COMPARE_OLD_VALUES Procedure</td>
<td>Specifies whether to compare the old value of one or more columns in a row logical change record (row LCR) with the current value of the corresponding columns at the destination site during apply</td>
</tr>
<tr>
<td>CREATE_APPLY Procedure</td>
<td>Creates an apply component</td>
</tr>
<tr>
<td>CREATE_OBJECT_DEPENDENCY Procedure</td>
<td>Creates an object dependency</td>
</tr>
<tr>
<td>DELETE_ALL_ERRORS Procedure</td>
<td>Deletes all the error transactions for the specified apply component</td>
</tr>
<tr>
<td>DELETE_ERROR Procedure</td>
<td>Deletes the specified error transaction</td>
</tr>
<tr>
<td>DROP_APPLY Procedure</td>
<td>Drops an apply component</td>
</tr>
<tr>
<td>DROP_OBJECT_DEPENDENCY Procedure</td>
<td>Drops an object dependency</td>
</tr>
<tr>
<td>EXECUTE_ALL_ERRORS Procedure</td>
<td>Reexecutes the error transactions for the specified apply component</td>
</tr>
<tr>
<td>EXECUTE_ERROR Procedure</td>
<td>Reexecutes the specified error transaction</td>
</tr>
<tr>
<td>GET_ERROR_MESSAGE Function</td>
<td>Returns the message payload from the error queue for the specified message number and transaction identifier</td>
</tr>
</tbody>
</table>
Table 20-1  (Cont.) DBMS_APPLY_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HANDLE_COLLISIONS</td>
<td>Enables or disables basic conflict resolution for an apply process and a table</td>
</tr>
<tr>
<td>REMOVE_STMT_HANDLER</td>
<td>Remoes a statement DML handler for a specified operation on a specified database object from a single apply component or from all apply components in the database</td>
</tr>
<tr>
<td>SET_CHANGE_HANDLER Procedure</td>
<td>Sets or unsets a statement DML handler that tracks changes for a specified operation on a specified database object for a single apply component</td>
</tr>
<tr>
<td>SET_DML_CONFLICT_HANDLER Procedure</td>
<td>Adds, modifies, or removes a prebuilt DML conflict handler for INSERT, UPDATE, or DELETE conflicts on the specified object</td>
</tr>
<tr>
<td>SET_DML_HANDLER Procedure</td>
<td>Sets a user procedure as a procedure DML handler for a specified operation on a specified database object for a single apply component or for all apply components in the database</td>
</tr>
<tr>
<td>SET_ENQUEUE_DESTINATION Procedure</td>
<td>Sets the queue where the apply component automatically enqueues a message that satisfies the specified rule</td>
</tr>
<tr>
<td>SET_EXECUTE Procedure</td>
<td>Specifies whether a message that satisfies the specified rule is executed by an apply component</td>
</tr>
<tr>
<td>SET_GLOBAL_INSTANTIATION_SCN Procedure</td>
<td>Records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas</td>
</tr>
<tr>
<td>SET_KEY_COLUMNS Procedures</td>
<td>Records the set of columns to be used as the substitute primary key for local apply purposes and removes existing substitute primary key columns for the specified object if they exist</td>
</tr>
<tr>
<td>SET_PARAMETER Procedure</td>
<td>Sets an apply parameter to the specified value</td>
</tr>
<tr>
<td>SET_REPERROR_HANDLER Procedure</td>
<td>Specifies how a particular error is handled based on its error number</td>
</tr>
<tr>
<td>SET_SCHEMA_INSTANTIATION_SCN Procedure</td>
<td>Records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database</td>
</tr>
<tr>
<td>SET_TABLE_INSTANTIATION_SCN Procedure</td>
<td>Records the specified instantiation SCN for the specified table in the specified source database and, optionally, for the tables owned by the schema at the source database</td>
</tr>
<tr>
<td>SET_UPDATE_CONFLICT_HANDLER Procedure</td>
<td>Adds, updates, or drops an update conflict handler for the specified object</td>
</tr>
<tr>
<td>SET_VALUE_DEPENDENCY Procedure</td>
<td>Sets or removes a value dependency</td>
</tr>
<tr>
<td>START_APPLY Procedure</td>
<td>Directs the apply component to start applying messages</td>
</tr>
<tr>
<td>STOP_APPLY Procedure</td>
<td>Stops the apply component from applying any messages and rolls back any unfinished transactions being applied</td>
</tr>
</tbody>
</table>
20.4.1 ADD_STMT_HANDLER Procedure

This procedure adds a statement DML handler for a specified operation on a specified database object. The procedure adds the statement DML handler to a single apply component or to all apply components in the database.

This procedure is overloaded. One version of this procedure contains the statement and comment parameters, and the other does not. The statement parameter enables you to create the statement DML handler and add it to one or more processes in one step. Otherwise, create the statement DML handler using the DBMS_STREAMS_HANDLER_ADM package before adding it to one or more processes.

See Also:

- DBMS_STREAMS_HANDLER_ADM
- Oracle Streams Concepts and Administration

Syntax

```
DBMS_APPLY_ADM.ADD_STMT_HANDLER(
  object_name     IN  VARCHAR2,
  operation_name  IN  VARCHAR2,
  handler_name    IN  VARCHAR2,
  statement       IN  CLOB,
  apply_name      IN  VARCHAR2  DEFAULT NULL,
  comment         IN  VARCHAR2  DEFAULT NULL);
```

```
DBMS_APPLY_ADM.ADD_STMT_HANDLER(
  object_name     IN  VARCHAR2,
  operation_name  IN  VARCHAR2,
  handler_name    IN  VARCHAR2,
  apply_name      IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 20-2  ADD_STMT_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the source object specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>operation_name</td>
<td>The name of the operation, which can be specified as:</td>
</tr>
<tr>
<td></td>
<td>· INSERT</td>
</tr>
<tr>
<td></td>
<td>· UPDATE</td>
</tr>
<tr>
<td></td>
<td>· DELETE</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>Note:</td>
<td>Statement DML handlers cannot be specified for LOB operations.</td>
</tr>
<tr>
<td>handler_name</td>
<td>The name of the statement DML handler.</td>
</tr>
<tr>
<td></td>
<td>If the specified statement DML handler exists, then the statement in the statement parameter is added to the existing handler. Ensure that the existing statement DML handler is for the same operation on the same database object as the settings for the operation_name and object_name parameters, respectively. If the existing handler is for a different operation or database object, then an apply error results when the handler is invoked.</td>
</tr>
<tr>
<td></td>
<td>If the specified statement DML handler does not exist and a non-NULL statement parameter is specified, then this procedure creates the statement DML handler.</td>
</tr>
<tr>
<td></td>
<td>If the specified statement DML handler does not exist and the statement parameter is not specified or is NULL, then this procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>statement</td>
<td>The text of the SQL statement to add to the statement DML handler.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of the apply component that uses the statement DML handler.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure adds the statement DML handler as a general handler to all apply components in the database.</td>
</tr>
<tr>
<td>comment</td>
<td>A comment for the statement DML handler.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then no comment is recorded for the statement DML handler.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The following usage notes apply to this procedure:

- The ADD_STMT_HANDLER Procedure and XStream Outbound Servers
- The ADD_STMT_HANDLER Procedure and XStream Inbound Servers

The ADD_STMT_HANDLER Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers. Outbound servers ignore all apply handlers.

The ADD_STMT_HANDLER Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.
20.4.2 ALTER_APPLY Procedure

This procedure alters an apply component.

Syntax

```sql
DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name                IN  VARCHAR2,
    rule_set_name             IN  VARCHAR2  DEFAULT NULL,
    remove_rule_set           IN  BOOLEAN   DEFAULT FALSE,
    message_handler           IN  VARCHAR2  DEFAULT NULL,
    remove_message_handler    IN  BOOLEAN   DEFAULT FALSE,
    ddl_handler               IN  VARCHAR2  DEFAULT NULL,
    remove_ddl_handler        IN  BOOLEAN   DEFAULT FALSE,
    apply_user                IN  VARCHAR2  DEFAULT NULL,
    apply_tag                 IN  RAW       DEFAULT NULL,
    remove_apply_tag          IN  BOOLEAN   DEFAULT FALSE,
    precommit_handler         IN  VARCHAR2  DEFAULT NULL,
    remove_precommit_handler  IN  BOOLEAN   DEFAULT FALSE,
    negative_rule_set_name    IN  VARCHAR2  DEFAULT NULL,
    remove_negative_rule_set  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 20-3 ALTER_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The name of the apply component being altered. You must specify the name of an existing apply component. Do not specify an owner.</td>
</tr>
</tbody>
</table>
| rule_set_name           | The name of the positive rule set for the apply component. The positive rule set contains the rules that instruct the apply component to apply messages.

If you want to use a positive rule set for the apply component, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the hr schema named job_apply_rules, enter hr.job_apply_rules. If the schema is not specified, then the current user is the default.

An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.

If you specify NULL and the remove_rule_set parameter is set to FALSE, then this procedure retains any existing positive rule set for the specified apply component. If you specify NULL and the remove_rule_set parameter is set to TRUE, then this procedure removes any existing positive rule set from the specified apply component.
Table 20-3  (Cont.) ALTER_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove_rule_set</td>
<td>If TRUE, then the procedure removes the positive rule set for the specified apply component. If you remove the positive rule set for an apply component, and the apply component does not have a negative rule set, then the apply component dequeues all messages in its queue.  If you remove the positive rule set for an apply component, and a negative rule set exists for the apply component, then the apply component dequeues all messages in its queue that are not discarded by the negative rule set.  If FALSE, then the procedure retains the positive rule set for the specified apply component.  If the rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
<tr>
<td>message_handler</td>
<td>A user-defined procedure that processes non-LCR messages in the queue for the apply component.  See &quot;Usage Notes&quot; in the CREATE_APPLY Procedure for more information about a message handler procedure.</td>
</tr>
<tr>
<td>remove_message_handler</td>
<td>If TRUE, then the procedure removes the message handler for the specified apply component.  If FALSE, then the procedure retains any message handler for the specified apply component.  If the message_handler parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
<tr>
<td>ddl_handler</td>
<td>A user-defined procedure that processes DDL logical change records (DDL LCRs) in the queue for the apply component.  All applied DDL LCRs commit automatically.  Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically.  See &quot;Usage Notes&quot; in the CREATE_APPLY Procedure for more information about a DDL handler procedure.</td>
</tr>
<tr>
<td>remove_ddl_handler</td>
<td>If TRUE, then the procedure removes the DDL handler for the specified apply component.  If FALSE, then the procedure retains any DDL handler for the specified apply component.  If the ddl_handler parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
</tbody>
</table>
Table 20-3  (Cont.) ALTER_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| apply_user  | The user in whose security domain an apply component dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations, and runs apply handlers. If NULL, then the apply user is not changed.

If a non-NULL value is specified to change the apply user, then the user who invokes the ALTER_APPLY procedure must be granted the DBA role. Only the SYS user can set the apply_user to SYS.

If you change the apply user, then this procedure grants the new apply user dequeue privilege on the queue used by the apply component. It also configures the user as a secure queue user of the queue.

In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user:

- The necessary privileges to perform DML and DDL changes on the apply objects
- EXECUTE privilege on the rule sets used by the apply component
- EXECUTE privilege on all rule-based transformation functions used in the rule set
- EXECUTE privilege on all apply handler procedures

These privileges can be granted directly to the apply user, or they can be granted through roles.

In addition, the apply user must be granted the EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply component. These privileges must be granted directly to the apply user. They cannot be granted through roles.

By default, this parameter is set to the user who created the apply component by running either the CREATE_APPLY procedure in this package or a procedure in the DBMS_STREAMS_ADM package.

**Note:** If the apply user for an apply component is dropped using DROP USER . . . CASCADE, then the apply component is also dropped automatically.
### Table 20-3 (Cont.) ALTER_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| apply_tag          | A binary tag that is added to redo entries generated by the specified apply component. The tag is a binary value that can be used to track LCRs. The tag is relevant only if a capture process at the database where the apply component is running captures changes made by the apply component. If so, then the captured changes include the tag specified by this parameter. If NULL, the default, then the apply tag for the apply component is not changed. The following is an example of a tag with a hexadecimal value of 17: HEXTORAW('17')  
  
  See Also: Oracle Streams Replication Administrator’s Guide for more information about tags |
| remove_apply_tag   | If TRUE, then the procedure sets the apply tag for the specified apply component to NULL, and the apply component generates redo entries with NULL tags. If FALSE, then the procedure retains any apply tag for the specified apply component. If the apply_tag parameter is non-NUL, then this parameter should be set to FALSE. |
| precommit_handler  | A user-defined procedure that can receive internal commit directives in the queue for the apply component before they are processed by the apply component. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply component. An internal commit directive is enqueued in the following ways:  
  - When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs.  
  - When a user or application enqueues messages and then issues a COMMIT statement, the commit directive is enqueued automatically.  
  For a captured row LCR, a commit directive contains the commit SCN of the transaction from the source database. For a user message, the commit SCN is generated by the apply component.  
  The precommit handler procedure must conform to the following restrictions:  
  - Any work that commits must be an autonomous transaction.  
  - Any rollback must be to a named savepoint created in the procedure.  
  If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue.  
  See "Usage Notes" in the CREATE_APPLY Procedure for more information about a precommit handler procedure. |

---

**Chapter 20**

Summary of DBMS_APPLY_ADM Subprograms
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove_precommit_handler</td>
<td>If TRUE, then the procedure removes the precommit handler for the specified apply component. If FALSE, then the procedure retains any precommit handler for the specified apply component. If the precommit_handler parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
<tr>
<td>negative_rule_set_name</td>
<td>The name of the negative rule set for the apply component. The negative rule set contains the rules that instruct the apply component to discard messages. If you want to use a negative rule set for the apply component, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a negative rule set in the hr schema named neg_apply_rules, enter hr.neg_apply_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL and the remove_negative_rule_set parameter is set to FALSE, then the procedure retains any existing negative rule set. If you specify NULL and the remove_negative_rule_set parameter is set to TRUE, then the procedure removes any existing negative rule set. If you specify both a positive and a negative rule set for an apply component, then the negative rule set is always evaluated first.</td>
</tr>
<tr>
<td>remove_negative_rule_set</td>
<td>If TRUE, then the procedure removes the negative rule set for the specified apply component. If you remove the negative rule set for an apply component, and the apply component does not have a positive rule set, then the apply component dequeues all messages in its queue. If you remove the negative rule set for an apply component, and a positive rule set exists for the apply component, then the apply component dequeues all messages in its queue that are not discarded by the positive rule set. If FALSE, then the procedure retains the negative rule set for the specified apply component. If the negative_rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The following usage notes apply to this procedure:

- **Automatic Restart of Apply Components**
- **The ALTER_APPLY Procedure and XStream Outbound Servers**
- **The ALTER_APPLY Procedure and XStream Inbound Servers**

Automatic Restart of Apply Components
An apply component is stopped and restarted automatically when you change the value of one or more of the following **ALTER_APPLY** procedure parameters:

- message_handler
- ddl_handler
- apply_user
- apply_tag
- precommit_handler

The **ALTER_APPLY** Procedure and XStream Outbound Servers

The following usage notes apply to this procedure and XStream outbound servers:

- The **apply_user** parameter can change the connect user for an outbound server.
- You cannot specify an apply handler for an outbound server. An outbound server ignores the settings for the following parameters: **message_handler**, **ddl_handler**, and **precommit_handler**.

  The client application can perform custom processing of the LCRs instead if necessary.

- An outbound server cannot set an apply tag for the changes it processes. An outbound server ignores the setting for the **apply_tag** parameter.

The **ALTER_APPLY** Procedure and XStream Inbound Servers

Inbound servers can use apply handlers and process only DML and DDL LCRs. Therefore, inbound servers ignore message handlers specified in the **message_handler** parameter.

### 20.4.3 COMPARE_OLD_VALUES Procedure

This procedure specifies whether to compare the old values of one or more columns in a row logical change record (row LCR) with the current values of the corresponding columns at the destination site during apply.

This procedure is relevant only for **UPDATE** and **DELETE** operations because only these operations result in old column values in row LCRs. The default is to compare old values for all columns.

This procedure is overloaded. The **column_list** and **column_table** parameters are mutually exclusive.

#### See Also:

*Oracle Streams Replication Administrator's Guide* for more information about conflict detection and resolution in an Oracle Streams environment

#### Syntax

```sql
DBMS_APPLY_ADM.COMpare_OLD_VALUES(
    object_name       IN VARCHAR2,
)```
DBMS_APPLY_ADM.COMpare_OLD_VALUES(
    object_name IN VARCHAR2,
    column_table IN DBMS_UTILITY.LNAME_ARRAY,
    operation IN VARCHAR2 DEFAULT 'UPDATE',
    compare IN BOOLEAN  DEFAULT TRUE,
    apply_database_link IN VARCHAR2 DEFAULT NULL);

Parameters

Table 20-4 COMPARE_OLD_VALUES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the source table specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-delimited list of column names in the table. There must be no spaces between entries. Specify * to include all nonkey columns.</td>
</tr>
<tr>
<td>column_table</td>
<td>A PL/SQL associative array of type DBMS_UTILITY.LNAME_ARRAY that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be NULL terminated.</td>
</tr>
<tr>
<td>operation</td>
<td>The name of the operation, which can be specified as:</td>
</tr>
<tr>
<td></td>
<td>• UPDATE for UPDATE operations</td>
</tr>
<tr>
<td></td>
<td>• DELETE for DELETE operations</td>
</tr>
<tr>
<td></td>
<td>• * for both UPDATE and DELETE operations</td>
</tr>
<tr>
<td>compare</td>
<td>If compare is TRUE, the old values of the specified columns are compared during apply. If compare is FALSE, the old values of the specified columns are not compared during apply.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- Conflict Detection
- The COMPARE_OLD_VALUES Procedure and XStream Outbound Servers
- The COMPARE_OLD_VALUES Procedure and XStream Inbound Servers

Conflict Detection

By default, an apply component uses the old column values in a row LCR to detect conflicts. You can choose not to compare old column values to avoid conflict detection for specific tables. For example, if you do not want to compare the old values for a set of columns during apply, then, using the COMPARE_OLD_VALUES procedure, specify the set of columns in the column_list or column_table parameter, and set the compare parameter to FALSE.
In addition, when the *compare_key_only* apply component parameter is set to *Y*, automatic conflict detection is disabled, and the apply component only uses primary key and unique key columns to identify the table row for a row LCR. When the *compare_key_only* apply component parameter is set to *N*, automatic conflict detection is enabled, and the apply component uses all of the old values in a row LCR to identify the table row for a row LCR.

**Note:**

- An apply component compares old values for non-key columns when they are present in a row LCR and when the apply component parameter *compare_key_only* is set to *N*.
- This procedure raises an error if a key column is specified in *column_list* or *column_table* and the *compare* parameter is set to *FALSE*.

**See Also:**

- [SET_PARAMETER Procedure](#) for more information about the *compare_key_only* apply component parameter.

The COMPARE_OLD_VALUES Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The COMPARE_OLD_VALUES Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

### 20.4.4 CREATE_APPLY Procedure

This procedure creates an apply component.

**Syntax**

```
DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name IN VARCHAR2,
    apply_name IN VARCHAR2,
    rule_set_name IN VARCHAR2 DEFAULT NULL,
    message_handler IN VARCHAR2 DEFAULT NULL,
    ddl_handler IN VARCHAR2 DEFAULT NULL,
    apply_user IN VARCHAR2 DEFAULT NULL,
    apply_database_link IN VARCHAR2 DEFAULT NULL,
    apply_tag IN RAW DEFAULT '00',
    apply_captured IN BOOLEAN DEFAULT FALSE,
    precommit_handler IN VARCHAR2 DEFAULT NULL,
    negative_rule_set_name IN VARCHAR2 DEFAULT NULL,
    source_database IN VARCHAR2 DEFAULT NULL);
```
## Parameters

### Table 20-5  CREATE_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the queue from which the apply component dequeues messages. You must specify an existing queue in the form [schema_name.]queue_name. For example, to specify a queue in the hr schema named streams_queue, enter hr.streams_queue. If the schema is not specified, then the current user is the default.  <strong>Note:</strong> The queue_name setting cannot be altered after the apply component is created.</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of the apply component being created. A NULL specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing apply component or messaging client.  <strong>Note:</strong> The apply_name setting cannot be altered after the apply component is created.</td>
</tr>
<tr>
<td>rule_set_name</td>
<td>The name of the positive rule set for the apply component. The positive rule set contains the rules that instruct the apply component to apply messages.  If you want to use a positive rule set for the apply component, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the hr schema named job_apply_rules, enter hr.job_apply_rules. If the schema is not specified, then the current user is the default.  If you specify NULL, and no negative rule set is specified, then the apply component applies either all captured messages or all messages in the persistent queue, depending on the setting of the apply_captured parameter. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</td>
</tr>
<tr>
<td>message_handler</td>
<td>A user-defined procedure that processes non-LCR messages in the queue for the apply component. See &quot;Usage Notes&quot; for more information about a message handler procedure.</td>
</tr>
<tr>
<td>ddl_handler</td>
<td>A user-defined procedure that processes DDL logical change record (DDL LCRs) in the queue for the apply component. All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically. See &quot;Usage Notes&quot; for more information about a DDL handler procedure.</td>
</tr>
</tbody>
</table>
Table 20-5  (Cont.) CREATE_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| apply_user            | The user who applies all DML and DDL changes that satisfy the apply component rule sets and who runs user-defined apply handlers. If NULL, then the user who runs the CREATE_APPLY procedure is used. The apply user is the user in whose security domain an apply component dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations configured for apply component rules, and runs apply handlers configured for the apply component. This user must have the necessary privileges to apply changes. This procedure grants the apply user dequeue privilege on the queue used by the apply component and configures the user as a secure queue user of the queue. In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user:  
  • The necessary privileges to perform DML and DDL changes on the apply objects  
  • EXECUTE privilege on the rule sets used by the apply component  
  • EXECUTE privilege on all rule-based transformation functions used in the rule set  
  • EXECUTE privilege on all apply handler procedures  
  These privileges can be granted directly to the apply user, or they can be granted through roles. In addition, the apply user must be granted EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply component. These privileges must be granted directly to the apply user. They cannot be granted through roles. You can use the packages DBMS STREAMS AUTH and DBMS XSTREAM AUTH to grant and revoke administrative privileges in Oracle Streams and XStream configurations, respectively. These packages do not configure the necessary privileges to perform DML or DDL changes on the apply objects. Note: If the apply user for an apply component is dropped using DROP USER . . . CASCADE, then the apply component is also dropped automatically. See "Usage Notes" for more information about this parameter. |
| apply_database_link   | The database at which the apply component applies messages. This parameter is used by an apply component when applying changes from Oracle to non-Oracle systems, such as Sybase. Set this parameter to NULL to specify that the apply component applies messages at the local database. Note: The apply_database_link setting cannot be altered after the apply component is created. |
### Table 20-5  (Cont.) CREATE_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_tag</td>
<td>A binary tag that is added to redo entries generated by the specified apply component. The tag is a binary value that can be used to track LCRs.</td>
</tr>
<tr>
<td></td>
<td>The tag is relevant only if a capture process at the database where the apply component is running captures changes made by the apply component. If so, then the captured changes include the tag specified by this parameter.</td>
</tr>
<tr>
<td></td>
<td>By default, the tag for an apply component is the hexadecimal equivalent of 00 (double zero).</td>
</tr>
<tr>
<td></td>
<td>The following is an example of a tag with a hexadecimal value of 17:</td>
</tr>
<tr>
<td></td>
<td>HEXTORAW('17')</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the apply component generates redo entries with NULL tags.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also</strong>: Oracle Streams Replication Administrator's Guide for more information about tags</td>
</tr>
<tr>
<td>apply_captured</td>
<td>Either TRUE or FALSE.</td>
</tr>
<tr>
<td></td>
<td>If TRUE, then the apply component applies only the captured LCRs in the queue. Captured LCRs are LCRs that were captured by an Oracle Streams capture process.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the apply component applies only the messages in a persistent queue. These are messages that were not captured by an Oracle Streams capture process, such as persistent LCRs or user messages.</td>
</tr>
<tr>
<td></td>
<td>To apply both captured LCRs and messages in a persistent queue, you must create at least two apply components.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: The apply_captured setting cannot be altered after the apply component is created.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also</strong>: Oracle Streams Concepts and Administration for more information about processing messages with an apply component</td>
</tr>
</tbody>
</table>
Table 20-5 (Cont.) CREATE_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| precommit_handler    | A user-defined procedure that can receive internal commit directives in the queue for the apply component before they are processed by the apply component. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply component. An internal commit directive is enqueued in the following ways:  
  • When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs.  
  • When a synchronous capture captures row LCRs, the persistent LCRs that were enqueued by the synchronous capture are organized into a message group. The synchronous capture records the transaction identifier in each persistent LCR in a transaction.  
  • When a user or application enqueues messages and then issues a COMMIT statement, the commit directive is enqueued automatically.  
For a row LCR captured by a capture process or synchronous capture, a commit directive contains the commit SCN of the transaction from the source database. For a message enqueued by a user or application, the commit SCN is generated by the apply component.  
The precommit handler procedure must conform to the following restrictions:  
  • Any work that commits must be an autonomous transaction.  
  • Any rollback must be to a named savepoint created in the procedure.  
If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue.  
See "Usage Notes" for more information about a precommit handler procedure. |
| negative_rule_set_name | The name of the negative rule set for the apply component. The negative rule set contains the rules that instruct the apply component to discard messages.  
If you want to use a negative rule set for the apply component, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a negative rule set in the hr schema named neg_apply_rules, enter hr.neg_apply_rules. If the schema is not specified, then the current user is the default.  
If you specify NULL, and no positive rule set is specified, then the apply component applies either all captured LCRs or all of the messages in the persistent queue, depending on the setting of the apply_captured parameter.  
An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.  
If you specify both a positive and a negative rule set for an apply component, then the negative rule set is always evaluated first. |
**Table 20-5 (Cont.) CREATE_APPLY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database for the changes that will be applied by the apply component. The source database is the database where the changes originated. If an apply component applies captured messages, then the apply component can apply messages from only one capture process at one source database. If NULL, then the source database name of the first LCR received by the apply component is used for the source database. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is NET, then the procedure specifies DBS1.NET automatically. The rules in the apply component rule sets determine which messages are dequeued by the apply component. If the apply component dequeues an LCR with a source database that is different than the source database for the apply component, then an error is raised. You can determine the source database for an apply component by querying the DBA_APPLY_PROGRESS data dictionary view.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The following sections describe usage notes for this procedure:

- DBA Role Requirement
- Handler Procedure Names
- Message Handler and DDL Handler Procedure
- Precommit Handler Procedure
- The CREATE_APPLY Procedure and XStream Outbound Servers
- The CREATE_APPLY Procedure and XStream Inbound Servers

**DBA Role Requirement**

If the user who invokes this procedure is different from the user specified in the apply_user parameter, then the invoking user must be granted the DBA role. If the user who invokes this procedure is the same as the user specified in the apply_user parameter, then the DBA role is not required for the invoking user. Only the SYS user can set the apply_user to SYS.

**Handler Procedure Names**

For the message_handler, ddl_handler, and precommit_handler parameters, specify an existing procedure in one of the following forms:

- [schema_name.]procedure_name
- [schema_name.]package_name.procedure_name

If the procedure is in a package, then the package_name must be specified. For example, to specify a procedure in the apply_pkg package in the hr schema named process_ddls, enter hr.apply_pkg.process_ddls. An error is returned if the specified procedure does not exist.
The user who invokes the `CREATE_APPLY` procedure must have `EXECUTE` privilege on a specified handler procedure. Also, if the `schema_name` is not specified, then the user who invokes the `CREATE_APPLY` procedure is the default.

**Message Handler and DDL Handler Procedure**

The procedure specified in both the `message_handler` parameter and the `ddl_handler` parameter must have the following signature:

```sql
PROCEDURE handler_procedure(
    parameter_name IN ANYDATA);
```

Here, `handler_procedure` stands for the name of the procedure and `parameter_name` stands for the name of the parameter passed to the procedure. For the message handler, the parameter passed to the procedure is a `ANYDATA` encapsulation of a user message. For the DDL handler procedure, the parameter passed to the procedure is a `ANYDATA` encapsulation of a DDL LCR.

**See Also:**

*Logical Change Record TYPEs* for information about DDL LCRs

**Precommit Handler Procedure**

The procedure specified in the `precommit_handler` parameter must have the following signature:

```sql
PROCEDURE handler_procedure(
    parameter_name IN NUMBER);
```

Here, `handler_procedure` stands for the name of the procedure and `parameter_name` stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is the commit SCN of a commit directive.

**The CREATE_APPLY Procedure and XStream Outbound Servers**

This procedure cannot create an XStream outbound server. To create an XStream outbound server, use the `DBMS_XSTREAM_ADM` package.

**The CREATE_APPLY Procedure and XStream Inbound Servers**

The following usage notes apply to this procedure and XStream inbound servers:

- The `CREATE_APPLY` procedure always creates an apply process. The apply process remains an apply process if it receives messages from a source other than an XStream client application, such as a capture process. The apply process can become an inbound server if an XStream client application attaches to it before it receives messages from any other source. After the initial contact, an apply process cannot be changed into an inbound server, and an inbound server cannot be changed into an apply process.

- When creating an inbound server using the `CREATE_APPLY` procedure, set the `apply_captured` parameter to `TRUE`. Inbound servers only process LCRs captured by a capture process.
Inbound servers can use apply handlers. Inbound servers process only DML and DDL LCRs. Therefore, inbound servers ignore message handlers specified in the message_handler parameter.

20.4.5 CREATE_OBJECT_DEPENDENCY Procedure

This procedure creates an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

An apply component schedules execution of transactions that involve the child object after all transactions with a lower commit system change number (commit SCN) that involve the parent object have been committed. An apply component uses the object identifier of the objects in the logical change records (LCRs) to detect dependencies. The apply component does not use column values in the LCRs to detect dependencies.

Note:
An error is raised if NULL is specified for either of the procedure parameters.

See Also:
- DROP_OBJECT_DEPENDENCY Procedure
- Oracle Streams Concepts and Administration

Syntax

```sql
DBMS_APPLY_ADM.CREATE_OBJECT_DEPENDENCY(
    object_name         IN  VARCHAR2,
    parent_object_name  IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the child database object, specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>parent_object_name</td>
<td>The name of the parent database object, specified as [schema_name.]object_name. For example, hr.departments. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- The CREATE_OBJECT_DEPENDENCY Procedure and XStream Outbound Servers
• The CREATE_OBJECT_DEPENDENCY Procedure and XStream Inbound Servers

The CREATE_OBJECT_DEPENDENCY Procedure and XStream Outbound Servers
This procedure has no effect on XStream outbound servers.

The CREATE_OBJECT_DEPENDENCY Procedure and XStream Inbound Servers
This procedure functions the same way for apply processes and inbound servers.

20.4.6 DELETE_ALL_ERRORS Procedure

This procedure deletes all the error transactions for the specified apply component.

Syntax

DBMS_APPLY_ADM.DELETE_ALL_ERRORS(
apply_name  IN  VARCHAR2 DEFAULT NULL);

Parameter

Table 20-7  DELETE_ALL_ERRORS Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The name of the apply component that raised the errors while processing the transactions. Do not specify an owner. If NULL, then all error transactions for all apply components are deleted.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

• The DELETE_ALL_ERRORS Procedure and XStream Outbound Servers
• The DELETE_ALL_ERRORS Procedure and XStream Inbound Servers

The DELETE_ALL_ERRORS Procedure and XStream Outbound Servers
Outbound servers do not enqueue error transactions into an error queue. This procedure has no effect on XStream outbound servers.

The DELETE_ALL_ERRORS Procedure and XStream Inbound Servers
This procedure functions the same way for apply processes and inbound servers.

20.4.7 DELETE_ERROR Procedure

This procedure deletes the specified error transaction.

Syntax

DBMS_APPLY_ADM.DELETE_ERROR(
local_transaction_id  IN  VARCHAR2);
Parameter

Table 20-8 DELETE_ERROR Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_transaction_id</td>
<td>The identification number of the error transaction to delete. If the specified transaction does not exist in the error queue, then an error is raised.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

The DELETE_ERROR Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure has no effect on XStream outbound servers.

The DELETE_ERROR Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

20.4.8 DROP_APPLY Procedure

This procedure drops an apply component.

Syntax

```sql
DBMS_APPLY_ADM.DROP_APPLY(
    apply_name             IN  VARCHAR2,
    drop_unused_rule_sets  IN  BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 20-9 DROP_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The name of the apply component being dropped. You must specify an existing apply component name. Do not specify an owner.</td>
</tr>
<tr>
<td>drop_unused_rule_sets</td>
<td>If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified apply component if these rule sets are not used by any other Oracle Streams or XStream component. These components include capture processes, propagations, apply processes, inbound servers, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set. If FALSE, then the procedure does not drop the rule sets used by the specified apply component, and the rule sets retain their rules.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- The DROP_APPLY Procedure and Rules
• The DROP_APPLY Procedure and XStream Outbound Servers

• The DROP_APPLY Procedure and XStream Inbound Servers

The DROP_APPLY Procedure and Rules

When you use this procedure to drop an apply component, information about rules created for the apply component using the DBMS_STREAMS_ADM package is removed from the data dictionary views for rules. Information about such a rule is removed even if the rule is not in either the positive or negative rule set for the apply component. The following are the data dictionary views for rules:

• ALL_STREAMS_GLOBAL_RULES
• DBA_STREAMS_GLOBAL_RULES
• ALL_STREAMS_MESSAGE_RULES
• DBA_STREAMS_MESSAGE_RULES
• ALL_STREAMS_SCHEMA_RULES
• DBA_STREAMS_SCHEMA_RULES
• ALL_STREAMS_TABLE_RULES
• DBA_STREAMS_TABLE_RULES

See Also:
Oracle Streams Concepts and Administration for more information about Oracle Streams data dictionary views

The DROP_APPLY Procedure and XStream Outbound Servers

When the DROP_APPLY procedure is executed on an outbound server, it runs the DROP_OUTBOUND procedure in the DBMS_XSTREAM_ADM package. Therefore, it might also drop the outbound server's capture process and queue.

The DROP_APPLY Procedure and XStream Inbound Servers

When the DROP_APPLY procedure is executed on an inbound server, it runs the DROP_INBOUND procedure in the DBMS_XSTREAM_ADM package. Therefore, it might also drop the inbound server's queue.
20.4.9 DROP_OBJECT_DEPENDENCY Procedure

This procedure drops an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

Note:

- An error is raised if an object dependency does not exist for the specified database objects.
- An error is raised if NULL is specified for either of the procedure parameters.

See Also:

- CREATE_OBJECT_DEPENDENCY Procedure
- Oracle Streams Concepts and Administration

Syntax

```
DBMS_APPLY_ADM.DROP_OBJECT_DEPENDENCY(
    object_name         IN  VARCHAR2,
    parent_object_name  IN  VARCHAR2);
```

Parameters

Table 20-10  DROP_OBJECT_DEPENDENCY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the child database object, specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>parent_object_name</td>
<td>The name of the parent database object, specified as [schema_name.]object_name. For example, hr.departments. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- The DROP_OBJECT_DEPENDENCY Procedure and XStream Outbound Servers
  This procedure has no effect on XStream outbound servers.
- The DROP_OBJECT_DEPENDENCY Procedure and XStream Inbound Servers
  This procedure functions the same way for apply processes and inbound servers.
20.4.10 EXECUTE_ALL_ERRORS Procedure

This procedure re-executes the error transactions in the error queue for the specified apply component.

The transactions are re-executed in commit SCN order. Error re-execution stops if an error is raised.

See Also:
Oracle Streams Concepts and Administration for more information about the error queue

Syntax

```sql
DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(
    apply_name       IN  VARCHAR2  DEFAULT NULL,
    execute_as_user  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 20-11 EXECUTE_ALL_ERRORS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The name of the apply component that raised the errors while processing the transactions. Do not specify an owner. If NULL, then all error transactions for all apply components are re-executed.</td>
</tr>
<tr>
<td>execute_as_user</td>
<td>If TRUE, then the procedure re-executes the transactions in the security context of the current user. If FALSE, then the procedure re-executes each transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The <code>DBA_APPLY_ERROR</code> data dictionary view lists the original receiver for each error transaction. The user who executes the transactions must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply component.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- The EXECUTE_ALL_ERRORS Procedure and XStream Outbound Servers
- The EXECUTE_ALL_ERRORS Procedure and XStream Inbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.
20.4.11 EXECUTE_ERROR Procedure

This procedure re-executes the specified error transaction in the error queue.

### Syntax

```
DBMS_APPLY_ADM.EXECUTE_ERROR(
    local_transaction_id  IN  VARCHAR2,
    execute_as_user       IN  BOOLEAN   DEFAULT FALSE,
    user_procedure        IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_transaction_id</td>
<td>The identification number of the error transaction to execute. If the specified transaction does not exist in the error queue, then an error is raised.</td>
</tr>
<tr>
<td>execute_as_user</td>
<td>If TRUE, then the procedure re-executes the transaction in the security context of the current user. If FALSE, then the procedure re-executes the transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The DBA_APPLY_ERROR data dictionary view lists the original receiver for each error transaction. The user who executes the transaction must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply component.</td>
</tr>
<tr>
<td>user_procedure</td>
<td>A user-defined procedure that modifies the error transaction so that it can be successfully executed. Specify NULL to execute the error transaction without running a user procedure.</td>
</tr>
</tbody>
</table>

**See Also:** "Usage Notes" for more information about the user procedure.

### Usage Notes

The following usage notes apply to this procedure:

- **The User Procedure**
- **The EXECUTE_ERROR Procedure and XStream Outbound Servers**
• The EXECUTE_ERROR Procedure and XStream Inbound Servers

The User Procedure

You must specify the full procedure name for the user_procedure parameter in one of the following forms:

• \([\text{schema}_\text{name}.]\text{package}_\text{name}.\text{procedure}_\text{name}\)

• \([\text{schema}_\text{name}.]\text{procedure}_\text{name}\)

If the procedure is in a package, then the package_name must be specified. The user who invokes the EXECUTE_ERROR procedure must have EXECUTE privilege on the specified procedure. Also, if the schema_name is not specified, then the user who invokes the EXECUTE_ERROR procedure is the default.

For example, suppose the procedure_name has the following properties:

• strmadmin is the schema_name.

• fix_errors is the package_name.

• fix_hr_errors is the procedure_name.

In this case, specify the following:

```
strmadmin.fix_errors.fix_hr_errors
```

The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure ( 
    in_anydata                    IN      ANYDATA, 
    error_record                  IN      DBA_APPLY_ERROR%ROWTYPE, 
    error_message_number          IN      NUMBER, 
    messaging_default_processing  IN OUT  BOOLEAN, 
    out_anydata                   OUT     ANYDATA);
```

The user procedure has the following parameters:

• in_anydata: The ANYDATA encapsulation of a message that the apply component passes to the procedure. A single transaction can include multiple messages. A message can be a row logical change record (row LCR), a DDL logical change record (DDL LCR), or a user message.

• error_record: The row in the DBA_APPLY_ERROR data dictionary view that identifies the transaction

• error_message_number: The message number of the ANYDATA object in the in_anydata parameter, starting at 1

• messaging_default_processing: If TRUE, then the apply component continues processing the message in the in_anydata parameter, which can include executing DML or DDL statements and invoking apply handlers.

  If FALSE, then the apply component skips processing the message in the in_anydata parameter and moves on to the next message in the in_anydata parameter.

• out_anydata: The ANYDATA object processed by the user procedure and used by the apply component if messaging_default_processing is TRUE.

If an LCR is executed using the EXECUTE LCR member procedure in the user procedure, then the LCR is executed directly, and the messaging_default_processing
parameter should be set to **FALSE**. In this case, the LCR is not passed to any apply handlers.

Processing an error transaction with a user procedure results in one of the following outcomes:

- The user procedure modifies the transaction so that it can be executed successfully.
- The user procedure fails to make the necessary modifications, and an error is raised when transaction execution is attempted. In this case, the transaction is rolled back and remains in the error queue.

The following restrictions apply to the user procedure:

- Do not execute `COMMIT` or `ROLLBACK` statements. Doing so can endanger the consistency of the transaction.
- Do not modify `LONG`, `LONG RAW` or LOB column data in an LCR.
- If the **ANYDATA** object in the **in_anydata** parameter is a row LCR, then the **out_anydata** parameter must be row LCR if the **messaging_default_processing** parameter is set to **TRUE**.
- If the **ANYDATA** object in the **in_anydata** parameter is a DDL LCR, then the **out_anydata** parameter must be DDL LCR if the **messaging_default_processing** parameter is set to **TRUE**.
- The user who runs the user procedure must have the **SELECT** or **READ** privilege on the **DBA_APPLY_ERROR** data dictionary view.

**Note:**

LCRs containing transactional directives, such as `COMMIT` and `ROLLBACK`, are not passed to the user procedure.

The **EXECUTE_ERROR** Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

**The EXECUTE_ERROR Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

**20.4.12 GET_ERROR_MESSAGE Function**

This function returns the message payload from the error queue for the specified message number and transaction identifier. The message can be a logical change record (LCR) or a non-LCR message.

This function is overloaded. One version of this function contains two **OUT** parameters. These **OUT** parameters contain the destination queue into which the message should be enqueued, if one exists, and whether the message should be executed. The destination queue is specified using the **SET_ENQUEUE_DESTINATION** procedure, and the execution directive is specified using the **SET_EXECUTE** procedure.
Syntax

```sql
DBMS_APPLY_ADM.GET_ERROR_MESSAGE(
    message_number          IN   NUMBER,
    local_transaction_id    IN   VARCHAR2,
    destination_queue_name  OUT  VARCHAR2,
    execute                 OUT  BOOLEAN)
RETURN ANYDATA;
```

\[
DBMS_APPLY_ADM.GET_ERROR_MESSAGE(
    message_number          IN   NUMBER,
    local_transaction_id    IN   VARCHAR2)
RETURN ANYDATA;
\]

Parameters

Table 20-13  GET_ERROR_MESSAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_number</td>
<td>The identification number of the message. This number identifies the position of the message in the transaction. Query the DBA_APPLY_ERROR data dictionary view to view the message number of each apply error.</td>
</tr>
<tr>
<td>local_transaction_id</td>
<td>Identifier of the error transaction for which to return a message</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>Contains the name of the queue into which the message should be enqueued. If the message should not be enqueued into a queue, then this parameter contains NULL.</td>
</tr>
<tr>
<td>execute</td>
<td>Contains TRUE if the message should be executed</td>
</tr>
<tr>
<td></td>
<td>Contains FALSE if the message should not be executed</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- The GET_ERROR_MESSAGE Procedure and XStream Outbound Servers
- The GET_ERROR_MESSAGE Procedure and XStream Inbound Servers

The GET_ERROR_MESSAGE Procedure and XStream Outbound Servers

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

The GET_ERROR_MESSAGE Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.
20.4.13 HANDLE_COLLISIONS

This procedure enables or disables basic conflict resolution for an apply process and a table.

Syntax

```
DBMS_APPLY_ADM.HANDLE_COLLISIONS(
    apply_name    IN  VARCHAR2,
    enable        IN  BOOLEAN,
    object        IN  VARCHAR2,
    source_object IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The name of the apply process.</td>
</tr>
<tr>
<td>enable</td>
<td>If TRUE, then the following conflict resolution methods are used:</td>
</tr>
<tr>
<td></td>
<td>- When a conflict is detected for a row that exists in the table, the data in the row LCR overwrites the data in the table.</td>
</tr>
<tr>
<td></td>
<td>- When a conflict is detected for a row that does not exist in the table, the data in the row LCR is ignored.</td>
</tr>
<tr>
<td></td>
<td>If FALSE then it disables conflict resolution set by this procedure for the specified apply process and object.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then removes any explicit table-level setting for collision handling for the specified apply process and object.</td>
</tr>
<tr>
<td>object</td>
<td>The schema and name of the target table, specified as [schema_name.]table_name for the change of the setting.</td>
</tr>
<tr>
<td></td>
<td>For example, if you are changing the setting for table employees owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>source_object</td>
<td>The schema and object name of the source table, specified as [schema_name.]table_name for the table where the change originated.</td>
</tr>
<tr>
<td></td>
<td>For example, if the change originated at the employees table owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
20.4.14 REMOVE_STMT_HANDLER

This procedure removes a statement DML handler for a specified operation on a specified database object from a single apply component or from all apply components in the database.

See Also:

- DBMS_STREAMS_HANDLER_ADM
- Oracle Streams Concepts and Administration

Syntax

DBMS_APPLY_ADM.REMOVE_STMT_HANDLER(
    object_name     IN  VARCHAR2,
    operation_name  IN  VARCHAR2,
    handler_name    IN  VARCHAR2,
    apply_name      IN  VARCHAR2  DEFAULT NULL);

Parameters

Table 20-15  REMOVE_STMT_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the source object specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>operation_name</td>
<td>The name of the operation, which can be specified as:</td>
</tr>
<tr>
<td></td>
<td>- INSERT</td>
</tr>
<tr>
<td></td>
<td>- UPDATE</td>
</tr>
<tr>
<td></td>
<td>- DELETE</td>
</tr>
<tr>
<td>handler_name</td>
<td>The name of the statement DML handler. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of the apply component that uses the statement DML handler. If NULL, then the procedure removes the statement DML handler from all apply components in the database.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- The REMOVE_STMT_HANDLER Procedure and XStream Outbound Servers
- The REMOVE_STMT_HANDLER Procedure and XStream Inbound Servers
The REMOVE_STMT_HANDLER Procedure and XStream Outbound Servers

Outbound servers ignore all apply handlers. This procedure has no effect on XStream outbound servers.

The REMOVE_STMT_HANDLER Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

20.4.15 SET_CHANGE_HANDLER Procedure

This procedure sets or unsets a change handler that tracks changes for a specified operation on a specified database object for a single apply component.

A change handler is a special type of statement DML handler that tracks table changes and was created by either the DBMS_STREAMS_ADM.MAINTAIN_CHANGE_TABLE procedure or this SET_CHANGE_HANDLER procedure. Information about change handlers is stored in the ALL_APPLY_CHANGE_HANDLERS and DBA_APPLY_CHANGE_HANDLERS views.

This procedure automatically generates the statement that is added to the change handler based on values specified in the procedure parameters. You should only run this procedure when a configuration that tracks database changes exists.

Note:

Use the MAINTAIN_CHANGE_TABLE Procedure to configure an environment that tracks table changes

Syntax

```
DBMS_APPLY_ADM.SET_CHANGE_HANDLER(
    change_table_name    IN  VARCHAR2,
    source_table_name    IN  VARCHAR2,
    capture_values       IN  VARCHAR2,
    apply_name           IN  VARCHAR2,
    operation_name       IN  VARCHAR2,
    change_handler_name  IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 20-16 SET_CHANGE_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>change_table_name</td>
<td>The table that records changes to the source table. Specify the table as [schema_name.]table_name. For example, hr.jobs_change_table. If the schema is not specified, then the current user is the default. If NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
Table 20-16  (Cont.) SET_CHANGE_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_table_name</td>
<td>The table at the source database for which changes are recorded. Specify the table as [schema_name.]table_name. For example, hr.jobs. If the schema is not specified, then the current user is the default. If NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
| capture_values        | Specify which values to record in the change table when update operations are performed on the source table:  
  - old - To record the original values for an updated column in the source table  
  - new - To record the new values for an updated column in the source table  
  - * - To record both the original and the new values for an updated column in the source table  
  If NULL, then the procedure raises an error.  
  **Note:** For insert operations, only new column values are captured. For delete operations, only old column values are captured. |
| apply_name            | The name of the apply component that applies changes to the change table. Do not specify an owner. If NULL, then the procedure raises an error.                                                                 |
| operation_name        | The name of the operation, which can be specified as:  
  - INSERT  
  - UPDATE  
  - DELETE  
  If NULL, then the procedure raises an error.  
  **Note:** Change handlers cannot be specified for LOB operations. |
| change_handler_name   | The name of the change handler. If the specified change handler exists, then a statement is added to the existing handler. Ensure that the existing change handler is for the same operation on the same table as the settings for the operation_name and source_table_name parameters, respectively. If the existing handler is for a different operation or table, then an apply error results when the handler is invoked. If non-NULL and the specified change handler does not exist, then this procedure creates the change handler. If NULL and a change handler exists for the same operation on the same table as the settings for the operation_name and source_table_name parameters, respectively, then the existing change handler is removed. If NULL and the specified change handler does not exist, then the procedure raises an error.  
  **See Also:** "Usage Notes" for more information about this parameter. |

Usage Notes

The following usage notes apply to this procedure:

- Checking for an Existing Change Handler
• The SET_CHANGE_HANDLER Procedure and XStream Outbound Servers

• The SET_CHANGE_HANDLER Procedure and XStream Inbound Servers

Checking for an Existing Change Handler

To check for an existing change handler for a specific operation on a specific source table, run the following query:

```sql
SELECT HANDLER_NAME, APPLY_NAME FROM DBA_APPLY_CHANGE_HANDLERS
WHERE operation_name = '
operation'
AND source_table_owner = '
source_table_owner'
AND source_table_name = '
source_table_name'
AND change_table_owner = '
change_table_owner'
AND change_table_name = '
change_table_name';
```

where:

- `operation` is operation specified for the new handler, either INSERT, UPDATE, or DELETE
- `source_table_owner` is the owner of the source table
- `source_table_name` is the name of the source table
- `change_table_owner` is the owner of the change table
- `change_table_name` is the owner of the change table

The SET_CHANGE_HANDLER Procedure and XStream Outbound Servers

Outbound servers ignore all apply handlers. This procedure has no effect on XStream outbound servers.

The SET_CHANGE_HANDLER Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

### 20.4.16 SET_DML_CONFLICT_HANDLER Procedure

This procedure adds, modifies, or removes a prebuilt DML conflict handler for INSERT, UPDATE, or DELETE conflicts on the specified object.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

**Syntax**

```sql
DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
apply_name             IN  VARCHAR2,
conflict_handler_name  IN  VARCHAR2,
object                 IN  VARCHAR2  DEFAULT NULL,
operation_name         IN  VARCHAR2  DEFAULT NULL,
conflict_type          IN  VARCHAR2  DEFAULT NULL,
method_name            IN  VARCHAR2  DEFAULT NULL,
column_list            IN  VARCHAR2  DEFAULT NULL,
resolution_column      IN  VARCHAR2  DEFAULT NULL,
source_object          IN  VARCHAR2  DEFAULT NULL);
```

```
DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
apply_name             IN  VARCHAR2,
conflict_handler_name  IN  VARCHAR2,
object                 IN  VARCHAR2  DEFAULT NULL,
operation_name         IN  VARCHAR2  DEFAULT NULL,
conflict_type          IN  VARCHAR2  DEFAULT NULL,
method_name            IN  VARCHAR2  DEFAULT NULL,
column_list            IN  VARCHAR2  DEFAULT NULL,
resolution_column      IN  VARCHAR2  DEFAULT NULL,
source_object          IN  VARCHAR2  DEFAULT NULL);
```
Parameters

Table 20-17  SET_DML_CONFLICT_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The name of the apply process.</td>
</tr>
<tr>
<td>conflict_handler_name</td>
<td>The name of the conflict handler.</td>
</tr>
<tr>
<td>object</td>
<td>The schema and name of the target table, specified as [schema name.]table name for which a conflict handler is being added, modified, or removed. For example, if an update conflict handler is being added for table employees owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>operation_name</td>
<td>The name of the operation, which can be specified as:</td>
</tr>
<tr>
<td></td>
<td>• INSERT</td>
</tr>
<tr>
<td></td>
<td>• UPDATE</td>
</tr>
<tr>
<td></td>
<td>• DELETE</td>
</tr>
<tr>
<td></td>
<td>In order to set up conflict handlers for different operations on the same table, you must make one call per operation.</td>
</tr>
<tr>
<td>conflict_type</td>
<td>Type of update conflict handler to create.</td>
</tr>
<tr>
<td></td>
<td>You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained:</td>
</tr>
<tr>
<td></td>
<td>• ROW_EXISTS: A row with the same primary key already exists in the database for an insert or update.</td>
</tr>
<tr>
<td></td>
<td>• ROW_MISSING: A row with the same primary key cannot be found for an update or delete.</td>
</tr>
</tbody>
</table>
Table 20-17 (Cont.) SET_DML_CONFLICT_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method_name</td>
<td>Type of update conflict handler to create. You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained:</td>
</tr>
<tr>
<td></td>
<td>• DELTA: If the conflict type is ROW_EXISTS and the LCR is an update, then take the difference between the old and new values of the column in the LCR and add it to the current value of the column in the target database. The columns in the column group must be of type NUMBER.</td>
</tr>
<tr>
<td></td>
<td>• IGNORE: Silently ignores the LCR and can be used for all conflict types.</td>
</tr>
<tr>
<td></td>
<td>• MAXIMUM: Applies the column list from the source database if it has the greater value for the resolution column. Otherwise, retains the values at the destination database. This resolution method is only supported for ROW_EXISTS and only applies to inserts and updates.</td>
</tr>
<tr>
<td></td>
<td>• MINIMUM: Applies the column list from the source database if it has the lesser value for the resolution column. Otherwise, retains the values at the destination database. This resolution method is only supported for ROW_EXISTS and only applies to inserts and updates.</td>
</tr>
<tr>
<td></td>
<td>• OVERWRITE: Applies the column list from the source database, overwriting the column values at the destination database. An INSERT with ROW_EXISTS is converted to an UPDATE. An UPDATE with ROW_MISSING is converted to an INSERT. A DELETE with ROW_MISSING is ignored.</td>
</tr>
<tr>
<td></td>
<td>• RECORD: Enqueue the LCR into the error queue. Can be used for all conflict types and can only be specified for a column group that contains all the columns in the table.</td>
</tr>
<tr>
<td></td>
<td>• MAX_AND_EQUALS: Applies the column list from the source database if the value of resolution column is greater than or equal to the value of the column in the database.</td>
</tr>
<tr>
<td></td>
<td>• MIN_AND_EQUALS: Applies the column list from the source database if the value of resolution column is less than or equal to the value of the column in the database.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure removes any existing conflict handler with the same object_name, resolution_group, and conflict_type.</td>
</tr>
<tr>
<td></td>
<td>If a conflict handler already exists with the same object_name and resolution_column and conflict_type, then the existing handler is replaced.</td>
</tr>
</tbody>
</table>
Table 20-17 (Cont.) SET_DML_CONFLICT_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| column_list     | A comma-separated list of the column names for which the conflict handler is called. The same column cannot be in more than one column list (for a given apply_name, object_name, operation_name and conflict_type). Specify * for the default column group, which includes all the columns in the table that are not already specified in another column list (for a given apply_name, object_name, operation_name and conflict_type). If a conflict occurs for one or more of the columns in the list when an apply component tries to apply a row logical change record (row LCR), then the conflict handler is called to resolve the conflict. The conflict handler is not called if a conflict occurs only for columns that are not in the list. You cannot use a column_list if you use a * in the object_name. The only time you can use multiple column groups is when you are specifying a conflict handler for insert or update for ROW_EXISTS. Note: Prebuilt conflict handlers do not support LOB, LONG, RAW, user-defined type, and Oracle-supplied type columns. Therefore, you should not include these types of columns in the column_list parameter. This parameter must be set to '*' in the following cases:  
  - The operation_name is DELETE.  
  - The method_name is RECORD.  
  - The operation_name is UPDATE and the conflict_type is ROW_MISSING. |
| column_table    | An array of column names for which the conflict handler is called. This parameter is the same as the column_list parameter, but it uses an array instead of a list for the column names. Note: The column_list and column_table parameters are mutually exclusive. |
| resolution_column | For the MAXIMUM and MINIMUM prebuilt methods, the resolution column is the one tested to determine whether the current row or the LCR has the smaller value. The resolution column must be one of the columns listed in the column_list or column_table parameter. You can specify NULL for other resolution methods. |
| source_object   | The schema and object name of the source table, specified as [schema_name.]table_name for the table where the change originated. For example, if the change originated at the employees table owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default. |
Usage Notes

The following usage notes apply to this procedure:

• Modifying an Existing Conflict Handler
• Removing an Existing Conflict Handler
• Series of Actions for Conflicts
• Procedure DML Handlers for Conflicts
• A Column Can Be in Only One Column List
• The SET_DML_CONFLICT_HANDLER Procedure and XStream Outbound Servers
• The SET_DML_CONFLICT_HANDLER Procedure and XStream Inbound Servers
• Table 20-18
• Example

Modifying an Existing Conflict Handler

If you want to modify an existing conflict handler, then you specify the object, conflict_type, and resolution_column of an the existing conflict handler. You can modify the method_name or the column_list.

Removing an Existing Conflict Handler

If you want to remove an existing conflict handler, then specify NULL for the method_name and specify the object, conflict_type, and resolution_column of the existing conflict handler.

Series of Actions for Conflicts

If an conflict occurs, then Oracle completes the following series of actions:

1. Calls the appropriate conflict handler to resolve the conflict
2. If no conflict handler is specified or if the conflict handler cannot resolve the conflict, then calls the appropriate error handler for the apply component, object name, and operation name to handle the error
3. If no error handler is specified or if the error handler cannot resolve the error, then raises an error and moves the transaction containing the row LCR that caused the error to the error queue

See Also:

"Signature of a DML Handler Procedure or Error Handler Procedure" for information about setting an error handler

Procedure DML Handlers for Conflicts

If you cannot use a prebuilt conflict handler to meet your requirements, then you can create a PL/SQL procedure to use as a custom conflict handler. You use the SET_DML_HANDLER procedure to designate one or more custom conflict handlers for a particular table. In addition, a custom conflict handler can process LOB columns and use LOB assembly.
A Column Can Be in Only One Column List

When a column is in a column list, and you try to add the same column to another column list, this procedure returns the following error:

ORA-00001: UNIQUE CONSTRAINT (SYS.APPLY$_CONF_HDLR_COLUMNS_UNQ1) VIOLATED

The **SET_DML_CONFLICT_HANDLER** Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The **SET_DML_CONFLICT_HANDLER** Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

### Table 20-18  Valid Combinations of Parameters

<table>
<thead>
<tr>
<th>Operation</th>
<th>Conflict Type</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>ROW_EXISTS</td>
<td>OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGNORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAXIMUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MINIMUM</td>
</tr>
<tr>
<td>UPDATE</td>
<td>ROW_EXISTS</td>
<td>OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGNORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAXIMUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MINIMUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DELTA</td>
</tr>
<tr>
<td>UPDATE</td>
<td>ROW_MISSING</td>
<td>OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGNORE</td>
</tr>
<tr>
<td>DELETE</td>
<td>ROW_EXISTS</td>
<td>OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGNORE</td>
</tr>
<tr>
<td>DELETE</td>
<td>ROW_MISSING</td>
<td>RECORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGNORE</td>
</tr>
</tbody>
</table>

### Example

The following is an example for setting a conflict handler for the employees table in the hr schema:

```sql
DECLARE
  cols  DBMSUTILITY.NAME_ARRAY;
```
BEGIN
  cols(1) := 'salary';
  cols(2) := 'commission_pct';
  DBMS_APPLY_ADM.SET_DML_CONFLICT_HANDLER(
    apply_name              => 'appl1',
    conflict_handler_name   => 'emp_handler_update',
    object                  => 'hr.employees',
    operation_name          => 'UPDATE',
    conflict_type           => 'ROW_EXISTS',
    method_name             => 'MAXIMUM',
    resolution_column       => 'salary',
    column_table            => cols);
END;
/

This example sets a conflict handler named `emp_handler_update` that is called if a conflict occurs for the `salary` or `commission_pct` column in the `hr.employees` table. If such a conflict occurs, then the `salary` column is evaluated to resolve the conflict. If a conflict occurs only for a column that is not in the column list, such as the `job_id` column, then this conflict handler is not called.

### 20.4.17 SET_DML_HANDLER Procedure

This procedure sets or unsets a user procedure as a procedure DML handler for a specified operation on a specified database object for a single apply component or for all apply components in the database. The user procedure alters the apply behavior for the specified operation on the specified object.

#### Syntax

```sql
DBMS_APPLY_ADM.SET_DML_HANDLER(
  object_name          IN VARCHAR2,
  object_type          IN VARCHAR2,
  operation_name       IN VARCHAR2,
  error_handler        IN BOOLEAN DEFAULT FALSE,
  user_procedure       IN VARCHAR2,
  apply_database_link  IN VARCHAR2 DEFAULT NULL,
  apply_name           IN VARCHAR2 DEFAULT NULL,
  assemble_lobs        IN BOOLEAN DEFAULT TRUE);
```

#### Parameters

**Table 20-19  SET_DML_HANDLER Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the source object specified as <code>[schema_name.]object_name</code>. For example, <code>hr.employees</code>. If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure.</td>
</tr>
<tr>
<td>object_type</td>
<td>The type of the source object. Currently, <code>TABLE</code> is the only possible source object type.</td>
</tr>
</tbody>
</table>
### Table 20-19  (Cont.) SET_DML_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation_name</td>
<td>The name of the operation, which can be specified as:</td>
</tr>
<tr>
<td></td>
<td>· INSERT</td>
</tr>
<tr>
<td></td>
<td>· UPDATE</td>
</tr>
<tr>
<td></td>
<td>· DELETE</td>
</tr>
<tr>
<td></td>
<td>· LOB_UPDATE</td>
</tr>
<tr>
<td></td>
<td>· DEFAULT</td>
</tr>
<tr>
<td></td>
<td>The procedure must be run for each operation individually.</td>
</tr>
<tr>
<td>error_handler</td>
<td>If TRUE, then the specified user procedure is run when a row logical change record (row LCR) involving the specified operation on the specified object raises an apply error. You can code the user procedure to resolve possible error conditions, notify administrators of the error, log the error, or any combination of these actions. If FALSE, then the handler being set is run for all row LCRs involving the specified operation on the specified object.</td>
</tr>
<tr>
<td>user_procedure</td>
<td>A user-defined procedure that is invoked during apply for the specified operation on the specified object. If the procedure is a procedure DML handler, then it is invoked instead of the default apply performed by Oracle. If the procedure is an error handler, then it is invoked when an apply error is encountered. Specify NULL to unset a procedure DML handler that is set for the specified operation on the specified object.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of the apply component that uses the procedure DML handler or error handler. If NULL, then the procedure sets the procedure DML handler or error handler as a general handler for all apply components in the database. If the user_procedure parameter is set to NULL to unset a handler, and the handler being unset is set for a specific apply component, then use the apply_name parameter to specify the apply component to unset the handler.</td>
</tr>
<tr>
<td>assemble_lobs</td>
<td>If TRUE, then LOB assembly is used for LOB columns in LCRs processed by the handler. LOB assembly combines multiple LCRs for a LOB column resulting from a single row change into one row LCR before passing the LCR to the handler. Database compatibility must be 10.2.0 or higher to use LOB assembly. If FALSE, then LOB assembly is not used for LOB columns in LCRs processed by the handler.</td>
</tr>
</tbody>
</table>

### Usage Notes

The following usage notes apply to this procedure:
Run the SET_DML_HANDLER Procedure at the Destination Database

Procedure DML Handlers and Error Handlers

The apply_name Parameter

Signature of a DML Handler Procedure or Error Handler Procedure

LOB Assembly

The SET_DML_HANDLER Procedure and XStream Outbound Servers

The SET_DML_HANDLER Procedure and XStream Inbound Servers

Run this procedure at the destination database. The SET_DML_HANDLER procedure provides a way for users to apply logical change records containing DML changes (row LCRs) using a customized apply.

Procedure DML Handlers and Error Handlers

If the error_handler parameter is set to TRUE, then it specifies that the user procedure is an error handler. An error handler is invoked only when a row LCR raises an apply error. Such an error can result from a data conflict if no conflict handler is specified or if the update conflict handler cannot resolve the conflict. If the error_handler parameter is set to FALSE, then the user procedure is a procedure DML handler, not an error handler, and a procedure DML handler is always run instead of performing the specified operation on the specified object.

This procedure either sets a procedure DML handler or an error handler for a particular operation on an object. It cannot set both a procedure DML handler and an error handler for the same object and operation.

Note:
Currently, setting an error handler for an apply component that is applying changes to a non-Oracle database is not supported.

The apply_name Parameter

If the apply_name parameter is non-NULL, then the procedure DML handler or error handler is set for the specified apply component. In this case, this handler is not invoked for other apply components at the local destination database. If the apply_name parameter is NULL, the default, then the handler is set as a general handler for all apply components at the destination database. When a handler is set for a specific apply component, then this handler takes precedence over any general handlers. For example, consider the following scenario:

- A procedure DML handler named handler_hr is specified for an apply component named apply_hr for UPDATE operations on the hr.employees table.
- A general procedure DML handler named handler_gen also exists for UPDATE operations on the hr.employees table.

In this case, the apply_hr apply component uses the handler_hr procedure DML handler for UPDATE operations on the hr.employees table.

At the source database, you must specify an unconditional supplemental log group for the columns needed by a DML or error handler.
Signature of a DML Handler Procedure or Error Handler Procedure

You can use the SET_DML_HANDLER procedure to set either a procedure DML handler or an error handler for row LCRs that perform a specified operation on a specified object. The signatures of a DML handler procedure and of an error handler procedure are described following this section.

In either case, you must specify the full procedure name for the user_procedure parameter in one of the following forms:

- `[schema_name.]package_name.procedure_name`
- `[schema_name.]procedure_name`

If the procedure is in a package, then the package_name must be specified. The user who invokes the SET_DML_HANDLER procedure must have EXECUTE privilege on the specified procedure. Also, if the schema_name is not specified, then the user who invokes the SET_DML_HANDLER procedure is the default.

For example, suppose the procedure_name has the following properties:

- hr is the schema_name.
- apply_pkg is the package_name.
- employees_default is the procedure_name.

In this case, specify the following:

`hr.apply_pkg.employees_default`

The following restrictions apply to the user procedure:

- Do not execute COMMIT or ROLLBACK statements. Doing so can endanger the consistency of the transaction that contains the LCR.

- If you are manipulating a row using the EXECUTE member procedure for the row LCR, then do not attempt to manipulate more than one row in a row operation. You must construct and execute manually any DML statements that manipulate more than one row.

- If the command type is UPDATE or DELETE, then row operations resubmitted using the EXECUTE member procedure for the LCR must include the entire key in the list of old values. The key is the primary key or the smallest unique index that has at least one NOT NULL column, unless a substitute key has been specified by the SET_KEY_COLUMNS procedure. If there is no specified key, then the key consists of all non LOB, non LONG, and non LONG RAW columns.

- If the command type is INSERT, then row operations resubmitted using the EXECUTE member procedure for the LCR should include the entire key in the list of new values. Otherwise, duplicate rows are possible. The key is the primary key or the smallest unique index that has at least one NOT NULL column, unless a substitute key has been specified by the SET_KEY_COLUMNS procedure. If there is no specified key, then the key consists of all of the table columns, except for columns of the following data types: LOB, LONG, LONG RAW, user-defined types (including object types, REFS, varrays, nested tables), and Oracle-supplied types (including Any types, XML types, spatial types, and media types).
The procedure specified in the `user_procedure` parameter must have the following signature:

```sql
PROCEDURE user_procedure (  
    parameter_name  IN  ANYDATA);
```

Here, `user_procedure` stands for the name of the procedure and `parameter_name` stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is a `ANYDATA` encapsulation of a row LCR.

The procedure you create for error handling must have the following signature:

```sql
PROCEDURE user_procedure (  
    message            IN  ANYDATA,  
    error_stack_depth  IN  NUMBER,  
    error_numbers      IN  DBMS_UTILITY.NUMBER_ARRAY,  
    error_messages     IN  emsg_array);
```

If you want to retry the DML operation within the error handler, then have the error handler procedure run the `EXECUTE` member procedure for the LCR. The last error raised is on top of the error stack. To specify the error message at the top of the error stack, use `error_numbers(1)` and `error_messages(1)`.

Note:

- Each parameter is required and must have the specified datatype. However, you can change the names of the parameters.
- The `emsg_array` value must be a user-defined array that is a table of type `VARCHAR2` with at least 76 characters.

Running an error handler results in one of the following outcomes:

- The error handler successfully resolves the error and returns control to the apply component.
- The error handler fails to resolve the error, and the error is raised. The raised error causes the transaction to be rolled back and placed in the error queue.
LOB Assembly

Do not modify LONG, LONG RAW, or nonassembled LOB column data in an LCR with procedure DML handlers, error handlers, or custom rule-based transformation functions. Procedure DML handlers and error handlers can modify LOB columns in row LCRs that have been constructed by LOB assembly.

The SET_DML_HANDLER Procedure and XStream Outbound Servers

Outbound servers ignore all apply handlers. This procedure has no effect on XStream outbound servers.

The SET_DML_HANDLER Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

20.4.18 SET_ENQUEUE_DESTINATION Procedure

This procedure sets the queue where the apply component automatically enqueues a message that satisfies the specified rule.

This procedure modifies the specified rule's action context to specify the queue. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to TRUE for a message. In this case, the client of the rules engine is an apply component. The information in an action context is an object of type SYS.RE$NV_LIST, which consists of a list of name-value pairs.

A queue destination specified by this procedure always consists of the following name-value pair in an action context:

- The name is APPLY$_ENQUEUE.
- The value is an ANYDATA instance containing the queue name specified as a VARCHAR2.

Syntax

DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION(
    rule_name               IN  VARCHAR2,
    destination_queue_name  IN  VARCHAR2);

Parameters

Table 20-20  SET_ENQUEUE_DESTINATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. For example, to specify a rule named hr5 in the hr schema, enter hr.hr5 for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
Table 20-20  (Cont.) SET_ENQUEUE_DESTINATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_queue_name</td>
<td>The name of the queue into which the apply component enqueues the message. Specify the queue in the form [schema_name.]queue_name. Only local queues can be specified. For example, to specify a queue in the hr schema named streams_queue, enter hr.streams_queue. If the schema is not specified, then the current user is the default. If NULL, then an existing name-value pair with the name APPLY$_ENQUEUE is removed. If no name-value pair exists with the name APPLY$_ENQUEUE for the rule, then no action is taken. If non-NULL and a name-value pair exists for the rule with the name APPLY$_ENQUEUE, then it is removed, and a new name-value pair with the value specified by this parameter is added.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

• The SET_ENQUEUE_DESTINATION Procedure and Apply Handlers
• Considerations for the SET_ENQUEUE_DESTINATION Procedure
• The SET_ENQUEUE_DESTINATION Procedure and XStream Outbound Servers
• The SET_ENQUEUE_DESTINATION Procedure and XStream Inbound Servers

The SET_ENQUEUE_DESTINATION Procedure and Apply Handlers

If an apply handler, such as a procedure DML handler, DDL handler, or message handler, processes a message that also is enqueued into a destination queue, then the apply handler processes the message before it is enqueued.

Considerations for the SET_ENQUEUE_DESTINATION Procedure

The following are considerations for using this procedure:

• This procedure does not verify that the specified queue exists. If the queue does not exist, then an error is raised when an apply component tries to enqueue a message into it.
• Oracle Streams capture processes, propagations, and messaging clients ignore the action context created by this procedure.
• The apply user of the apply component using the specified rule must have the necessary privileges to enqueue messages into the specified queue. If the queue is a secure queue, then the apply user must be a secure queue user of the queue.
• The specified rule must be in the positive rule set for an apply component. If the rule is in the negative rule set for an apply component, then the apply component does not enqueue the message into the destination queue.
• If the commit SCN for a message is less than or equal to the relevant instantiation SCN for the message, then the message is not enqueued into the destination queue, even if the message satisfies the apply component rule sets.
The SET_ENQUEUE_DESTINATION Procedure and XStream Outbound Servers
This procedure has no effect on XStream outbound servers.

The SET_ENQUEUE_DESTINATION Procedure and XStream Inbound Servers
This procedure functions the same way for apply processes and inbound servers.

20.4.19 SET_EXECUTE Procedure

This procedure specifies whether a message that satisfies the specified rule is executed by an apply component.

This procedure modifies the specified rule's action context to specify message execution. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to TRUE for a message. In this case, the client of the rules engine is an apply component. The information in an action context is an object of type SYS.RE$NV_LIST, which consists of a list of name-value pairs.

A message execution directive specified by this procedure always consists of the following name-value pair in an action context:

- The name is APPLY$_EXECUTE.
- The value is an ANYDATA instance that contains NO as a VARCHAR2. When the value is NO, an apply component does not execute the message and does not send the message to any apply handler.

Syntax

```sql
DBMS_APPLY_ADM.SET_EXECUTE(
    rule_name  IN  VARCHAR2,
    execute    IN  BOOLEAN);
```

Parameters

Table 20-21  SET_EXECUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. For example, to specify a rule named hr5 in the hr schema, enter hr.hr5 for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>execute</td>
<td>If TRUE, then the procedure removes the name-value pair with the name APPLY$_EXECUTE for the specified rule. Removing the name-value pair means that the apply component executes messages that satisfy the rule. If no name-value pair with name APPLY$_EXECUTE exists for the rule, then no action is taken. If FALSE, then the procedure adds a name-value pair to the rule's action context. The name is APPLY$_EXECUTE and the value is NO. An apply component does not execute a message that satisfies the rule and does not send the message to any apply handler. If a name-value pair exists for the rule with the name APPLY$_EXECUTE, then it is removed, and a new one with the value NO is added. If NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
Usage Notes

The following usage notes apply to this procedure:

• Considerations for the SET_EXECUTE Procedure
• The SET_EXECUTE Procedure and XStream Outbound Servers
• The SET_EXECUTE Procedure and XStream Inbound Servers

Considerations for the SET_EXECUTE Procedure

The following are considerations for using this procedure:

• If the message is a logical change record (LCR) and the message is not executed, then the change encapsulated in the LCR is not made to the relevant local database object. Also, if the message is not executed, then it is not sent to any apply handler.
• Oracle Streams capture processes, propagations, and messaging clients ignore the action context created by this procedure.
• The specified rule must be in the positive rule set for an apply component for the apply component to follow the execution directive. If the rule is in the negative rule set for an apply component, then the apply component ignores the execution directive for the rule.

The SET_EXECUTE Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The SET_EXECUTE Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

20.4.20 SET_GLOBAL_INSTANTIATION_SCN Procedure

This procedure records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas. This procedure overwrites any existing instantiation SCN for the database, and, if it sets the instantiation SCN for a schema or a table, then it overwrites any existing instantiation SCN for the schema or table.

This procedure gives you precise control over which DDL logical change records (DDL LCRs) from a source database are ignored and which DDL LCRs are applied by an apply component.

Syntax

```sql
DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL,
  recursive             IN  BOOLEAN   DEFAULT FALSE,
  source_root_name      IN  VARCHAR2  DEFAULT NULL);
```
Parameters

Table 20-22  SET_GLOBAL_INSTANTIATION_SCN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database_name</td>
<td>The global name of the source database. For example, DBS1.NET. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>instantiation_scn</td>
<td>The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source database from the data dictionary.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database.</td>
</tr>
<tr>
<td>recursive</td>
<td>If TRUE, then the procedure sets the instantiation SCN for the source database, all schemas in the source database, and all tables owned by the schemas in the source database. The procedure selects the schemas and tables from the ALL_USERS and ALL_TABLES data dictionary views, respectively, at the source database under the security context of the current user. If FALSE, then the procedure sets the global instantiation SCN for the source database, but does not set the instantiation SCN for any schemas or tables. <strong>Note:</strong> If recursive is set to TRUE, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the current user. Also, a table must be accessible to the current user in either the ALL_TABLES or DBA_TABLES data dictionary view at the source database for this procedure to set the instantiation SCN for the table at the destination database.</td>
</tr>
<tr>
<td>source_root_name</td>
<td>The global name of the source root database. In a non-CDB, this parameter must be NULL. In a CDB, if you want to do the instantiation for a specific container than you must specify both source_database and source_root_name. If you want to do the instantiation for all the containers in the CDB, specify the source_root_name for the database and leave the source_database name as NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- Instantiation SCNs and DDL LCRs
- The recursive Parameter
- Considerations for the SET_GLOBAL_INSTANTIATION_SCN Procedure
- The SET_GLOBAL_INSTANTIATION_SCN Procedure and XStream Outbound Servers
Instantiation SCNs and DDL LCRs

If the commit SCN of a DDL LCR for a database object from a source database is less than or equal to the instantiation SCN for that source database at a destination database, then the apply component at the destination database disregards the DDL LCR. Otherwise, the apply component applies the DDL LCR.

The global instantiation SCN specified by this procedure is used for a DDL LCR only if the DDL LCR does not have object_owner, base_table_owner, and base_table_name specified. For example, the global instantiation SCN set by this procedure is used for DDL LCRs with a command_type of CREATE USER.

The recursive Parameter

If the recursive parameter is set to TRUE, then this procedure sets the instantiation SCN for each schema at a source database and for the tables owned by these schemas. This procedure uses the SET_SCHEMA_INSTANTIATION_SCN procedure to set the instantiation SCN for each schema, and it uses the SET_TABLE_INSTANTIATION_SCN procedure to set the instantiation SCN for each table. Each schema instantiation SCN is used for DDL LCRs on the schema, and each table instantiation SCN is used for DDL LCRs and row LCRs on the table.

If the recursive parameter is set to FALSE, then this procedure does not set the instantiation SCN for any schemas or tables.

Considerations for the SET_GLOBAL_INSTANTIATION_SCN Procedure

The following are considerations for using this procedure:

- Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.
- The instantiation SCN is not set for the SYS or SYSTEM schemas.

The SET_GLOBAL_INSTANTIATION_SCN Procedure and XStream Outbound Servers

Instantiation SCNs are not required for database objects processed by an outbound server. If an instantiation SCN is set for a database object, then the outbound server only sends the LCRs for the database object with SCN values that are greater than the instantiation SCN value. If a database object does not have an instantiation SCN set, then the outbound server skips the instantiation SCN check and sends all LCRs for that database object. In both cases, the outbound server only sends LCRs that satisfy its rule sets.
The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

See Also:

*Oracle Database XStream Guide* for more information about outbound servers and instantiation SCNs

The SET_GLOBAL_INSTANTIATION_SCN Procedure and XStream Inbound Servers

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.

The SET_GLOBAL_INSTANTIATION_SCN Procedure and CDBs

In a CDB, this procedure must be invoked from the same container as the apply process that uses the instantiation SCN information.

### 20.4.21 SET_KEY_COLUMNS Procedures

This procedure records the set of columns to be used as the substitute primary key for apply purposes and removes existing substitute primary key columns for the specified object if they exist.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

**Syntax**

```sql
DBMS_APPLY_ADM.SET_KEY_COLUMNS(
    object_name          IN  VARCHAR2,
    column_list          IN  VARCHAR2,
    apply_database_link  IN  VARCHAR2  DEFAULT NULL);

DBMS_APPLY_ADM.SET_KEY_COLUMNS(
    object_name          IN  VARCHAR2,
    column_table         IN  DBMS_UTILITY.NAME_ARRAY,
    apply_database_link  IN  VARCHAR2  DEFAULT NULL);
```

**Parameters**

**Table 20-23  SET_KEY_COLUMNS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object_name</code></td>
<td>The name of the table specified as <code>[schema_name.]object_name</code>. For example, <code>hr.employees</code>. If the schema is not specified, then the current user is the default. If the apply component is applying changes to a non-Oracle database in a heterogeneous environment, then the object name is not verified.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>column_list</td>
<td>A comma-delimited list of the columns in the table to use as the substitute primary key, with no spaces between the column names. If the <code>column_list</code> parameter is empty or <code>NULL</code>, then the current set of key columns is removed.</td>
</tr>
<tr>
<td>column_table</td>
<td>A PL/SQL associative array of type <code>DBMS_UTILITY.NAME_ARRAY</code> of the columns in the table to use as the substitute primary key. The index for <code>column_table</code> must be 1-based, increasing, dense, and terminated by a <code>NULL</code>.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The following usage notes apply to this procedure:

- **Considerations for the SET_KEY_COLUMNS Procedure**
- **Duplicate Rows and Substitute Primary Key Columns**
- **The SET_KEY_COLUMNS Procedure and XStream Outbound Servers**
- **The SET_KEY_COLUMNS Procedure and XStream Inbound Servers**
- **The SET_KEY_COLUMNS Procedure and CDBs**

**Considerations for the SET_KEY_COLUMNS Procedure**

The following are considerations for using this procedure:

- When not empty, the specified set of columns takes precedence over any primary key for the specified object. Do not specify substitute key columns if the object has primary key columns and you want to use those primary key columns as the key.
- Run this procedure at the destination database. At the source database, you must specify an unconditional supplemental log group for the substitute key columns.
- Unlike true primary keys, columns specified as substitute key column columns can contain`NULLS`. However, Oracle recommends that each column you specify as a substitute key column be a `NOT NULL` column. You also should create a single index that includes all of the columns in a substitute key. Following these guidelines improves performance for updates, deletes, and piecewise updates to LOBs because Oracle can locate the relevant row more efficiently.
- Do not permit applications to update the primary key or substitute key columns of a table. This ensures that Oracle can identify rows and preserve the integrity of the data.
- If there is neither a primary key, nor a unique index that has at least one `NOT NULL` column, nor a substitute key for a table, then the key consists of all of the table columns, except for columns of the following data types: LOB, `LONG`, `LONG RAW`, user-defined types (including object types, `REFs`, varrays, nested tables), and Oracle-supplied types (including Any types, XML types, spatial types, and media types).
Duplicate Rows and Substitute Primary Key Columns

A table has duplicate rows when all of the column values are identical for two or more rows in the table, excluding LOB, LONG, and LONG RAW columns. You can specify substitute primary key columns for a table at a destination database using the SET_KEY_COLUMNS procedure. When substitute primary key columns are specified for a table with duplicate rows at a destination database, and the allow_duplicate_rows apply component parameter is set to Y, meet the following requirements to keep the table data synchronized at the source and destination databases:

- Ensure that supplemental logging is specified at source database for the columns specified as substitute key columns at the destination database. The substitute key columns must be in an unconditional log group at the source database.
- Ensure that the substitute key columns uniquely identify each row in the table at the destination database.

The rest of this section provides more details about these requirements.

When there is no key for a table and the allow_duplicate_rows apply component parameter is set to Y, a single row LCR with an UPDATE or DELETE command type only is applied to one of the duplicate rows. In this case, if the table at the source database and the table at the destination database have corresponding duplicate rows, then a change that changes all of the duplicate rows at the source database also changes all the duplicate rows at the destination database when the row LCRs resulting from the change are applied.

For example, suppose a table at a source database has two duplicate rows. An update is performed on the duplicate rows, resulting in two row LCRs. At the destination database, one row LCR is applied to one of the duplicate rows. At this point, the rows are no longer duplicate at the destination database because one of the rows has changed. When the second row LCR is applied at the destination database, the rows are duplicate again. Similarly, if a delete is performed on these duplicate rows at the source database, then both rows are deleted at the destination database when the row LCRs resulting from the source change are applied.

When substitute primary key columns are specified for a table, row LCRs are identified with rows in the table during apply using the substitute primary key columns. If substitute primary key columns are specified for a table with duplicate rows at a destination database, and the allow_duplicate_rows apply component parameter is set to Y, then an update performed on duplicate rows at the source database can result in different changes when the row LCRs are applied at the destination database. Specifically, if the update does not change one of the columns specified as a substitute primary key column, then the same duplicate row can be updated multiple times at the destination database, while other duplicate rows might not be updated.

Also, if the substitute key columns do not identify each row in the table at the destination database uniquely, then a row LCR identified with multiple rows can update any one of the rows. In this case, the update in the row LCR might not be applied to the correct row in the table at the destination database.

An apply component ignores substitute primary key columns when it determines whether rows in a table are duplicates. An apply component determines that rows are duplicates only if all of the column values in the rows are identical (excluding LOB, LONG, and LONG RAW columns). Therefore, an apply component always raises an error if a single update or delete changes two or more nonduplicate rows in a table.
For example, consider a table with columns \( c_1, c_2, \) and \( c_3 \) on which the `SET_KEY_COLUMNS` procedure is used to designate column \( c_1 \) as the substitute primary key. If two rows have the same key value for the \( c_1 \) column, but different value for the \( c_2 \) or \( c_3 \) columns, then an apply component does not treat the rows as duplicates. If an update or delete modifies more than one row because the \( c_1 \) values in the rows are the same, then the apply component raises an error regardless of the setting for the `allow_duplicate_rows` apply component parameter.

### See Also:

- [SET_PARAMETER Procedure](#) for more information about the `allow_duplicate_rows` apply component parameter

The `SET_KEY_COLUMNS` Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

The `SET_KEY_COLUMNS` Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

The `SET_KEY_COLUMNS` Procedure and CDBs

This procedure defines the columns that are used as a substitute primary key. You must perform the `SET_KEY_COLUMNS` procedure in the appropriate PDB.

### 20.4.22 SET_PARAMETER Procedure

This procedure sets an apply parameter to the specified value.

#### Syntax

```sql
DBMS_APPLY_ADM.SET_PARAMETER (apply_name IN VARCHAR2,
                            parameter IN VARCHAR2,
                            value IN VARCHAR2 DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The apply component name. Do not specify an owner.</td>
</tr>
<tr>
<td>parameter</td>
<td>The name of the parameter you are setting.</td>
</tr>
<tr>
<td>value</td>
<td>The value to which the parameter is set.</td>
</tr>
<tr>
<td></td>
<td>If <code>NULL</code>, then the parameter is set to its default value.</td>
</tr>
</tbody>
</table>

#### Apply Component Parameters

The following table lists the parameters for an apply component.
### Table 20-25  Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow_duplicate_rows</td>
<td>Y or N</td>
<td>N</td>
<td>If Y and more than one row is changed by a single row logical change record (row LCR) with an UPDATE or DELETE command type, then the apply component only updates or deletes one of the rows. If N, then the apply component raises an error when it encounters a single row LCR with an UPDATE or DELETE command type that changes more than one row in a table. Note: Regardless of the setting for this parameter, apply components do not allow changes to duplicate rows for tables with LOB, LONG, or LONG RAW columns. See Also: “Usage Notes” and “Duplicate Rows and Substitute Primary Key Columns”</td>
</tr>
<tr>
<td>apply_sequence_nextval</td>
<td>Y or N</td>
<td>N</td>
<td>Controls whether the apply component checks and adjusts sequence values. If Y, then the apply component checks and adjusts sequence values. For ascending sequences, setting this parameter to Y ensures that the destination sequence values are equal to or greater than the source sequence values. For descending sequences, setting this parameter to Y ensures that the destination sequence values are equal to or less than the source sequence values. If N, then the apply component does not check or adjust sequence values. Note: This parameter is intended for XStream. Do not set this parameter to Y for an apply process in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See “ENABLE_GG_XSTREAM_FOR_STREAMS Procedure” for information about enabling XStream optimizations. See Also: SET_PARAMETER Procedure for information about the capture_sequence_nextval capture process parameter</td>
</tr>
</tbody>
</table>
Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| batchsql_mode  | DEPENDENT, DEPENDENT_EAGER, or SEQUENTIAL | DEPENDENT | Determines the batching method used to generate batch transactions for reordering. This parameter can be set to one of the following:  
• DEPENDENT - Batch transactions in a dependency-aware manner to minimize cross-batch dependencies and improve parallel processing performance in Oracle GoldenGate BATCHSQL mode. An executing batch has no unresolved dependencies.  
• DEPENDENT_EAGER - Batch transactions in a dependency-aware manner to minimize cross-batch dependencies and improve parallel processing performance in Oracle GoldenGate BATCHSQL mode. A batch can be executed when there are unresolved dependencies. The apply server waits for dependencies to be resolved before executing an LCR.  
• SEQUENTIAL - Batch transactions in a sequential manner. COMMIT SERIALIZATION = FULL runs in this mode regardless of the Oracle GoldenGate BATCHSQL mode.  

Note: This parameter is intended for Oracle GoldenGate. Do not use this parameter in an Oracle Streams replication environment or in an XStream environment. |
| cdgranularity | ROW or COLGROUP | COLGROUP for XStream In ROW for Oracle GoldenGate | Specifies the conflict detection granularity. This parameter can be set to one of the following:  
• ROW - Conflict resolution is applied for all column groups if there is a conflict for any column group.  
• COLGROUP - Conflict resolution is applied for column groups that have a conflict. Conflict resolution is not applied for column groups that do not have a conflict.  

Note: This parameter is intended for XStream and Oracle GoldenGate. Do not set this parameter for an apply process in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See "ENABLE_GG_XSTREAM_FOR_STREAMS Procedure" for information about enabling XStream optimizations. |
Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| commit_serialization   | DEPENDENT_TRANSACTIONS or FULL   | DEPENDENT_TRANSACTIONS | The order in which applied transactions are committed. Apply servers can apply nondependent transactions at the destination database in an order that is different from the commit order at the source database. Dependent transactions are always applied at the destination database in the same order as they were committed at the source database. You control whether the apply servers can apply nondependent transactions in a different order at the destination database using the commit_serialization apply parameter. This parameter has the following settings:
  
  - DEPENDENT_TRANSACTIONS - The apply component can commit nondependent transactions in any order. Performance is best if you specify DEPENDENT_TRANSACTIONS.
  - FULL - The apply component commits applied transactions in the order in which they were committed at the source database.

Regardless of the specification, applied transactions can execute in parallel subject to data dependencies and constraint dependencies.

If you specify DEPENDENT_TRANSACTIONS, then a destination database might commit changes in a different order than the source database. For example, suppose two nondependent transactions are committed at the source database in the following order:

1. Transaction A
2. Transaction B

At the destination database, these transactions might be committed in the opposite order:

1. Transaction B
2. Transaction A

If you specify DEPENDENT_TRANSACTIONS and there are application constraints that are not enforced by the database, then use virtual dependency definitions or add RELY constraints to account for the application constraints. See Oracle Streams Concepts and Administration for information about virtual dependency definitions and Oracle.
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compare_key_only</code></td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then disables automatic conflict detection and only uses primary and unique key columns to identify the table row for a row LCR. If N, then enables automatic conflict detection and uses all of the old values in a row LCR to identify the table row for a row LCR. Note: The <code>COMPARE_OLD_VALUES</code> procedure in this package can disable comparison of old values for specified columns during apply. See <code>COMPARE_OLD_VALUES Procedure</code>. See Also: &quot;Usage Notes&quot; and Oracle Streams Replication Administrator’s Guide for information about automatic conflict detection.</td>
</tr>
<tr>
<td><code>compute_lcr_dep_on_arrival</code></td>
<td>Y or N</td>
<td>N</td>
<td>If Y, the dependencies are computed as the LCRs for the transaction are received. If N, the dependencies are computed only after all the LCRs for a transaction are received. If the target table has all of the same constraints as the source table, you can improve the performance by setting this parameter to Y. If the number of LCRs in transaction exceeds the value of the number of the eager size parameter, then the dependencies for that transaction are calculated on arrival regardless of the setting of <code>compute_lcr_dep_on_arrival</code>. Note: This parameter is intended for XStream. Do not set this parameter to Y for an apply process in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Possible Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>disable_on_error</td>
<td>Y or N</td>
<td>Y</td>
<td>If Y, then the apply component is disabled on the first unresolved error, even if the error is not irrecoverable. If N, then the apply component continues regardless of unresolved errors.</td>
</tr>
<tr>
<td>disable_on_limit</td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then the apply component is disabled if the apply component terminates because it reached a value specified by the time_limit parameter or transaction_limit parameter. If N, then the apply component is restarted immediately after stopping because it reached a limit. When an apply component is restarted, it gets a new session identifier, and the processes associated with the apply component also get new session identifiers. However, the coordinator process number (APnn) remains the same.</td>
</tr>
</tbody>
</table>
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eager_size</td>
<td>A positive integer</td>
<td>9500</td>
<td>The apply component usually waits until it receives a commit record before starting to apply changes of a transaction. If XStream is enabled and more than eager_size LCRs arrive for a given transaction, then apply starts processing the changes. If XStream is not enabled and more than eager_size LCRs arrive for a given transaction, then apply waits until the complete transaction is received before processing the changes. Since it is possible that all existing apply servers are handling complete transactions from the source, additional apply servers are automatically created to handle outstanding eager transactions. The apply parameter max_parallelism limits the maximum number of apply servers that can be used for an apply process. This apply parameter is relevant only if its value is less than the value of the txn_lcr_spill_threshold apply parameter. When the value of txn_lcr_spill_threshold is lower than the value of eager_size, transactions spill to disk before eager apply begins. <strong>Note:</strong> This parameter is intended for XStream. Do not set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations.</td>
</tr>
<tr>
<td>enable_xstream_table_stats</td>
<td>Y or N</td>
<td>Y</td>
<td>When this parameter is set to Y, statistics about the operations of applied transactions are collected and made available in the V$XSTREAM_TABLE_STATS view. When this parameter is set to N, no statistics are collected. <strong>Note:</strong> This parameter is intended for XStream. Do not set this parameter to Y for an apply process in an Oracle Streams replication environment.</td>
</tr>
</tbody>
</table>
Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>excludetag</td>
<td>Comma-delimited list of Oracle Streams tags</td>
<td>NULL</td>
<td>Controls whether the capture process for an outbound server captures DML changes that are tagged with one of the specified Oracle Streams tags. Whether the capture process captures these changes depends on the settings for the getapplops and getreplicates parameters. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Streams replication environment. <strong>See Also:</strong> &quot;Usage Notes&quot; for the DBMS_CAPTURE_ADM.SET_PARAMETER procedure for more information about this parameter.</td>
</tr>
<tr>
<td>excludetrans</td>
<td>Comma-delimited list of transaction names</td>
<td>NULL</td>
<td>Controls whether the capture process for an outbound server captures DML changes in the specified transaction names. Whether the capture process captures these changes depends on the settings for the getapplops and getreplicates parameters. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Streams replication environment. <strong>See Also:</strong> &quot;Usage Notes&quot; for the DBMS_CAPTURE_ADM.SET_PARAMETER procedure for more information about this parameter.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Possible Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>excludeuser</td>
<td>Comma-delimited list of user names</td>
<td>NULL</td>
<td>Controls whether the capture process for an outbound server captures DML changes made by the specified users. Whether the capture process captures these changes depends on the settings for the <code>getapplops</code> and <code>getreplicates</code> parameters. Specify an exact pattern match for each user name. The pattern match is case sensitive. For example, specify <code>HR</code> for the <code>hr</code> user. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Streams replication environment. <strong>See Also:</strong> &quot;Usage Notes&quot; for the <code>DBMS_CAPTURE_ADM.SET_PARAMETER</code> procedure for more information about this parameter.</td>
</tr>
<tr>
<td>excludeuserid</td>
<td>Comma-delimited list of user ID values</td>
<td>NULL</td>
<td>Controls whether the capture process for an outbound server captures data manipulation language (DML) changes made by the specified users. Whether the capture process captures these changes depends on the settings for the <code>getapplops</code> and <code>getreplicates</code> parameters. To view the user ID for a user, query the <code>USER_ID</code> column in the <code>ALL_USERS</code> data dictionary view. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Streams replication environment. <strong>See Also:</strong> &quot;Usage Notes&quot; for the <code>DBMS_CAPTURE_ADM.SET_PARAMETER</code> procedure for more information about this parameter.</td>
</tr>
</tbody>
</table>
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getapplops</code></td>
<td>Y or N</td>
<td>Y</td>
<td>If Y, then the capture process captures DML changes if the original user is not specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is not specified in the <code>excludetrans</code> parameter. If N, then the capture process ignores DML changes if the original user is not specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is not specified in the <code>excludetrans</code> parameter. In either case, the capture process captures a DML change only if it satisfies the capture process’s rule sets. When N is set for both <code>getapplops</code> and <code>getreplicates</code>, no data is captured. Note: This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Streams replication environment. <strong>See Also:</strong> &quot;Usage Notes&quot; for the <code>DBMS_CAPTURE_ADM.SET_PARAMETER</code> procedure for more information about this parameter.</td>
</tr>
<tr>
<td><code>getreplicates</code></td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then the capture process captures DML changes if the original user is specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is specified in the <code>excludetrans</code> parameter. If N, then the capture process ignores DML changes if the original user is specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is specified in the <code>excludetrans</code> parameter. In either case, the capture process captures a DML change only if it satisfies the capture process’s rule sets. When N is set for both <code>getapplops</code> and <code>getreplicates</code>, no data is captured. Note: This parameter is intended for an XStream Out environment in which multiple outbound servers use the same capture process. XStream inbound servers ignore this parameter. Do not set this parameter in an Oracle Streams replication environment. <strong>See Also:</strong> &quot;Usage Notes&quot; for the <code>DBMS_CAPTURE_ADM.SET_PARAMETER</code> procedure for more information about this parameter</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Possible Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>grouptransops</td>
<td>A positive integer from 1 to 10000</td>
<td>250</td>
<td>The minimum number of LCRs that can be grouped into a single transaction. The commit LCR for a transaction is not included in the LCR count for the transaction. This parameter enables an apply component to group LCRs from multiple transactions into a single transaction. The apply component groups only LCRs that are part of committed transactions. If a transaction has more LCRs than the setting for this parameter, then the transaction is applied as a single transaction. The apply component does not split a transaction into separate transactions. This parameter only takes effect if the parallelism parameter setting is 1. The grouptransops parameter is ignored if the parallelism parameter setting is greater than 1. <strong>Note:</strong> This parameter is intended for XStream outbound servers and inbound servers. An Oracle Streams apply process ignores this parameter unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| `handlecollisions` | `Y` or `N`      | `N`     | This parameter controls whether the apply component tries to resolve duplicate-record and missing-record errors when applying changes during data loading. This parameter should be set to `N` for normal replication activity. It should be set to `Y` only when data is being loaded (instantiated) and replication is enabled. If `Y`, then does the equivalent of `OVERWRITE` for `INSERT`, `UPDATE` and `DELETE` operations that get `ROW_EXISTS` errors, and ignores `UPDATE` and `DELETE` operations that get `ROW_MISSING` errors. Specifically, the apply component performs the following actions when this parameter is set to `Y`:
  - If the operation is an insert and the primary key or unique key exists, then the insert is converted to an update.
  - If the operation is an update that does not modify the primary key or unique key columns, and the row does not exist, then the change is ignored.
  - If the operation is an update that modifies the primary key or unique key columns, and the row does not exist, then the change is ignored.
  - If the operation is an update that modifies the primary key or unique key columns, and the a row with the new key values already exists, then delete the row with the old key values and replace the row with the new key values.
  - If the operation is a delete and the row does not exist, then the change is ignored.
If `N` then it disables the above settings. **Note:** This parameter is intended for an XStream In environment with one or more inbound servers. Do not set this parameter in an Oracle Streams replication environment. |
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignore_transaction</td>
<td>A valid source transaction ID or NULL</td>
<td>NULL</td>
<td>Instructs the apply component to ignore the specified transaction from the source database, effective immediately. Use caution when setting this parameter because ignoring a transaction might lead to data divergence between the source database and destination database. To ignore multiple transactions, specify each transaction in a separate call to the SET_PARAMETER procedure. The DBA_APPLY_PARAMETERS view displays a comma-delimited list of all transactions to be ignored. To clear the list of ignored transactions, run the SET_PARAMETER procedure and specify NULL for the ignore_transaction parameter. If NULL, then the apply component ignores this parameter. <strong>Note:</strong> An apply component ignores this parameter for transactions that were not captured by a capture process. <strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>maximum_scn</td>
<td>A valid SCN or INFINITE</td>
<td>INFINITE</td>
<td>The apply component is disabled before applying a transaction with a commit SCN greater than or equal to the value specified. If INFINITE, then the apply component runs regardless of the SCN value. <strong>Note:</strong> An apply component ignores this parameter for transactions that were not captured by a capture process. <strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_parallelism</td>
<td>A positive integer</td>
<td>50</td>
<td>Limits the maximum number of apply servers that can be used for an apply component. When the apply parallelism parameter is set greater than one, the apply component adds apply servers when necessary to process transactions until it reaches the limit set by this parameter (max_parallelism). Transactions include both unassigned (eager) and assigned transactions. Apply servers that idle for more than 5 minutes are shut down until the configured parallelism is attained. Runtime statistics for servers that have been shut down are aggregated into apply server 0 so that accurate apply statistics for the entire apply process can be maintained. <strong>Note:</strong> This parameter is intended for XStream. Do not set this parameter to Y for an apply process in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See “ENABLE_GG_XSTREAM_FOR_STREAMS Procedure” for information about enabling XStream optimizations.</td>
</tr>
</tbody>
</table>
### Table 20-25 (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_sga_size</td>
<td>A positive integer</td>
<td>INFINITE</td>
<td>Controls the amount of system global area (SGA) memory allocated specifically to the apply component, in megabytes. The memory is allocated for the duration of the apply component's session and is released when the apply component becomes disabled. <strong>Note:</strong> The sum of system global area (SGA) memory allocated for all components on a database must be less than the value set for the <code>STREAMS_POOL_SIZE</code> initialization parameter. If <code>NULL</code>, then the apply component uses the original default value. A <code>NULL</code> value has the same effect as resetting the parameter to its default value. <strong>Note:</strong> This parameter is intended for XStream. Do not use or attempt to set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See “ENABLE_GG_XSTREAM_FOR_STREAMS Procedure” for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>message_tracking_frequency</td>
<td>0 or a positive integer</td>
<td>2000000</td>
<td>The frequency at which messages applied by the inbound server are tracked automatically. For example, if this parameter is set to the default value of <code>2000000</code>, then every two-millionth message is tracked automatically. The tracking label used for automatic message tracking is <code>inbound_server_name:AUTOTRACK</code>, where <code>inbound_server_name</code> is the name of the inbound server. Only the first 20 bytes of the inbound server name are used; the rest is truncated if it exceeds 20 bytes. If <code>0</code> (zero), then no messages are tracked automatically. See Oracle Streams Replication Administrator's Guide for more information about message tracking.</td>
</tr>
</tbody>
</table>
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimize_progress_table</td>
<td>Y or N</td>
<td>Y</td>
<td>This parameter determines if a table or the redo log tracks the apply transactions. The progress table tracks completed apply transactions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If Y, transactions are tracked in the redo log.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If N, transactions are tracked in a table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the database is not in archivelog mode, then the setting of Y is ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note</strong>: This parameter is intended for XStream. Do not set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also</strong>: &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>optimize_self_updates</td>
<td>Y or N</td>
<td>Y</td>
<td>This parameter affects conflict resolution when an update in the source database sets a column to its existing value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When this parameter is set to Y, a conflict between the value in the LCR and the corresponding column in the target database is considered resolved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When this parameter is set to N, the conflict is processed.</td>
</tr>
</tbody>
</table>
### Table 20-25 (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallelism</td>
<td>A positive integer</td>
<td>4</td>
<td>The number of apply servers that can concurrently apply transactions. The reader server and the apply server process names are ASnn, where nn can include letters and numbers. The total number of ASnn processes is the value of the parallelism parameter plus one. For example, if parallelism is set to 4, then an apply component uses a total of five ASnn processes. In this case, there is one reader server and four apply servers. Setting the parallelism parameter to a number higher than the number of available operating system user processes can disable the apply component. Make sure the PROCESSES initialization parameter is set appropriately when you set the parallelism parameter. <strong>Note:</strong> When the value of this parameter is changed from 1 to a higher value for a running apply component, the apply component is stopped and restarted automatically. This can take some time depending on the size of the transactions currently being applied. When the value of this parameter is greater than 1, and the parameter value is decreased or increased, the apply component does not restart. <strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>
Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallelism_interval</td>
<td>0 or a positive integer</td>
<td>0</td>
<td>The parallelism interval is the interval in seconds at which the current workload activity is computed. The apply component calculates the mean throughput every 5 X parallelism_interval seconds. After each calculation, the apply component can increase or decrease the number of apply servers to try to improve throughput. If throughput is improved, then the apply component keeps the new number of apply servers. The parallelism interval is used only if the parallelism parameter is set to a value greater than one and the max_parallelism parameter value is greater than the parallelism parameter value. Note: This parameter is intended for an XStream In environment with one or more inbound servers. XStream outbound servers ignore this parameter. Do not set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. See Also: &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Possible Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>preserve_encryption</td>
<td>Y or N</td>
<td>Y</td>
<td>Whether to preserve encryption for columns encrypted using Transparent Data Encryption. If Y, then columns in tables at the destination database must be encrypted when corresponding columns in row LCRs are encrypted. If columns are encrypted in row LCRs but the corresponding columns are not encrypted in the tables at the destination database, then an error is raised when the apply component tries to apply the row LCRs. If N, then columns in tables at the destination database do not need to be encrypted when corresponding columns in row LCRs are encrypted. If columns are encrypted in row LCRs but the corresponding columns are not encrypted in the tables at the destination database, then the apply component applies the changes in the row LCRs. <strong>Note:</strong> When the value of this parameter is changed for a running apply component, the apply component is stopped and restarted automatically. This can take some time depending on the size of the transactions currently being applied. <strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>rtrim_on_implicit_conversion</td>
<td>Y or N</td>
<td>Y</td>
<td>Whether to remove blank padding from the right end of a column when automatic data type conversion is performed during apply. If Y, then blank padding is removed when a CHAR or NCHAR source column in a row LCR is converted to a VARCHAR2, NVARCHAR2, or CLOB column in a table. If N, then blank padding is preserved in the column. <strong>See Also:</strong> &quot;Usage Notes&quot; and Oracle Streams Concepts and Administration for information about automatic data type conversion during apply</td>
</tr>
<tr>
<td>startup_seconds</td>
<td>0, a positive integer, or INFINITE</td>
<td>0</td>
<td>The maximum number of seconds to wait for another instantiation of the same apply component to finish. If the other instantiation of the same apply component does not finish within this time, then the apply component does not start. If INFINITE, then an apply component does not start until another instantiation of the same apply component finishes. <strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>
Table 20-25 (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>suppresstriggers</td>
<td>Y or N</td>
<td>Y</td>
<td>This parameter controls whether triggers fire when a change is made by the apply component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If Y, triggers do not fire for changes made by the apply component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If N, triggers fire for changes made by the apply component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If a trigger's firing property is set to always fire, then the trigger always fires for changes made by the apply component, regardless of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the value of the suppresstriggers parameter. A trigger's firing property is set to always fire by running the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DBMS_DDL.SET_TRIGGER_FIRING_PROPERT Y procedure with the fire_once parameter set to FALSE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: This parameter is intended for an XStream In environment with one or more inbound servers. Do not set this parameter in an Oracle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Streams replication environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>See Also: &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>time_limit</td>
<td>A positive</td>
<td>INFINITE</td>
<td>The apply component stops as soon as possible after the specified number of seconds since it started.</td>
</tr>
<tr>
<td></td>
<td>integer or</td>
<td></td>
<td>If INFINITE, then the apply component continues to run until it is stopped explicitly.</td>
</tr>
<tr>
<td></td>
<td>INFINITE</td>
<td></td>
<td>See Also: &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>trace_level</td>
<td>0 or a positive</td>
<td>0</td>
<td>Set this parameter only under the guidance of Oracle Support Services.</td>
</tr>
<tr>
<td></td>
<td>integer</td>
<td></td>
<td>See Also: &quot;Usage Notes&quot;</td>
</tr>
<tr>
<td>transaction_limit</td>
<td>A positive</td>
<td>INFINITE</td>
<td>The apply component stops after applying the specified number of transactions.</td>
</tr>
<tr>
<td></td>
<td>integer or</td>
<td></td>
<td>If INFINITE, then the apply component continues to run regardless of the number of transactions applied.</td>
</tr>
<tr>
<td></td>
<td>INFINITE</td>
<td></td>
<td>See Also: &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>txn_age_spill_threshold</td>
<td>A positive integer or INFINITE</td>
<td>900</td>
<td>The apply component begins to spill messages from memory to hard disk for a particular transaction when the amount of time that any message in the transaction has been in memory exceeds the specified number. The parameter specifies the age in seconds. When the reader server spills messages from memory, the messages are stored in a database table on the hard disk. These messages are not spilled from memory to a queue table. Message spilling occurs at the transaction level. For example, if this parameter is set to 900, and the reader server of an apply component detects that one message in a transaction has been in memory longer than 900 seconds, then all of the messages in the transaction spill from memory to hard disk. If INFINITE, then the apply component does not spill messages to the hard disk based on the age of the messages. Query the DBA_APPLY_SPILL_TXN data dictionary view for information about transactions spilled by an apply component. See Also: &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>
### Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>txn_lcr_spill_threshold</td>
<td>A positive integer or INFINITE</td>
<td>10000</td>
<td>The apply component begins to spill messages from memory to hard disk for a particular transaction when the number of messages in memory for the transaction exceeds the specified number. The number of messages in first chunk of messages spilled from memory equals the number specified for this parameter, and the number of messages spilled in future chunks is either 100 or the number specified for this parameter, whichever is less. If the reader server of an apply component has the specified number of messages in memory for a particular transaction, then when it detects the next message for this transaction, it spills the messages that are in memory to the hard disk. For example, if this parameter is set to 10000, and a transaction has 10,200 messages, then the reader server handles the transaction in the following way:&lt;br&gt;1. Reads the first 10,000 messages in the transaction into memory&lt;br&gt;2. Spills messages 1 - 10,000 to hard disk when it detects message 10,000&lt;br&gt;3. Reads the next 100 messages in the transaction into memory&lt;br&gt;4. Spills messages 10,001 - 10,100 to hard disk when it detects message 10,100&lt;br&gt;5. Reads the next 100 messages in the transaction into memory&lt;br&gt;The apply component applies the first 10,100 messages from the hard disk and the last 100 messages from memory. When the reader server spills messages from memory, the messages are stored in a database table on the hard disk. These messages are not spilled from memory to a queue table. Message spilling occurs at the transaction level. For example, if this parameter is set to 10000, and the reader server of an apply component is assembling two transactions, one with 7,500 messages and another with 8,000 messages, then it does not spill any messages. If INFINITE, then the apply component does not spill messages to the hard disk based on the number of messages in a transaction.</td>
</tr>
</tbody>
</table>
Table 20-25  (Cont.) Apply Component Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write_alert_log</td>
<td>Y or N</td>
<td>Y</td>
<td>Query the DBA_APPLY_SPILL_TXN data dictionary view for information about transactions spilled by an apply component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>See Also:</strong> &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>

- If **Y**, then the apply component writes a message to the alert log on exit.
- If **N**, then the apply component does not write a message to the alert log on exit.
The message specifies the reason why the apply component stopped.

**Usage Notes**

The following usage notes apply to this procedure:

- Delays Are Possible Before New Parameter Settings Take Effect
- Parameters Interpreted as Positive Integers
- Parameters with a System Change Number (SCN) Setting
- The SET_PARAMETER Procedure and Streams
- The SET_PARAMETER Procedure and XStream Outbound Servers
- The SET_PARAMETER Procedure and XStream Inbound Servers

**Delays Are Possible Before New Parameter Settings Take Effect**

When you alter a parameter value, a short amount of time might pass before the new value for the parameter takes effect.

**Parameters Interpreted as Positive Integers**

For all parameters that are interpreted as positive integers, the maximum possible value is 4,294,967,295. Where applicable, specify INFINITE for larger values.

**Parameters with a System Change Number (SCN) Setting**

For parameters that require an SCN setting, any valid SCN value can be specified.

**The SET_PARAMETER Procedure and Streams**

You can use the following parameters in Streams if you enable XStream performance optimizations for Oracle Streams using the procedure DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS:

- apply_sequence_nextval
- compute_lcr_dep_on_arrival
- eager_size
- grouptransops
- max_parallelism
• max_sga_size
• optimize_progress_table
• parallelism_interval

The SET_PARAMETER Procedure and XStream Outbound Servers

Outbound servers ignore the settings for the following apply parameters:

• allow_duplicate_rows
• commit_serialization
• compare_key_only
• compute_lcr_dep_on_arrival
• disable_on_error
• eager_size
• enable_xstream_table_stats
• grouptransops
• handlecollisions
• optimize_self_updates
• parallelism
• parallelism_interval
• preserve_encryption
• rtrim_on_implicit_conversion
• suppress_triggers

The commit_serialization parameter is always set to FULL for an outbound server, and the parallelism parameter is always set to 1 for an outbound server.

You can use the other apply parameters with outbound servers.

---

**Note:**

Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

The SET_PARAMETER Procedure and XStream Inbound Servers

Inbound servers ignore the settings for the following apply parameters:

• excludetag
• excludetrans
• excludeuser
• excludeuserid
• getapplops
• getreplicates
• ignore_transaction
• maximum_scn

You can use all of the other apply component parameters with inbound servers.

The default setting for the compare_key_only parameter for an inbound server is `Y`.

The default setting for the parallelism parameter for an inbound server is 4.

---

**Note:**

Using XStream requires purchasing a license for the Oracle GoldenGate product.
See *Oracle Database XStream Guide*.

---

20.4.23 SET_REPERROR_HANDLER Procedure

This procedure specifies how a particular error is handled based on its error number.

You can choose between several predefined actions for a given error.

**Syntax**

```sql
DBMS_APPLY_ADM.SET_REPERROR_HANDLER(
    apply_name     IN  VARCHAR2,
    object         IN  VARCHAR2,
    error_number   IN  NUMBER,
    method         IN  VARCHAR2,
    source_object  IN  VARCHAR2  DEFAULT NULL,
    max_retries    IN  NUMBER    DEFAULT NULL,
    delay_csecs    IN  NUMBER    DEFAULT 6000);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The name of the apply process.</td>
</tr>
<tr>
<td>object</td>
<td>The schema and name of the target table, specified as <code>[schema name.]table name</code> for which an error handler is being added, modified, or removed. The table must exist. For example, if an update conflict handler is being added for table <code>employees</code> owned by user <code>hr</code>, then specify <code>hr.employees</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>error_number</td>
<td>The error handling number. If 0, then use the default for all error handling for <code>object</code>.</td>
</tr>
</tbody>
</table>
Table 20-26  (Cont.) SET_REPERROR_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Specifies the action to take when the given error_number occurs.</td>
</tr>
<tr>
<td></td>
<td>If NULL, remove the error handler for error_number</td>
</tr>
<tr>
<td></td>
<td>The possible actions are:</td>
</tr>
<tr>
<td></td>
<td>• ABEND: Stop the apply process when the error occurs.</td>
</tr>
<tr>
<td></td>
<td>• RECORD: Move the LCR to the error queue when the error is encountered.</td>
</tr>
<tr>
<td></td>
<td>• IGNORE: Silently ignore the error and do not apply the LCR.</td>
</tr>
<tr>
<td></td>
<td>• RETRY: Retry the LCR max_retries times.</td>
</tr>
<tr>
<td></td>
<td>• RETRY_TRANSACTION: Retry the transaction max_retries times. Wait delay_csecs centiseconds before each retry.</td>
</tr>
<tr>
<td>source_object</td>
<td>The schema and object name of the source table, specified as [schema_name.]table_name for the table where the change originated. For example, if the change originated at the employees table owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>max_retries</td>
<td>Maximum number of times to retry for RETRY and RETRY_TRANSACTION actions in method. Must be specified with either the RETRY or RETRY_TRANSACTION action.</td>
</tr>
<tr>
<td>delay_csecs</td>
<td>The number of centiseconds between retries for RETRY and RETRY_TRANSACTION action in method.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- **Priority of Error Handlers**

**Priority of Error Handlers**

Any conflict handling specified by SET_UPDATE_CONFLICT_HANDLER or SET_DML_CONFLICT_HANDLER is tried before the actions specified by SET_REPERROR_HANDLER. The PL/SQL procedure specified by SET_DML_HANDLER is called to handle the error if none of the previously mentioned methods resolve it.

**20.4.24 SET_SCHEMA_INSTANTIATION_SCN Procedure**

This procedure records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database. This procedure overwrites any existing instantiation SCN for the schema, and, if it sets the instantiation SCN for a table, it overwrites any existing instantiation SCN for the table.

This procedure gives you precise control over which DDL logical change records (LCRs) for a schema are ignored and which DDL LCRs are applied by an apply component.
Syntax

DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN(
    source_schema_name    IN  VARCHAR2,
    source_database_name  IN  VARCHAR2,
    instantiation_scn     IN  NUMBER,
    apply_database_link   IN  VARCHAR2  DEFAULT NULL,
    recursive             IN  BOOLEAN   DEFAULT FALSE,
    source_root_name      IN  VARCHAR2  DEFAULT NULL);

Parameters

Table 20-27 SET_SCHEMA_INSTANTIATION_SCN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_schema_name</td>
<td>The name of the source schema. For example, hr.</td>
</tr>
<tr>
<td></td>
<td>When setting an instantiation SCN for schema, always specify the name of the</td>
</tr>
<tr>
<td></td>
<td>schema at the source database, even if a rule-based transformation or apply</td>
</tr>
<tr>
<td></td>
<td>handler is configured to change the schema name.</td>
</tr>
<tr>
<td>source_database_name</td>
<td>The global name of the source database. For example, DBS1.NET.</td>
</tr>
<tr>
<td></td>
<td>If you do not include the domain name, then the procedure appends it to the</td>
</tr>
<tr>
<td></td>
<td>database name automatically. For example, if you specify DBS1 and the</td>
</tr>
<tr>
<td></td>
<td>domain is NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>instantiation_scn</td>
<td>The instantiation SCN. Specify NULL to remove the instantiation SCN</td>
</tr>
<tr>
<td></td>
<td>metadata for the source schema from the data dictionary.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>The name of the database link to a non-Oracle database. This parameter</td>
</tr>
<tr>
<td></td>
<td>should be set only when the destination database of a local apply component</td>
</tr>
<tr>
<td></td>
<td>is a non-Oracle database.</td>
</tr>
<tr>
<td>recursive</td>
<td>If TRUE, then the procedure sets the instantiation SCN for the specified</td>
</tr>
<tr>
<td></td>
<td>schema and all tables owned by the schema in the source database. This</td>
</tr>
<tr>
<td></td>
<td>procedure selects the tables owned by the specified schema from the ALL</td>
</tr>
<tr>
<td></td>
<td>TABLES data dictionary view at the source database under the security</td>
</tr>
<tr>
<td></td>
<td>context of the current user.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure sets the instantiation SCN for specified</td>
</tr>
<tr>
<td></td>
<td>schema, but does not set the instantiation SCN for any tables.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If <code>recursive</code> is set to TRUE, then a database link from the</td>
</tr>
<tr>
<td></td>
<td>destination database to the source database is required. This database link</td>
</tr>
<tr>
<td></td>
<td>must have the same name as the global name of the source database and must</td>
</tr>
<tr>
<td></td>
<td>be accessible to the current user. Also, a table must be accessible to the</td>
</tr>
<tr>
<td></td>
<td>current user in either the ALL TABLES or DBA TABLES data dictionary view at</td>
</tr>
<tr>
<td></td>
<td>the source database for this procedure to set the instantiation SCN for the</td>
</tr>
<tr>
<td></td>
<td>table at the destination database.</td>
</tr>
<tr>
<td>source_root_name</td>
<td>The global name of the source root database.</td>
</tr>
<tr>
<td></td>
<td>In a non-CDB, this parameter must be NULL.</td>
</tr>
<tr>
<td></td>
<td>In a CDB, both <code>source_database</code> and <code>source_root_name</code> must be specified to</td>
</tr>
<tr>
<td></td>
<td>identify a specific container.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:
• The SET_SCHEMA_INSTANTIATION_SCN Procedure and LCRs
• Instantiation SCNs and DDL LCRs
• The recursive Parameter
• The SET_SCHEMA_INSTANTIATION_SCN Procedure and XStream Outbound Servers
• The SET_SCHEMA_INSTANTIATION_SCN Procedure and XStream Inbound Servers
• The SET_SCHEMA_INSTANTIATION_SCN Procedure and CDBs

See Also:

• SET_GLOBAL_INSTANTIATION_SCN Procedure
• SET_TABLE_INSTANTIATION_SCN Procedure
• LCR$_DDL.Record Type for more information about DDL LCRs
• Oracle Streams Replication Administrator's Guide

The SET_SCHEMA_INSTANTIATION_SCN Procedure and LCRs

Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

Instantiation SCNs and DDL LCRs

If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at a destination database, then the apply component at the destination database disregards the DDL LCR. Otherwise, the apply component applies the DDL LCR.

The schema instantiation SCN specified by this procedure is used on the following types of DDL LCRs:

• DDL LCRs with a command_type of CREATE TABLE
• DDL LCRs with a non-NULL object_owner specified and neither base_table_owner nor base_table_name specified.

For example, the schema instantiation SCN set by this procedure is used for a DDL LCR with a command_type of CREATE TABLE and ALTER USER.

The schema instantiation SCN specified by this procedure is not used for DDL LCRs with a command_type of CREATE USER. A global instantiation SCN is needed for such DDL LCRs.

The recursive Parameter

If the recursive parameter is set to TRUE, then this procedure sets the table instantiation SCN for each table at the source database owned by the schema. This procedure uses the SET_TABLE_INSTANTIATION_SCN procedure to set the instantiation SCN for each table. Each table instantiation SCN is used for DDL LCRs and row LCRs on the table.
If the `recursive` parameter is set to `FALSE`, then this procedure does not set the instantiation SCN for any tables.

The `SET_SCHEMA_INSTANTIATION_SCN` Procedure and XStream Outbound Servers

Instantiation SCNs are not required for database objects processed by an outbound server. If an instantiation SCN is set for a database object, then the outbound server only sends the LCRs for the database object with SCN values that are greater than the instantiation SCN value. If a database object does not have an instantiation SCN set, then the outbound server skips the instantiation SCN check and sends all LCRs for that database object. In both cases, the outbound server only sends LCRs that satisfy its rule sets.

The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

See Also:

Oracle Database XStream Guide for more information about outbound servers and instantiation SCNs

The `SET_SCHEMA_INSTANTIATION_SCN` Procedure and XStream Inbound Servers

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.

The `SET_SCHEMA_INSTANTIATION_SCN` Procedure and CDBs

In a CDB, this procedure must be invoked from the same container as the apply process that uses the instantiation SCN information.

20.4.25 `SET_TABLE_INSTANTIATION_SCN` Procedure

This procedure records the specified instantiation SCN for the specified table in the specified source database. This procedure overwrites any existing instantiation SCN for the particular table.

This procedure gives you precise control over which logical change records (LCRs) for a table are ignored and which LCRs are applied by an apply component.

Syntax

```sql
DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
  source_object_name    IN  VARCHAR2,
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL,
  source_root_name      IN  VARCHAR2  DEFAULT NULL);
```
Parameters

Table 20-28  SET_TABLE_INSTANTIATION_SCN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_object_name</td>
<td>The name of the source object specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>When setting an instantiation SCN for a database object, always specify the name of the schema and database object at the source database, even if a rule-based transformation or apply handler is configured to change the schema name or database object name.</td>
</tr>
<tr>
<td>source_database_name</td>
<td>The global name of the source database. For example, DBS1.NET. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>instantiation_scn</td>
<td>The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source table from the data dictionary.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This parameter must be NULL when the procedure is invoked from the root of a CDB.</td>
</tr>
<tr>
<td>source_root_name</td>
<td>The global name of the source root database.</td>
</tr>
<tr>
<td></td>
<td>In a non-CDB, this parameter must be NULL.</td>
</tr>
<tr>
<td></td>
<td>In a CDB, both source_database and source_root_name must be specified to identify a specific container.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- Instantiation SCNs and LCRs
- The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Outbound Servers
- The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Inbound Servers
- The SET_TABLE_INSTANTIATION_SCN Procedure and CDBs

Instantiation SCNs and LCRs

If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at some destination database, then the apply component at the destination database disregards the LCR. Otherwise, the apply component applies the LCR.

The table instantiation SCN specified by this procedure is used on the following types of LCRs:
• Row LCRs for the table
• DDL LCRs that have a non-NULL base_table_owner and base_table_name specified, except for DDL LCRs with a command_type of CREATE TABLE

For example, the table instantiation SCN set by this procedure is used for DDL LCRs with a command_type of ALTER TABLE or CREATE TRIGGER.

Note:
The instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

See Also:
• SET_GLOBAL_INSTANTIATION_SCN Procedure
• SET_SCHEMA_INSTANTIATION_SCN Procedure
• LCR$_ROW_RECORD Type for more information about row LCRs
• LCR$_DDL_RECORD Type for more information about DDL LCRs
• Oracle Streams Replication Administrator's Guide

The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Outbound Servers
Instantiation SCNs are not required for database objects processed by an outbound server. If an instantiation SCN is set for a database object, then the outbound server only sends the LCRs for the database object with SCN values that are greater than the instantiation SCN value. If a database object does not have an instantiation SCN set, then the outbound server skips the instantiation SCN check and sends all LCRs for that database object. In both cases, the outbound server only sends LCRs that satisfy its rule sets.

The apply_database_link parameter must be set to NULL or to the local database for this procedure to set an instantiation SCN for an outbound server.

See Also:
Oracle Database XStream Guide for more information about outbound servers and instantiation SCNs

The SET_TABLE_INSTANTIATION_SCN Procedure and XStream Inbound Servers
Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.

The SET_TABLE_INSTANTIATION_SCN Procedure and CDBs
In a CDB, this procedure must be invoked from the same container as the apply process that uses the instantiation SCN information.
20.4.26 SET_UPDATE_CONFLICT_HANDLER Procedure

This procedure adds, modifies, or removes a prebuilt update conflict handler for the specified object.

Syntax

DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name          IN  VARCHAR2,
    method_name          IN  VARCHAR2,
    resolution_column    IN  VARCHAR2,
    column_list          IN  DBMS_UTILITY.NAME_ARRAY,
    apply_database_link  IN  VARCHAR2  DEFAULT NULL);

Parameters

Table 20-29  SET_UPDATE_CONFLICT_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The schema and name of the table, specified as [schema_name.]object_name, for which an update conflict handler is being added, modified, or removed. For example, if an update conflict handler is being added for table employees owned by user hr, then specify hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>method_name</td>
<td>Type of update conflict handler to create. You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained:</td>
</tr>
<tr>
<td></td>
<td>• MAXIMUM: Applies the column list from the source database if it has the greater value for the resolution column. Otherwise, retains the values at the destination database.</td>
</tr>
<tr>
<td></td>
<td>• MINIMUM: Applies the column list from the source database if it has the lesser value for the resolution column. Otherwise, retains the values at the destination database.</td>
</tr>
<tr>
<td></td>
<td>• OVERWRITE: Applies the column list from the source database, overwriting the column values at the destination database.</td>
</tr>
<tr>
<td></td>
<td>• DISCARD: Retains the column list from the destination database, discarding the column list from the source database.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>If NULL, then the procedure removes any existing update conflict handler with the same object_name, resolution_column, and column_list. If non-NULL, then the procedure replaces any existing update conflict handler with the same object_name and resolution_column.</td>
</tr>
</tbody>
</table>
Table 20-29  (Cont.) SET_UPDATE_CONFLICT_HANDLER Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolution_column</td>
<td>Name of the column used to uniquely identify an update conflict handler. For the MAXIMUM and MINIMUM prebuilt methods, the resolution column is also used to resolve the conflict. The resolution column must be one of the columns listed in the column_list parameter. NULL is not allowed for this parameter. For the OVERWRITE and DISCARD prebuilt methods, you can specify any column in the column list.</td>
</tr>
<tr>
<td>column_list</td>
<td>List of columns for which the conflict handler is called. The same column cannot be in more than one column list. If a conflict occurs for one or more of the columns in the list when an apply component tries to apply a row logical change record (row LCR), then the conflict handler is called to resolve the conflict. The conflict handler is not called if a conflict occurs only for columns that are not in the list. <strong>Note:</strong> Prebuilt update conflict handlers do not support LOB, LONG, LONG RAW, user-defined type, and Oracle-supplied type columns. Therefore, you should not include these types of columns in the column_list parameter.</td>
</tr>
<tr>
<td>apply_database_link</td>
<td>The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database. <strong>Note:</strong> Currently, conflict handlers are not supported when applying changes to a non-Oracle database.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The following usage notes apply to this procedure:

- Modifying an Existing Update Conflict Handler
- Removing an Existing Update Conflict Handler
- Series of Actions for Conflicts
- Procedure DML Handlers for Conflicts
- A Column Can Be in Only One Column List
- Update Conflict Handlers and Non-Oracle Databases
- The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Outbound Servers
- The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Inbound Servers

**See Also:**

*Oracle Streams Replication Administrator's Guide* for more information about prebuilt and custom update conflict handlers
Modifying an Existing Update Conflict Handler

If you want to modify an existing update conflict handler, then you specify the table and resolution column of an the existing update conflict handler. You can modify the prebuilt method or the column list.

Removing an Existing Update Conflict Handler

If you want to remove an existing update conflict handler, then specify NULL for the prebuilt method and specify the table, column list, and resolution column of the existing update conflict handler.

Series of Actions for Conflicts

If an update conflict occurs, then Oracle completes the following series of actions:

1. Calls the appropriate update conflict handler to resolve the conflict
2. If no update conflict handler is specified or if the update conflict handler cannot resolve the conflict, then calls the appropriate error handler for the apply component, table, and operation to handle the error
3. If no error handler is specified or if the error handler cannot resolve the error, then raises an error and moves the transaction containing the row LCR that caused the error to the error queue

See Also:

"Signature of a DML Handler Procedure or Error Handler Procedure" for information about setting an error handler

Procedure DML Handlers for Conflicts

If you cannot use a prebuilt update conflict handler to meet your requirements, then you can create a PL/SQL procedure to use as a custom conflict handler. You use the SET_DML_HANDLER procedure to designate one or more custom conflict handlers for a particular table. In addition, a custom conflict handler can process LOB columns and use LOB assembly.

See Also:

SET_DML_HANDLER Procedure

A Column Can Be in Only One Column List

When a column is in a column list, and you try to add the same column to another column list, this procedure returns the following error:

ORA-00001: UNIQUE CONSTRAINT (SYS.APPLY$_CONF_HDLR_COLUMNS_UNQ1) VIOLATED

Update Conflict Handlers and Non-Oracle Databases

Setting an update conflict handler for an apply component that is applying to a non-Oracle database is not supported.
The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Outbound Servers
This procedure has no effect on XStream outbound servers.

The SET_UPDATE_CONFLICT_HANDLER Procedure and XStream Inbound Servers
This procedure functions the same way for apply processes and inbound servers.

Examples
The following is an example for setting an update conflict handler for the employees table in the hr schema:

```sql
DECLARE
    cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'salary';
    cols(2) := 'commission_pct';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
        object_name => 'hr.employees',
        method_name => 'MAXIMUM',
        resolution_column => 'salary',
        column_list => cols);
END;
/
```

This example sets a conflict handler that is called if a conflict occurs for the salary or commission_pct column in the hr.employees table. If such a conflict occurs, then the salary column is evaluated to resolve the conflict. If a conflict occurs only for a column that is not in the column list, such as the job_id column, then this conflict handler is not called.

20.4.27 SET_VALUE_DEPENDENCY Procedure
This procedure sets or removes a value dependency. A value dependency is a virtual dependency definition that defines a relationship between the columns of two or more tables.

An apply component uses the name of a value dependencies to detect dependencies between row logical change records (row LCRs) that contain the columns defined in the value dependency. Value dependencies can define virtual foreign key relationships between tables, but, unlike foreign key relationships, value dependencies can involve more than two database objects.

This procedure is overloaded. The attribute_list and attribute_table parameters are mutually exclusive.

See Also:
Oracle Streams Concepts and Administration

Syntax

```sql
DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY(
    dependency_name IN VARCHAR2,
    object_name IN VARCHAR2,
    attribute_list IN VARCHAR2);
```
DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY(
  dependency_name IN VARCHAR2,
  object_name IN VARCHAR2,
  attribute_table IN DBMSUTILITY.NAME_ARRAY);

Parameters

Table 20-30  SET_VALUE_DEPENDENCY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependency_name</td>
<td>The name of the value dependency. If a dependency with the specified name does not exist, then it is created. If a dependency with the specified name exists, then the specified object and attributes are added to the dependency. If NULL, an error is raised.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the table, specified as [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default. If NULL and the specified dependency exists, then the dependency is removed. If NULL and the specified dependency does not exist, then an error is raised. If NULL, then attribute_list and attribute_table also must be NULL.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>A comma-delimited list of column names in the table. There must be no spaces between entries.</td>
</tr>
<tr>
<td>attribute_table</td>
<td>A PL/SQL associative array of type DBMSUTILITY.NAME_ARRAY that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be NULL terminated.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- The SET_VALUE_DEPENDENCY Procedure and XStream Outbound Servers
- The SET_VALUE_DEPENDENCY Procedure and XStream Inbound Servers

The SET_VALUE_DEPENDENCY Procedure and XStream Outbound Servers
This procedure has no effect on XStream outbound servers.

The SET_VALUE_DEPENDENCY Procedure and XStream Inbound Servers
This procedure functions the same way for apply processes and inbound servers.

20.4.28 START_APPLY Procedure

This procedure directs the apply component to start applying messages.

Syntax

DBMS_APPLY_ADM.START_APPLY(
  apply_name IN VARCHAR2);
### Parameter

#### Table 20-31 START_APPLY Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The apply component name. A NULL setting is not allowed. Do not specify an owner.</td>
</tr>
</tbody>
</table>

### Usage Notes

The following usage notes apply to this procedure:

- **Apply Component Status**
- **The START_APPLY Procedure and XStream Outbound Servers**
- **The START_APPLY Procedure and XStream Inbound Servers**

#### Apply Component Status

The apply component status is persistently recorded. Hence, if the status is **ENABLED**, then the apply component is started upon database instance startup. An apply component (an nn) is an Oracle background process. The enqueue and dequeue state of `DBMS_AQADM.START_QUEUE` and `DBMS_AQADM.STOP_QUEUE` have no effect on the start status of an apply component.

#### The START_APPLY Procedure and XStream Outbound Servers

This procedure functions the same way for apply processes and outbound servers.

#### The START_APPLY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

### 20.4.29 STOP_APPLY Procedure

This procedure stops the apply component from applying messages and rolls back any unfinished transactions being applied.

#### Syntax

```sql
DBMS_APPLY_ADM.STOP_APPLY(
    apply_name  IN  VARCHAR2,
    force       IN  BOOLEAN   DEFAULT FALSE);
```

#### Parameters

#### Table 20-32 STOP_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>The apply component name. A NULL setting is not allowed. Do not specify an owner.</td>
</tr>
</tbody>
</table>
### Usage Notes

The following usage notes apply to this procedure:

- **Apply Component Status**
- **Queue Subprograms Have No Effect on Apply Component Status**
- **The STOP_APPLY force Parameter and the commit_serialization Apply Parameter**
- **The STOP_APPLY Procedure and XStream Outbound Servers**
- **The STOP_APPLY Procedure and XStream Inbound Servers**

#### Apply Component Status

The apply component status is persistently recorded. Hence, if the status is DISABLED or ABORTED, then the apply component is not started upon database instance startup.

#### Queue Subprograms Have No Effect on Apply Component Status

The enqueue and dequeue state of `DBMS_AQADM.START_QUEUE` and `DBMS_AQADM.STOP_QUEUE` have no effect on the STOP status of an apply component.

#### The STOP_APPLY force Parameter and the commit_serialization Apply Parameter

The following table describes apply component behavior for each setting of the force parameter in the STOP_APPLY procedure and the commit_serialization apply component parameter. In all cases, the apply component rolls back any unfinished transactions when it stops.

<table>
<thead>
<tr>
<th>force</th>
<th>commit_serialization</th>
<th>Apply Component Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FULL</td>
<td>The apply component stops immediately and does not apply any unfinished transactions.</td>
</tr>
<tr>
<td>TRUE</td>
<td>DEPENDENT_TRANSACTIONS</td>
<td>When the apply component stops, some transactions that have been applied locally might have committed at the source database at a later point in time than some transactions that have not been applied locally.</td>
</tr>
<tr>
<td>FALSE</td>
<td>FULL</td>
<td>The apply component stops after applying the next uncommitted transaction in the commit order, if any such transaction is in progress.</td>
</tr>
<tr>
<td>FALSE</td>
<td>DEPENDENT_TRANSACTIONS</td>
<td>Before stopping, the apply component applies all of the transactions that have a commit time that is earlier than the applied transaction with the most recent commit time.</td>
</tr>
</tbody>
</table>
For example, assume that the `commit_serialization` apply component parameter is set to `DEPENDENT_TRANSACTIONS` and there are three transactions: transaction 1 has the earliest commit time, transaction 2 is committed after transaction 1, and transaction 3 has the latest commit time. Also assume that an apply component has applied transaction 1 and transaction 3 and is in the process of applying transaction 2 when the `STOP_APPLY` procedure is run. Given this scenario, if the `force` parameter is set to `TRUE`, then transaction 2 is not applied, and the apply component stops (transaction 2 is rolled back). If, however, the `force` parameter is set to `FALSE`, then transaction 2 is applied before the apply component stops.

A different scenario would result if the `commit_serialization` apply component parameter is set to `FULL`. For example, assume that the `commit_serialization` apply component parameter is set to `FULL` and there are three transactions: transaction A has the earliest commit time, transaction B is committed after transaction A, and transaction C has the latest commit time. In this case, the apply component has applied transaction A and is in the process of applying transactions B and C when the `STOP_APPLY` procedure is run. Given this scenario, if the `force` parameter is set to `TRUE`, then transactions B and C are not applied, and the apply component stops (transactions B and C are rolled back). If, however, the `force` parameter is set to `FALSE`, then transaction B is applied before the apply component stops, and transaction C is rolled back.

### See Also:

[SET_PARAMETER Procedure](#) for more information about the `commit_serialization` apply component parameter.

The `STOP_APPLY` Procedure and XStream Outbound Servers

This procedure functions the same way for apply processes and outbound servers.

The `STOP_APPLY` Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.
The **DBMS_AQ** package provides an interface to Oracle Streams Advanced Queuing (AQ).

This chapter contains the following topics:

- Security Model
- Constants
- Data Structures
- Operational Notes
- Summary of DBMS_AQ Subprograms

**See Also:**
- *Oracle Database Advanced Queuing User's Guide*
- *Oracle Database Advanced Queuing (AQ) Types* for information about `TYPES` to use with `DBMS_AQ`.

### 21.1 DBMS_AQ Security Model

Initially, only **SYS** and **SYSTEM** have execution privilege for the procedures in `DBMS_AQ` and `DBMS_AQADM`.

**Queue Security**

To enqueue or dequeue, users need `EXECUTE` rights on `DBMS_AQ` and either `ENQUEUE` or `DEQUEUE` privileges on target queues, or `ENQUEUE_ANY/DEQUEUE_ANY` system privileges. Users who have been granted `EXECUTE` rights to `DBMS_AQ` and `DBMS_AQADM` are able to create, manage, and use queues in their own schemas. The `MANAGE_ANY AQ` system privilege is used to create and manage queues in other schemas.

As a database user, you do not need any explicit object-level or system-level privileges to enqueue or dequeue to queues in your own schema other than the `EXECUTE` right on `DBMS_AQ`.

**See Also:**
- *Oracle Database Advanced Queuing User's Guide* for more information on queue privileges and access control.
OCI Applications and Queue Access

For an Oracle Call Interface (OCI) application to access a queue, the session user must be granted either the object privilege of the queue he intends to access or the ENQUEUE ANY QUEUE or DEQUEUE ANY QUEUE system privileges. The EXECUTE right of DBMS_AQ is not checked against the session user's rights.

Security Required for Propagation

Propagation jobs are owned by SYS, but the propagation occurs in the security context of the queue table owner. Previously propagation jobs were owned by the user scheduling propagation, and propagation occurred in the security context of the user setting up the propagation schedule. The queue table owner must be granted EXECUTE privileges on the DBMS_AQADM package. Otherwise, the Oracle Database snapshot processes do not propagate and generate trace files with the error identifier SYS.DBMS_AQADM not defined. Private database links owned by the queue table owner can be used for propagation. The username specified in the connection string must have EXECUTE access on the DBMS_AQ and DBMS_AQADM packages on the remote database.

See Also:

• DBMS_AQADM
• Oracle Database Advanced Queuing User's Guide for more information on security required for propagation

21.2 DBMS_AQ Constants

This topic describes the constants used by DBMS_AQ.

The DBMS_AQ package uses the constants shown in the following table.

When using enumerated constants such as BROWSE, LOCKED, or REMOVE, the PL/SQL constants must be specified with the scope of the packages defining it. All types associated with the operational interfaces have to be prepended with DBMS_AQ. For example: DBMS_AQ.BROWSE.

Note:

The sequence_deviation attribute has no effect in releases prior to Oracle Streams AQ 10g Release 1 (10.1) if message_grouping parameter of DBMS_AQADM subprograms is set to TRANSACTIONAL. The sequence deviation feature is deprecated in Oracle Streams AQ 10g Release 2 (10.2).
Table 21-1  Enumerated Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISIBILITY</td>
<td>IMMEDIATE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>ON_COMMIT</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DEQUEUE_MODE</td>
<td>BROWSE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>LOCKED</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>REMOVE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>REMOVE_NODATA</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NAVIGATION</td>
<td>FIRST_MESSAGE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>NEXT_MESSAGE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>STATE</td>
<td>WAITING</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>READY</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>PROCESSED</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>EXPIRED</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SEQUENCE_DEVIATION</td>
<td>BEFORE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>TOP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WAIT</td>
<td>FOREVER</td>
<td>BINARY_INTEGER</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>NO_WAIT</td>
<td>BINARY_INTEGER</td>
<td>-</td>
</tr>
<tr>
<td>DELAY</td>
<td>NO_DELAY</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EXPIRATION</td>
<td>NEVER</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NAMESPACE</td>
<td>NAMESPACE_AQ</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>NAMESPACE_ANONYMOUS</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NTFN_GROUPING_CLASS</td>
<td>NTFN_GROUPING_CLASS_NUMBER</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>_TIME</td>
<td>NUMBER</td>
<td>-</td>
</tr>
<tr>
<td>NTFN_GROUPING_TYPE</td>
<td>NTFN_GROUPING_TYPE_NUMBER</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>_SUMMARY</td>
<td>NUMBER</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>_LAST</td>
<td>NUMBER</td>
<td>-</td>
</tr>
<tr>
<td>NTFN_GROUPING_REPEAT_COUNT</td>
<td>NTFN_GROUPING_FOREV_NUMBER</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

21.3 DBMS_AQ Data Types

This topic lists and describes DBMS_AQ data types.

Table 21-2  DBMS_AQ Data Structures

<table>
<thead>
<tr>
<th>Data Structures</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_NAME</td>
<td>Names database objects</td>
</tr>
<tr>
<td>TYPE_NAME</td>
<td>Defines queue types</td>
</tr>
</tbody>
</table>
### Table 21-2  (Cont.) DBMS_AQ Data Structures

<table>
<thead>
<tr>
<th>Data Structures</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Database Advanced Queuing PL/SQL</td>
<td>Specifies the user-defined PL/SQL procedure, defined in the database to be invoked on message notification</td>
</tr>
<tr>
<td>Callback</td>
<td></td>
</tr>
</tbody>
</table>

**OBJECT_NAME**

The `object_name` data structure names database objects. It applies to queues, queue tables, agent names, and object types.

**Syntax**

```sql
object_name := VARCHAR2;
object_name := [schema_name.]name;
```

**Usage Notes**

Names for objects are specified by an optional schema name and a name. If the schema name is not specified, the current schema is assumed. The name must follow object name guidelines in *Oracle Database SQL Language Reference* with regard to reserved characters. Schema names, agent names, and object type names can be up to 128 bytes long. Queue names and queue table names can be up to 122 bytes long. Maximum length of agent names and subscriber names can be 128 characters with datablock size greater than 2k. For 2k datablock size, maximum length for subscriber is 128 bytes.

**TYPE_NAME**

The `type_name` data structure defines queue types.

**Syntax**

```sql
type_name := VARCHAR2;
type_name := object_type | "RAW";
```

**Attributes**

**Table 21-3  Type Name Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object_type</code></td>
<td>Maximum number of attributes in the object type is limited to 900.</td>
</tr>
</tbody>
</table>
Table 21-3  (Cont.) Type Name Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;RAW&quot;</td>
<td>To store payload of type RAW, Oracle Database Advanced Queuing creates a queue table with a LOB column as the payload repository. The theoretical maximum size of the message payload is the maximum amount of data that can be stored in a LOB column. However, the maximum size of the payload is determined by which programmatic environment you use to access Oracle Database Advanced Queuing. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G. Because the PL/SQL enqueue and dequeue interfaces accept RAW buffers as the payload parameters you will be limited to 32K bytes. In OCI, the maximum size of your RAW data will be limited to the maximum amount of contiguous memory (as an OCIRaw is simply an array of bytes) that the OCI Object Cache can allocate. Typically, this will be at least 32K bytes and much larger in many cases. Because LOB columns are used for storing RAW payload, the Oracle Database Advanced Queuing administrator can choose the LOB tablespace and configure the LOB storage by constructing a LOB storage string in the storage_clause parameter during queue table creation time.</td>
</tr>
</tbody>
</table>

Oracle Database Advanced Queuing PL/SQL Callback

The plsqlcallback data structure specifies the user-defined PL/SQL procedure, defined in the database to be invoked on message notification.

Syntax

If a notification message is expected for a RAW payload enqueue, then the PL/SQL callback must have the following signature:

```sql
procedure plsqlcallback(
    context  IN  RAW,
    reginfo  IN  SYS.AQ$_REG_INFO,
    descr    IN  SYS.AQ$_DESCRIPTOR,
    payload  IN  RAW,
    payloadl IN  NUMBER);
```

Attributes

Table 21-4  Oracle Database Advanced Queuing PL/SQL Callback Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Specifies the context for the callback function that was passed by dbms_aq.register. See AQ$_REG_INFO Type.</td>
</tr>
<tr>
<td>reginfo</td>
<td>See AQ$_REG_INFO Type.</td>
</tr>
<tr>
<td>descr</td>
<td>See AQ$_DESCRIPTOR Type</td>
</tr>
<tr>
<td>payload</td>
<td>If a notification message is expected for a raw payload enqueue then this contains the raw payload that was enqueued into a non persistent queue. In case of a persistent queue with raw payload this parameter will be null.</td>
</tr>
<tr>
<td>payloadl</td>
<td>Specifies the length of payload. If payload is null, payloadl = 0.</td>
</tr>
</tbody>
</table>
If the notification message is expected for an ADT payload enqueue, the PL/SQL callback must have the following signature:

```plsql
procedure plsqlcallback(
    context  IN RAW,
    reginfo  IN SYS.AQ$_REG_INFO,
    descr    IN SYS.AQ$_DESCRIPTOR,
    payload  IN VARCHAR2,
    payloadl IN NUMBER);
```

### 21.4 DBMS_AQ Operational Notes

This topic lists various DBMS_AQ operational notes.

**DBMS_AQ and DBMS_AQADM Java Classes**

Java interfaces are available for DBMS_AQ and DBMS_AQADM. The Java interfaces are provided in the `$ORACLE_HOME/rdbms/jlib/aqapi.jar`. Users are required to have `EXECUTE` privileges on the `DBMS_AQIN` package to use these interfaces.

### 21.5 Summary of DBMS_AQ Subprograms

The `DBMS_AQ` package uses subprograms described in this table.

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIND_AGENT Procedure</strong></td>
<td>Creates an entry for an Oracle Database Advanced Queuing agent in the LDAP directory</td>
</tr>
<tr>
<td>DEQUEUE Procedure</td>
<td>Dequeues a message from the specified queue</td>
</tr>
<tr>
<td>DEQUEUE_ARRAY Function</td>
<td>Dequeues an array of messages from the specified queue</td>
</tr>
<tr>
<td>ENQUEUE Procedure</td>
<td>Adds a message to the specified queue</td>
</tr>
<tr>
<td>ENQUEUE_ARRAY Function</td>
<td>Adds an array of messages to the specified queue</td>
</tr>
<tr>
<td>LISTEN Procedures</td>
<td>Listen to one or more queues on behalf of a list of agents</td>
</tr>
<tr>
<td>POST Procedure</td>
<td>Posts to an anonymous subscription which allows all clients who are registered for the subscription to get notifications</td>
</tr>
<tr>
<td>REGISTER Procedure</td>
<td>Registers for message notifications</td>
</tr>
<tr>
<td>UNBIND_AGENT Procedure</td>
<td>Removes an entry for an Oracle Database Advanced Queuing agent from the LDAP directory</td>
</tr>
<tr>
<td>UNREGISTER Procedure</td>
<td>Unregisters a subscription which turns off notification</td>
</tr>
</tbody>
</table>

**Note:**

DBMS_AQ does not have a purity level defined; therefore, you cannot call any procedure in this package from other procedures that have RNDS, WNDS, RNPS or WNPS constraints defined.
21.5.1 BIND_AGENT Procedure

This procedure creates an entry for an Oracle Database Advanced Queuing agent in the LDAP server.

Syntax

```
DBMS_AQ.BIND_AGENT(
    agent        IN SYS.AQ$_AGENT,
    certificate  IN VARCHAR2 default NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent</td>
<td>Agent that is to be registered in LDAP server.</td>
</tr>
<tr>
<td>certificate</td>
<td>Location (LDAP distinguished name) of the &quot;organizationalperson&quot; entry in LDAP whose digital certificate (attribute usercertificate) is to be used for this agent. Example: &quot;cn=OE, cn=ACME, cn=com&quot; is a distinguished name for a OrganizationalPerson OE whose certificate will be used with the specified agent.</td>
</tr>
</tbody>
</table>

Usage Notes

In the LDAP server, digital certificates are stored as an attribute (usercertificate) of the OrganizationalPerson entity. The distinguished name for this OrganizationalPerson must be specified when binding the agent.

21.5.2 DEQUEUE Procedure

This procedure dequeues a message from the specified queue.

Syntax

```
DBMS_AQ.DEQUEUE (
    queue_name          IN      VARCHAR2,
    dequeue_options     IN      dequeue_options_t,
    message_properties  OUT     message_properties_t,
    payload             OUT     "<ADT_1>"
    msgid               OUT     RAW);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Specifies the name of the queue.</td>
</tr>
<tr>
<td>dequeue_options</td>
<td>See DEQUEUE_OPTIONS_T Type.</td>
</tr>
<tr>
<td>message_properties</td>
<td>See MESSAGE_PROPERTIES_T Type.</td>
</tr>
</tbody>
</table>
payload
Not interpreted by Oracle Database Advanced Queuing. The payload must be specified according to the specification in the associated queue table. For the definition of type_name refer to TYPE_NAME in DBMS_AQ Data Types.

msgid
System generated identification of the message.

Usage Notes
The search criteria for messages to be dequeued is determined by the following parameters in dequeue_options:

• consumer_name
• msgid
  Msgid uniquely identifies the message to be dequeued. Only messages in the READY state are dequeued unless msgid is specified.
• correlation
  Correlation identifiers are application-defined identifiers that are not interpreted by Oracle Database Advanced Queuing.
• deq_condition
  Dequeue condition is an expression based on the message properties, the message data properties and PL/SQL functions. A deq_condition is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Message properties include priority, corrid and other columns in the queue table.
  To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload.
  Example: tab.user_data.orderstatus='EXPRESS'

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the msgid and correlation ID in dequeue_options.

The database-consistent read mechanism is applicable for queue operations. For example, a BROWSE call may not see a message that is enqueued after the beginning of the browsing transaction.

The default NAVIGATION parameter during dequeue is NEXT_MESSAGE. This means that subsequent dequeues will retrieve the messages from the queue based on the snapshot obtained in the first dequeue. In particular, a message that is enqueued after the first dequeue command will be processed only after processing all the remaining messages in the queue. This is usually sufficient when all the messages have already been enqueued into the queue, or when the queue does not have a priority-based ordering. However, applications must use the FIRST_MESSAGE navigation option when
the first message in the queue needs to be processed by every dequeue command. This usually becomes necessary when a higher priority message arrives in the queue while messages already-enqueued are being processed.

**Note:**

It may be more efficient to use the `FIRST_MESSAGE` navigation option when messages are concurrently enqueued. If the `FIRST_MESSAGE` option is not specified, Oracle Database Advanced Queuing continually generates the snapshot as of the first dequeue command, leading to poor performance. If the `FIRST_MESSAGE` option is specified, then Oracle Database Advanced Queuing uses a new snapshot for every dequeue command.

Messages enqueued in the same transaction into a queue that has been enabled for message grouping will form a group. If only one message is enqueued in the transaction, then this will effectively form a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in `LOCKED` or `REMOVE` mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group will lock the entire group. This is useful when all the messages in a group need to be processed as an atomic unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use the `NEXT_TRANSACTION` to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue will time out after the specified `WAIT` period.

Using Secure Queues

For secure queues, you must specify `consumer_name` in the `dequeue_options` parameter. See `DEQUEUE_OPTIONS_T Type` for more information about `consumer_name`.

When you use secure queues, the following are required:

- You must have created a valid Oracle Database Advanced Queuing agent using `DBMS_AQADM.CREATE_AQ_AGENT`. See `CREATE_AQ_AGENT Procedure`.
- You must map the Oracle Database Advanced Queuing agent to a database user with dequeue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this. See `ENABLE_DB_ACCESS Procedure`.

**See Also:**

*Oracle Streams Concepts and Administration* for information about secure queues.
21.5.3 DEQUEUE_ARRAY Function

This function dequeues an array of messages and returns them in the form of an array of payloads, an array of message properties and an array of message IDs. This function returns the number of messages successfully dequeued.

Syntax

```
DBMS_AQ.DEQUEUE_ARRAY (  
qu_name IN VARCHAR2,  
d_equeu_options IN dequeue_options_t,  
ar_size IN pls_integer,  
message_properties_array OUT message_properties_array_t,  
payload_array OUT "<COLLECTION_1>",  
msgid_array OUT msgid_array_t,  
error_array OUT error_array_t)
RETURN pls_integer;
```

Parameters

Table 21-8  DEQUEUE_ARRAY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The queue name from which messages are dequeued (same as single-row dequeue).</td>
</tr>
<tr>
<td>dequeue_options</td>
<td>The set of options which will be applied to all messages in the array (same as single-row dequeue).</td>
</tr>
<tr>
<td>array_size</td>
<td>The number of elements to dequeue. For buffered messages, array_size should be 1.</td>
</tr>
<tr>
<td>message_properties_array</td>
<td>A record containing an array corresponding to each message property. Each payload element has a corresponding set of message properties. See MESSAGE_PROPERTIES_ARRAY_T Type.</td>
</tr>
<tr>
<td>payload_array</td>
<td>An array of dequeued payload data. &quot;&lt;COLLECTION_1&gt;&quot; can be an associative array, varray or nested table in its PL/SQL representation. Users can dequeue RAW and ADT payloads.</td>
</tr>
<tr>
<td>msgid_array</td>
<td>An array of message IDs of the dequeued messages. See MSGID_ARRAY_T Type.</td>
</tr>
<tr>
<td>error_array</td>
<td>Currently not implemented</td>
</tr>
</tbody>
</table>

Usage Notes

A nonzero wait time, as specified in dequeue_options, is recognized only when there are no messages in the queue. If the queue contains messages that are eligible for dequeue, then the DEQUEUE_ARRAY function will dequeue up to array_size messages and return immediately.

Dequeue by message_id is not supported. See DEQUEUE Procedure for more information on the navigation parameter. Existing NAVIGATION modes are supported. In addition, two new NAVIGATION modes are supported for queues enabled for message grouping:

- FIRST_MESSAGE_MULTI_GROUP
For transaction grouped queues and 


**NEXT_MESSAGE_MULTI_GROUP**

See Also:

**ENQUEUE_OPTIONS_T Type**

For transaction grouped queues and **ONE_GROUP** navigation, messages are dequeued from a single transaction group only, subject to the **array_size** limit. In **MULTI_GROUP** navigation, messages are dequeued across multiple transaction groups, still subject to the **array_size** limit. **ORA-25235** is returned to indicate the end of a transaction group.

**DEQUEUE_ARRAY** is not supported for buffered messages, but you can still use this procedure on individual buffered messages by setting **array_size** to one message.

### 21.5.4 ENQUEUE Procedure

This procedure adds a message to the specified queue.

**Syntax**

```sql
DBMS_AQ.ENQUEUE (queue_name          IN      VARCHAR2,
                   enqueue_options     IN      enqueue_options_t,
                   message_properties  IN      message_properties_t,
                   payload             IN       "<ADT_1>",
                   msgid               OUT     RAW);
```

**Parameters**

**Table 21-9** ENQUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Specifies the name of the queue to which this message should be enqueued. The queue cannot be an exception queue.</td>
</tr>
<tr>
<td>enqueue_options</td>
<td>See <strong>ENQUEUE_OPTIONS_T Type</strong>.</td>
</tr>
<tr>
<td>message_properties</td>
<td>See <strong>MESSAGE_PROPERTIES_T Type</strong>.</td>
</tr>
<tr>
<td>payload</td>
<td>Not interpreted by Oracle Database Advanced Queuing. The payload must be specified according to the specification in the associated queue table. <strong>NULL</strong> is an acceptable parameter. For the definition of <strong>type_name</strong> refer to <strong>TYPE_NAME</strong> in <strong>DBMS_AQ Data Types</strong>.</td>
</tr>
<tr>
<td>msgid</td>
<td>System generated identification of the message. This is a globally unique identifier that can be used to identify the message at dequeue time.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The **sequence_deviation** parameter in **enqueue_options** can be used to change the order of processing between two messages. The identity of the other message, if any, is specified by the **enqueue_options** parameter **relative_msgid**. The relationship is identified by the **sequence_deviation** parameter.
Specifying `sequence_deviation` for a message introduces some restrictions for the delay and priority values that can be specified for this message. The delay of this message must be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message must be greater than or equal to the priority of the message before which this message is to be enqueued.

**Note:**

The `sequence_deviation` attribute has no effect in releases prior to Oracle Streams AQ 10g Release 1 (10.1) if `message_grouping` parameter of `DBMS_AQADM` subprograms is set to `TRANSACTIONAL`. The sequence deviation feature is deprecated in Oracle Streams AQ 10g Release 2 (10.2).

If a message is enqueued to a multiconsumer queue with no recipient, and if the queue has no subscribers (or rule-based subscribers that match this message), then Oracle error `ORA_24033` is raised. This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

**Using Secure Queues**

For secure queues, you must specify the `sender_id` in the `messages_properties` parameter. See `MESSAGE_PROPERTIES_T Type` for more information about `sender_id`.

When you use secure queues, the following are required:

- You must have created a valid Oracle Database Advanced Queuing agent using `DBMS_AQADM.CREATE_AQ_AGENT`. See `CREATE_AQ_AGENT Procedure`.
- You must map `sender_id` to a database user with enqueue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this. See `ENABLE_DB_ACCESS Procedure`.

**See Also:**

`Oracle Streams Concepts and Administration` for information about secure queues

### 21.5.5 ENQUEUE_ARRAY Function

This function enqueues an array of payloads using a corresponding array of message properties. The output will be an array of message IDs of the enqueued messages.

**Syntax**

```sql
DBMS_AQ.ENQUEUE_ARRAY (queue_name                IN   VARCHAR2,
 enqueue_options           IN   enqueue_options_t,
 array_size                IN   pls_integer,
 message_properties_array  IN   message_properties_array_t,
 payload_array             IN   "<COLLECTION_1>",
 msgid_array               OUT  msgid_array_t,
)```

21-12
error_array               OUT  error_array_t)
RETURN pls_integer;

Parameters

Table 21-10    ENQUEUE_ARRAY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The queue name in which messages are enqueued (same as single-row enqueue).</td>
</tr>
<tr>
<td>enqueue_options</td>
<td>See ENQUEUE_OPTIONS_T Type.</td>
</tr>
<tr>
<td>array_size</td>
<td>The number of elements to enqueue. For buffered messages, array_size should be 1.</td>
</tr>
<tr>
<td>message_properties_array</td>
<td>A record containing an array corresponding to each message property. For each property, the user must allocate array_size elements. See MESSAGE_PROPERTIES_ARRAY_T Type.</td>
</tr>
<tr>
<td>payload_array</td>
<td>An array of payload data. &quot;&lt;COLLECTION_1&gt;&quot; can be an associative array, VARRAY, or nested table in its PL/SQL representation. Users can enqueue RAW and ADT payloads.</td>
</tr>
<tr>
<td>msgid_array</td>
<td>An array of message IDs for the enqueued messages. If an error occurs for a particular message, then its corresponding message ID is null. See MSGID_ARRAY_T Type.</td>
</tr>
<tr>
<td>error_array</td>
<td>Currently not implemented</td>
</tr>
</tbody>
</table>

Usage Notes

ENQUEUE_ARRAY is not supported for buffered messages, but you can still use this procedure on individual buffered messages by setting array_size to one message.

21.5.6 LISTEN Procedures

This procedure listens on one or more queues on behalf of a list of agents. The address field of the agent indicates the queue the agent wants to monitor. Only local queues are supported as addresses. Protocol is reserved for future use.

Syntax

DBMS_AQ.LISTEN ( 
    agent_list            IN    AQ$_AGENT_LIST_T,
    wait                  IN    BINARY_INTEGER DEFAULT DBMS_AQ.FOREVER,
    agent                 OUT   SYS.AQ$_AGENT);

DBMS_AQ.LISTEN ( 
    agent_list            IN    AQ$_AGENT_LIST_T,
    wait                  IN    BINARY_INTEGER DEFAULT FOREVER,
    listen_delivery_mode   IN    PLS_INTEGER DEFAULT DBMS_AQ.PERSISTENT,
    agent                 OUT   SYS.AQ$_AGENT,
    message_delivery_mode  OUT   PLS_INTEGER);

TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_list</td>
<td>List of agents to listen for</td>
</tr>
<tr>
<td>wait</td>
<td>Time out for the listen call in seconds. By default, the call will block forever.</td>
</tr>
<tr>
<td>listen_delivery_mode</td>
<td>The caller specifies whether it is interested in persistent, buffered messages or both types of messages, specifying a delivery mode of DBMS_AQ.PERSISTENT or DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT_OR_BUFFERED</td>
</tr>
<tr>
<td>agent</td>
<td>Agent with a message available for consumption</td>
</tr>
<tr>
<td>message_delivery_mode</td>
<td>Returns the message type along with the queue and consumer for which there is a message</td>
</tr>
</tbody>
</table>

Usage Notes

If agent-address is a multiconsumer queue, then agent-name is mandatory. For single-consumer queues, agent-name must not be specified.

This procedure takes a list of agents as an argument. You specify the queue to be monitored in the address field of each agent listed. You also must specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. Only local queues are supported as addresses. Protocol is reserved for future use.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, only the first agent listed is returned. If there are no messages found when the wait time expires, an error is raised.

A successful return from the LISTEN call is only an indication that there is a message for one of the listed agents in one the specified queues. The interested agent must still dequeue the relevant message.

**Note:**

You cannot call LISTEN on nonpersistent queues.
21.5.7 POST Procedure

This procedure posts to a list of anonymous subscriptions that allows all clients who are registered for the subscriptions to get notifications.

Syntax

```
DBMS_AQ.POST (
    post_list       IN  SYS.AQ$_POST_INFO_LIST,
    post_count      IN  NUMBER);
```

Parameters

Table 21-12  POST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>post_list</td>
<td>Specifies the list of anonymous subscriptions to which you want to post. It is a list of AQ$_POST_INFO_LIST Type.</td>
</tr>
<tr>
<td>post_count</td>
<td>Specifies the number of entries in the post_list.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is used to post to anonymous subscriptions which allows all clients who are registered for the subscriptions to get notifications. Several subscriptions can be posted to at one time.

21.5.8 REGISTER Procedure

This procedure registers an e-mail address, user-defined PL/SQL procedure, or HTTP URL for message notification.

Syntax

```
DBMS_AQ.REGISTER (
    reg_list     IN  SYS.AQ$_REG_INFO_LIST,
    count        IN  NUMBER);
```

Parameters

Table 21-13  REGISTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_list</td>
<td>Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ$_REG_INFO Type.</td>
</tr>
<tr>
<td>count</td>
<td>Specifies the number of entries in the reg_list.</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure is used to register for notifications. You can specify an e-mail address to which message notifications are sent, register a procedure to be invoked on a notification, or register an HTTP URL to which the notification is posted. Interest in several subscriptions can be registered at one time.
The procedure can also be used to register for grouping notifications using five grouping attributes:

- **Class** – grouping criterion (currently only **TIME** criterion is supported)
- **Value** – the value of the grouping criterion (currently only time in seconds for criterion **TIME**)
- **Type** – summary or last, also contains count of notifications received in group (for AQ namespace only, not for DBCHANGE namespace)
- **Repeat count** – how many times to perform grouping (Default is **FOREVER**)
- **Start time** – when to start grouping (Default is current time)

If you register for e-mail notifications, you should set the host name and port name for the SMTP server that will be used by the database to send e-mail notifications. If required, you should set the send-from e-mail address, which is set by the database as the `sent from` field. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, you may want to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

**See Also:**

**DBMS_AQELM** for more information on e-mail and HTTP notifications

### 21.5.9 UNBIND_AGENT Procedure

This procedure removes the entry for an Oracle Database Advanced Queuing agent from the LDAP server.

**Syntax**

```sql
DBMS_AQ.UNBIND_AGENT(
    agent    IN SYS.AQ$_AGENT);
```

**Parameters**

**Table 21-14   BIND_AGENT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent</td>
<td>Agent that is to be removed from the LDAP server</td>
</tr>
</tbody>
</table>

### 21.5.10 UNREGISTER Procedure

This procedure unregisters a subscription which turns off notifications.

**Syntax**

```sql
DBMS_AQ.UNREGISTER (
    reg_list     IN SYS.AQ$_REG_INFO_LIST,
    reg_count    IN NUMBER);
```
Parameters

Table 21-15  UNREGISTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_list</td>
<td>Specifies the list of subscriptions to which you want to register for message notifications. It is a list of AQ$_REG_INFO Type.</td>
</tr>
<tr>
<td>reg_count</td>
<td>Specifies the number of entries in the reg_list.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is used to unregister a subscription which turns off notifications. Several subscriptions can be unregistered from at one time.
The `DBMS_AQADM` package provides procedures to manage Oracle Database Advanced Queuing (AQ) configuration and administration information.

**See Also:**

- *Oracle Database Advanced Queuing User’s Guide*
- *Oracle Database Advanced Queuing (AQ) Types* for information about the types to use with `DBMS_AQADM`

This chapter contains the following topics:

- Security Model
- Constants
- Subprogram Groups
- Summary of `DBMS_AQADM` Subprograms

### 22.1 DBMS_AQADM Security Model

Initially, only `SYS` and `SYSTEM` have execution privilege for the procedures in `DBMS_AQADM` and `DBMS_AQ`. Users who have been granted `EXECUTE` rights to `DBMS_AQ` and `DBMS_AQADM` are able to create, manage, and use queues in their own schemas. The `MANAGE_ANY` AQ system privilege is used to create and manage queues in other schemas and can be granted and revoked through `DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE` and `DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE` respectively. Starting from Oracle Database 12c Release 2, `MANAGE_ANY` privilege will not allow access to `SYS` owned queues by users other than `SYS`.

**User Roles**

The database administrator has the option of granting the system privileges `ENQUEUE_ANY` and `DEQUEUE_ANY`, exercising `DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE` and `DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE` directly to a database user, if you want the user to have this level of control.

The application developer gives rights to a queue by granting and revoking privileges at the object level by exercising `DBMS_AQADM.GRANT_QUEUE_PRIVILEGE` and `DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE`. Starting from Oracle Database 12c Release 2, `ENQUEUE_ANY` and `DEQUEUE_ANY` privileges will not allow access to `SYS` owned queues by users other than `SYS`. 
See Also:

- DBMS_AQ.
- Oracle Database Advanced Queuing User's Guide for more information on queue privileges and access control.

Security Required for Propagation

Propagation jobs are owned by SYS, but the propagation occurs in the security context of the queue table owner. Previously propagation jobs were owned by the user scheduling propagation, and propagation occurred in the security context of the user setting up the propagation schedule. The queue table owner must be granted EXECUTE privileges on the DBMS_AQADM package. Otherwise, the Oracle Database snapshot processes do not propagate and generate trace files with the error identifier SYS.DBMS_AQADM not defined. Private database links owned by the queue table owner can be used for propagation. The username specified in the connection string must have EXECUTE access on the DBMS_AQ and DBMS_AQADM packages on the remote database.

See Also:


Queue Table Migration

The MIGRATE_QUEUE_TABLE procedure requires that the EXECUTE privilege on DBMS_AQADM be granted to the queue table owner, who is probably an ordinary queue user. If you do not want ordinary queue users to be able to create and drop queues and queue tables, add and delete subscribers, and so forth, then you must revoke the EXECUTE privilege as soon as the migration is done.

See Also:

- "MIGRATE_QUEUE_TABLE Procedure."
- Oracle Database Advanced Queuing User's Guide for more information on granting Oracle Database Advanced Queuing system privileges.

22.2 DBMS_AQADM Constants

When using enumerated constants, such as INFINITE, TRANSACTIONAL, or NORMAL_QUEUE, the symbol must be specified with the scope of the packages defining it. All types associated with the administrative interfaces must be prepended with DBMS_AQADM. For example: DBMS_AQADM.NORMAL_QUEUE.
Table 22-1  Enumerated Types in the Administrative Interface

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>retention</td>
<td>0, 1, 2...INFINITE</td>
</tr>
<tr>
<td>message_grouping</td>
<td>TRANSACTIONAL, NONE</td>
</tr>
<tr>
<td>queue_type</td>
<td>NORMAL_QUEUE, EXCEPTION_QUEUE, NON_PERSISTENT_QUEUE</td>
</tr>
</tbody>
</table>

See Also:
For more information on the Java classes and data structures used in both DBMS_AQ and DBMS_AQADM, see the DBMS_AQ package.

22.3 DBMS_AQADM Subprogram Groups

This section lists and describes the DBMS_AQADM subprogram groups.

This DBMS_AQADM package is made up of the following subprogram groups:

- Queue Table Subprograms
- Privilege Subprograms
- Queue Subprograms
- Subscriber Subprograms
- Propagation Subprograms
- Miscellaneous Subprograms
- Oracle Database Advanced Queuing Agent Subprograms
- Alias Subprograms

22.3.1 DBMS_AQADM Queue Table Subprograms

This section lists and describes the DBMS_AQADM Queue Table subprograms.

Table 22-2  Queue Table Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_QUEUE_TABLE Procedure</td>
<td>Alters the existing properties of a queue table</td>
</tr>
<tr>
<td>CREATE_QUEUE_TABLE Procedure</td>
<td>Creates a queue table for messages of a predefined type</td>
</tr>
<tr>
<td>DROP_QUEUE_TABLE Procedure</td>
<td>Drops an existing queue table</td>
</tr>
<tr>
<td>ENABLE_JMS_TYPES Procedure</td>
<td>A precondition for the enqueue of JMS types and XML types</td>
</tr>
</tbody>
</table>
22.3.2 DBMS_AQADM Privilege Subprograms

This section lists and describes the DBMS_AQADM Privilege subprograms.

Table 22-3 Privilege Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT_QUEUE_PRIVILEGE</td>
<td>Grants privileges on a queue to users and roles</td>
</tr>
<tr>
<td>GRANT_SYSTEM_PRIVILEGE</td>
<td>Grants Oracle Database Advanced Queuing system privileges to users and roles</td>
</tr>
<tr>
<td>REVOKE_QUEUE_PRIVILEGE</td>
<td>Revokes privileges on a queue from users and roles</td>
</tr>
<tr>
<td>REVOKE_SYSTEM_PRIVILEGE</td>
<td>Revokes Oracle Database Advanced Queuing system privileges from users and roles</td>
</tr>
</tbody>
</table>

22.3.3 DBMS_AQADM Queue Subprograms

This section lists and describes the DBMS_AQADM Queue subprograms.

Table 22-4 Queue Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_QUEUE Procedure</td>
<td>Alters existing properties of a queue</td>
</tr>
<tr>
<td>CREATE_NP_QUEUE Procedure</td>
<td>Creates a nonpersistent RAW queue</td>
</tr>
<tr>
<td>CREATE_QUEUE Procedure</td>
<td>Creates a queue in the specified queue table</td>
</tr>
<tr>
<td>CREATE_SHARDED_QUEUE</td>
<td>Creates a queue and its queue table for a sharded queue all together.</td>
</tr>
<tr>
<td>DROP_SHARDED_QUEUE</td>
<td>Drops an existing sharded queue from the database queuing system</td>
</tr>
<tr>
<td>ALTER_SHARDED_QUEUE</td>
<td>Alters an sharded queue in the database queuing system</td>
</tr>
<tr>
<td>CREATE_EXCEPTION_QUEUE E Procedure</td>
<td>Creates an exception queue for a sharded queue</td>
</tr>
<tr>
<td>DROP_QUEUE Procedure</td>
<td>Drops an existing queue</td>
</tr>
</tbody>
</table>
Table 22-4  (Cont.) Queue Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE_SUBSCRIBERS Function</td>
<td>Returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type DBMS_AQADM.AQ$Subscriber_list_t</td>
</tr>
<tr>
<td>START_QUEUE Procedure</td>
<td>Enables the specified queue for enqueuing or dequeuing</td>
</tr>
<tr>
<td>STOP_QUEUE Procedure</td>
<td>Disables enqueuing or dequeuing on the specified queue</td>
</tr>
</tbody>
</table>

22.3.4 DBMS_AQADM Subscriber Subprograms

This section lists and describes the DBMS_AQADM Subscriber subprograms.

Table 22-5  Subscriber Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_SUBSCRIBER Procedure</td>
<td>Adds a default subscriber to a queue</td>
</tr>
<tr>
<td>ALTER_SUBSCRIBER Procedure</td>
<td>Alters existing properties of a subscriber to a specified queue</td>
</tr>
<tr>
<td>REMOVE_SUBSCRIBER Procedure</td>
<td>Removes a default subscriber from a queue</td>
</tr>
</tbody>
</table>

22.3.5 DBMS_AQADM Propagation Subprograms

This section lists and describes the DBMS_AQADM propagation subprograms.

Table 22-6  Propagation Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_PROPAGATION_SCHEDU LE Procedure</td>
<td>Alters parameters for a propagation schedule</td>
</tr>
<tr>
<td>DISABLE_PROPAGATION_SCHEDU LE Procedure</td>
<td>Disables a propagation schedule</td>
</tr>
<tr>
<td>ENABLE_PROPAGATION_SCHEDU LE Procedure</td>
<td>Enables a previously disabled propagation schedule</td>
</tr>
<tr>
<td>SCHEDULE_PROPAGATION Procedure</td>
<td>Schedules propagation of messages from a queue to a destination identified by a specific database link</td>
</tr>
<tr>
<td>UNSCHEDULE_PROPAGATION Procedure</td>
<td>Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link</td>
</tr>
<tr>
<td>VERIFY_QUEUE_TYPES Procedure</td>
<td>Verifies that the source and destination queues have identical types</td>
</tr>
</tbody>
</table>

22.3.6 DBMS_AQADM Miscellaneous Subprograms

This section lists and describes the DBMS_AQADM miscellaneous subprograms.
Table 22-7  Miscellaneous Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_WATERMARK Procedure</td>
<td>Retrieves the value of watermark set by the SET_WATERMARK Procedure</td>
</tr>
<tr>
<td>GET_MAX_STREAMS_POOL Procedure</td>
<td>Retrieves the value of Oracle Database Advanced Queuing maximum streams pool memory limit</td>
</tr>
<tr>
<td>GET_MIN_STREAMS_POOL Procedure</td>
<td>Retrieves the value of Oracle Database Advanced Queuing minimum streams pool memory limit</td>
</tr>
<tr>
<td>SET_WATERMARK Procedure</td>
<td>Used for Oracle Database Advanced Queuing notification to specify and limit memory use</td>
</tr>
<tr>
<td>SET_MAX_STREAMS_POOL Procedure</td>
<td>Used for Oracle Database Advanced Queuing to specify and limit maximum streams pool memory use</td>
</tr>
<tr>
<td>SET_MIN_STREAMS_POOL Procedure</td>
<td>Used for Oracle Database Advanced Queuing to specify and limit minimum streams pool memory use</td>
</tr>
<tr>
<td>SET_QUEUE_PARAMETER Procedure</td>
<td>Used to set different parameters for sharded queues at queue or database level.</td>
</tr>
<tr>
<td>GET_QUEUE_PARAMETER Procedure</td>
<td>Used to get different parameters for sharded queues at queue or database level.</td>
</tr>
<tr>
<td>UNSET_QUEUE_PARAMETER Procedure</td>
<td>Used to unset different parameters for sharded queues at queue or database level.</td>
</tr>
</tbody>
</table>

22.3.7 DBMS_AQADM Agent Subprograms

This section lists and describes the DBMS_AQADM agent subprograms.

Table 22-8  Oracle Streams AQ Agent Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_AQ_AGENT Procedure</td>
<td>Alters an agent registered for Oracle Database Advanced Queuing Internet access, and an Oracle Database Advanced Queuing agent that accesses secure queues</td>
</tr>
<tr>
<td>CREATE_AQ_AGENT Procedure</td>
<td>Registers an agent for Oracle Database Advanced Queuing Internet access using HTTP/SMTP protocols, and creates an Oracle Database Advanced Queuing agent to access secure queues</td>
</tr>
<tr>
<td>DISABLE_DB_ACCESS Procedure</td>
<td>Revokes the privileges of a specific database user from an Oracle Database Advanced Queuing Internet agent</td>
</tr>
<tr>
<td>DROP_AQ_AGENT Procedure</td>
<td>Drops an agent that was previously registered for Oracle Database Advanced Queuing Internet access</td>
</tr>
<tr>
<td>ENABLE_DB_ACCESS Procedure</td>
<td>Grants an Oracle Database Advanced Queuing Internet agent the privileges of a specific database user</td>
</tr>
</tbody>
</table>

22.3.8 DBMS_AQADM Alias Subprograms

This section lists and describes the DBMS_AQADM alias subprograms.
Table 22-9  Alias Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_ALIAS_TO_LDAP Procedure</td>
<td>Creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP</td>
</tr>
<tr>
<td>DEL_ALIAS_FROM_LDAP Procedure</td>
<td>Drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP</td>
</tr>
</tbody>
</table>

22.4 Summary of DBMS_AQADM Subprograms

This section lists and describes the DBMS_AQADM package subprograms.

Table 22-10  DBMS_AQADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_ALIAS_TO_LDAP Procedure</td>
<td>Creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP</td>
</tr>
<tr>
<td>ADD_SUBSCRIBER Procedure</td>
<td>Adds a default subscriber to a queue</td>
</tr>
<tr>
<td>ALTER_AQ_AGENT Procedure</td>
<td>Alters an agent registered for Oracle Database Advanced Queuing Internet access, and an Oracle Database Advanced Queuing agent that accesses secure queues</td>
</tr>
<tr>
<td>ALTER_PROPAGATION_SCHEDULE Procedure</td>
<td>Alters parameters for a propagation schedule</td>
</tr>
<tr>
<td>ALTER_QUEUE Procedure</td>
<td>Alters existing properties of a queue</td>
</tr>
<tr>
<td>ALTER_QUEUE_TABLE Procedure</td>
<td>Alters the existing properties of a queue table</td>
</tr>
<tr>
<td>ALTER_SHARDED_QUEUE Procedure</td>
<td>Provides user the ability to alter the cache_hint and comment for the sharded queue</td>
</tr>
<tr>
<td>ALTER_SUBSCRIBER Procedure</td>
<td>Alters existing properties of a subscriber to a specified queue</td>
</tr>
<tr>
<td>CREATE_AQ_AGENT Procedure</td>
<td>Registers an agent for Oracle Database Advanced Queuing Internet access using HTTP/SMTP protocols, and creates an Oracle Database Advanced Queuing agent to access secure queues</td>
</tr>
<tr>
<td>CREATE_NP_QUEUE Procedure</td>
<td>Creates a nonpersistent RAW queue</td>
</tr>
<tr>
<td>CREATE_QUEUE Procedure</td>
<td>Creates a queue in the specified queue table</td>
</tr>
<tr>
<td>CREATE_SHARDED_QUEUE Procedure</td>
<td>Creates a queue and its queue table for a sharded queue all together.</td>
</tr>
<tr>
<td>CREATE_EXCEPTION_QUEUE Procedure</td>
<td>Creates an exception queue.</td>
</tr>
<tr>
<td>CREATE_QUEUE_TABLE Procedure</td>
<td>Creates a queue table for messages of a predefined type</td>
</tr>
<tr>
<td>DEL_ALIAS_FROM_LDAP Procedure</td>
<td>Drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP</td>
</tr>
<tr>
<td>DISABLE_DB_ACCESS Procedure</td>
<td>Revokes the privileges of a specific database user from an Oracle Database Advanced Queuing Internet agent</td>
</tr>
<tr>
<td>Subprograms</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DISABLE_PROPAGATION_SCHEDULE Procedure</td>
<td>Disables a propagation schedule</td>
</tr>
<tr>
<td>DROP_AQ_AGENT Procedure</td>
<td>Drops an agent that was previously registered for Oracle Database Advanced Queuing Internet access</td>
</tr>
<tr>
<td>DROP_QUEUE Procedure</td>
<td>Drops an existing queue</td>
</tr>
<tr>
<td>DROP_SHARDED_QUEUE Procedure</td>
<td>Drops an existing sharded queue from the database queuing system</td>
</tr>
<tr>
<td>DROP_QUEUE_TABLE Procedure</td>
<td>Drops an existing queue table</td>
</tr>
<tr>
<td>ENABLE_DB_ACCESS Procedure</td>
<td>Grants an Oracle Database Advanced Queuing Internet agent the privileges of a specific database user</td>
</tr>
<tr>
<td>ENABLE_JMS_TYPES Procedure</td>
<td>A precondition for the enqueue of JMS types and XML types</td>
</tr>
<tr>
<td>ENABLE_PROPAGATION_SCHEDULE Procedure</td>
<td>Enables a previously disabled propagation schedule</td>
</tr>
<tr>
<td>GET_MAX_STREAMS_POOL Procedure</td>
<td>Retrieves the value of Oracle Database Advanced Queuing maximum streams pool memory limit</td>
</tr>
<tr>
<td>GET_MIN_STREAMS_POOL Procedure</td>
<td>Retrieves the value of Oracle Database Advanced Queuing minimum streams pool memory limit</td>
</tr>
<tr>
<td>GET_QUEUE_PARAMETER Procedure</td>
<td>Used to get different parameters for sharded queues at queue or database level.</td>
</tr>
<tr>
<td>GET_WATERMARK Procedure</td>
<td>Retrieves the value of watermark set by the SET_WATERMARK Procedure</td>
</tr>
<tr>
<td>GRANT_QUEUE_PRIVILEGE Procedure</td>
<td>Grants privileges on a queue to users and roles</td>
</tr>
<tr>
<td>GRANT_SYSTEM_PRIVILEGE Procedure</td>
<td>Grants Oracle Database Advanced Queuing system privileges to users and roles</td>
</tr>
<tr>
<td>MIGRATE_QUEUE_TABLE Procedure</td>
<td>Upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table</td>
</tr>
<tr>
<td>PURGE_QUEUE_TABLE Procedure</td>
<td>Purges messages from queue tables</td>
</tr>
<tr>
<td>QUEUE_SUBSCRIBERS Function</td>
<td>Returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type</td>
</tr>
<tr>
<td>REMOVE_SUBSCRIBER Procedure</td>
<td>Removes a default subscriber from a queue</td>
</tr>
<tr>
<td>REVOKE_QUEUE_PRIVILEGE Procedure</td>
<td>Revokes privileges on a queue from users and roles</td>
</tr>
<tr>
<td>REVOKE_SYSTEM_PRIVILEGE Procedure</td>
<td>Revokes Oracle Database Advanced Queuing system privileges from users and roles</td>
</tr>
<tr>
<td>SCHEDULE_PROPAGATION Procedure</td>
<td>Schedules propagation of messages from a queue to a destination identified by a specific database link</td>
</tr>
</tbody>
</table>
Table 22-10  (Cont.) DBMS_AQADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_WATERMARK Procedure</td>
<td>Used for Oracle Database Advanced Queuing notification to specify and limit memory use</td>
</tr>
<tr>
<td>SET_MAX_STREAMS_POOL Procedure</td>
<td>Used for Oracle Database Advanced Queuing to specify and limit maximum streams pool memory use</td>
</tr>
<tr>
<td>SET_MIN_STREAMS_POOL Procedure</td>
<td>Used for Oracle Database Advanced Queuing to specify and limit minimum streams pool memory use</td>
</tr>
<tr>
<td>SET_QUEUE_PARAMETER Procedure</td>
<td>Used to set different parameters for sharded queues at queue or database level.</td>
</tr>
<tr>
<td>START_QUEUE Procedure</td>
<td>Enables the specified queue for enqueuing or dequeuing</td>
</tr>
<tr>
<td>STOP_QUEUE Procedure</td>
<td>Disables enqueuing or dequeuing on the specified queue</td>
</tr>
<tr>
<td>UNSCHEDULE_PROPAGATION Procedure</td>
<td>Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link</td>
</tr>
<tr>
<td>UNSET_QUEUE_PARAMETER Procedure</td>
<td>Used to unset different parameters for sharded queues at queue or database level.</td>
</tr>
<tr>
<td>VERIFY_QUEUE_TYPES Procedure</td>
<td>Verifies that the source and destination queues have identical types</td>
</tr>
</tbody>
</table>

22.4.1 ADD_ALIAS_TO_LDAP Procedure

This procedure creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP. The alias will be placed directly under the database server's distinguished name in LDAP hierarchy.

Syntax

```
DBMS_AQADM.ADD_ALIAS_TO_LDAP(
    alias          IN VARCHAR2,
    obj_location   IN VARCHAR2);
```

Parameters

Table 22-11  ADD_ALIAS_TO_LDAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Name of the alias. Example: west_shipping.</td>
</tr>
<tr>
<td>obj_location</td>
<td>The distinguished name of the object (queue, agent or connection factory) to which alias refers.</td>
</tr>
</tbody>
</table>
Usage Notes

This method can be used to create aliases for queues, agents, and JMS ConnectionFactory objects. These object must exist before the alias is created. These aliases can be used for JNDI lookup in JMS and Oracle Database Advanced Queuing Internet access.

22.4.2 ADD_SUBSCRIBER Procedure

This procedure adds a default subscriber to a queue.

Syntax

```
DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name      IN    VARCHAR2,
    subscriber      IN    sys.aq$_agent,
    rule            IN    VARCHAR2 DEFAULT NULL,
    transformation  IN    VARCHAR2 DEFAULT NULL,
    queue_to_queue  IN    BOOLEAN DEFAULT FALSE,
    delivery_mode   IN    PLS_INTEGER DEFAULT DBMS_AQADM.PERSISTENT);
```

Parameters

Table 22-12  ADD_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>subscriber</td>
<td>Agent on whose behalf the subscription is being defined.</td>
</tr>
<tr>
<td>rule</td>
<td>A conditional expression based on the message properties, the message data</td>
</tr>
<tr>
<td></td>
<td>properties and PL/SQL functions. A rule is specified as a Boolean expression</td>
</tr>
<tr>
<td></td>
<td>using syntax similar to the WHERE clause of a SQL query. This Boolean</td>
</tr>
<tr>
<td></td>
<td>expression can include conditions on message properties, user data</td>
</tr>
<tr>
<td></td>
<td>properties (object payloads only), and PL/SQL or SQL functions (as specified</td>
</tr>
<tr>
<td></td>
<td>in the where clause of a SQL query). Currently supported message</td>
</tr>
<tr>
<td></td>
<td>properties are priority and corrId.</td>
</tr>
<tr>
<td></td>
<td>To specify rules on a message payload (object payload), use attributes of</td>
</tr>
<tr>
<td></td>
<td>the object type in clauses. You must prefix each attribute with tab.user_data</td>
</tr>
<tr>
<td></td>
<td>as a qualifier to indicate the specific column of the queue table that</td>
</tr>
<tr>
<td></td>
<td>stores the payload. The rule parameter cannot exceed 4000 characters.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies a transformation that will be applied when this subscriber</td>
</tr>
<tr>
<td></td>
<td>dequeues the message. The source type of the transformation must match the</td>
</tr>
<tr>
<td></td>
<td>type of the queue. If the subscriber is remote, then the transformation is</td>
</tr>
<tr>
<td></td>
<td>applied before propagation to the remote queue.</td>
</tr>
<tr>
<td>queue_to_queue</td>
<td>If TRUE, propagation is from queue-to-queue.</td>
</tr>
<tr>
<td>delivery_mode</td>
<td>The administrator may specify one of DBMS_AQADM.PERSISTENT,</td>
</tr>
<tr>
<td></td>
<td>DBMS_AQADM.BUFFERED, or DBMS_AQADM.PERSISTENT OR BUFFERED for the delivery</td>
</tr>
<tr>
<td></td>
<td>mode of the messages the subscriber is interested in. This parameter will</td>
</tr>
<tr>
<td></td>
<td>not be modifiable by ALTER_SUBSCRIBER.</td>
</tr>
</tbody>
</table>
Usage Notes

A program can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation only succeeds on queues that allow multiple consumers. This operation takes effect immediately, and the containing transaction is committed. Enqueue requests that are executed after the completion of this call will reflect the new behavior.

Any string within the rule must be quoted:

```
rule => 'PRIORITY <= 3 AND CORRID = ''FROM JAPAN''
```

Note that these are all single quotation marks.

22.4.3 ALTER_AQ_AGENT Procedure

This procedure alters an agent registered for Oracle Database Advanced Queuing Internet access. It is also used to alter an Oracle Database Advanced Queuing agent that accesses secure queues.

![See Also:](https://oracle.com)

Oracle Streams Concepts and Administration for information about secure queues

Syntax

```sql
DBMS_AQADM.ALTER_AQ_AGENT (  
  agent_name                IN VARCHAR2,  
  certificate_location      IN VARCHAR2 DEFAULT NULL,  
  enable_http               IN BOOLEAN DEFAULT FALSE,  
  enable_smtp               IN BOOLEAN DEFAULT FALSE,  
  enable_ssnyp              IN BOOLEAN DEFAULT FALSE )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the Oracle Database Advanced Queuing Internet agent.</td>
</tr>
<tr>
<td>certificate_location</td>
<td>Agent's certificate location in LDAP (default is NULL). If the agent is allowed to access Oracle Database Advanced Queuing through SMTP, then its certificate must be registered in LDAP. For access through HTTP, the certificate location is not required.</td>
</tr>
<tr>
<td>enable_http</td>
<td>TRUE means the agent can access Oracle Database Advanced Queuing through HTTP. FALSE means the agent cannot access Oracle Database Advanced Queuing through HTTP.</td>
</tr>
<tr>
<td>enable_smtp</td>
<td>TRUE means the agent can access Oracle Database Advanced Queuing through SMTP (e-mail). FALSE means the agent cannot access Oracle Database Advanced Queuing through SMTP.</td>
</tr>
</tbody>
</table>
Table 22-13 (Cont.) ALTER_AQ_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_anyp</td>
<td>TRUE means the agent can access Oracle Database Advanced Queuing through any protocol (HTTP or SMTP).</td>
</tr>
</tbody>
</table>

22.4.4 ALTER_PROPAGATION_SCHEDULE Procedure

This procedure alters parameters for a propagation schedule.

Syntax

```sql
DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE (
    queue_name           IN    VARCHAR2,
    destination          IN    VARCHAR2 DEFAULT NULL,
    duration             IN    NUMBER   DEFAULT NULL,
    next_time            IN    VARCHAR2 DEFAULT NULL,
    latency              IN    NUMBER   DEFAULT 60,
    destination_queue    IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 22-14 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.</td>
</tr>
<tr>
<td>destination</td>
<td>Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.</td>
</tr>
<tr>
<td>next_time</td>
<td>Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, <code>next_time</code> should be specified as <code>SYSDATE + 1 - duration/86400</code>.</td>
</tr>
<tr>
<td>latency</td>
<td>Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. The default value is 60. Caution: if latency is not specified for this call, then latency will over-write any existing value with the default value. For example, if the latency is 60 seconds and there are no messages to be propagated during the propagation window, then messages from that queue for the destination are not propagated for at least 60 more seconds. It will be at least 60 seconds before the queue will be checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue will not be checked for 10 minutes and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination and as soon as a message is enqueued it will be propagated.</td>
</tr>
</tbody>
</table>
Table 22-14  (Cont.) ALTER_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_queue</td>
<td>Name of the target queue to which messages are to be propagated in the form of a dblink</td>
</tr>
</tbody>
</table>

22.4.5 ALTER_QUEUE Procedure

This procedure alters existing properties of a queue. The parameters max_retries, retention_time, and retry_delay are not supported for nonpersistent queues.

Syntax

```sql
DBMS_AQADM.ALTER_QUEUE (  
    queue_name        IN    VARCHAR2,  
    max_retries       IN    NUMBER   DEFAULT NULL,  
    retry_delay       IN    NUMBER   DEFAULT NULL,  
    retention_time    IN    NUMBER   DEFAULT NULL,  
    auto_commit       IN    BOOLEAN  DEFAULT TRUE,  
    comment           IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 22-15  ALTER_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue that is to be altered</td>
</tr>
<tr>
<td>max_retries</td>
<td>Limits the number of times a dequeue with REMOVE mode can be attempted on a message. The maximum value of max_retries is 2**31 -1. A message is moved to an exception queue if RETRY_COUNT is greater than MAX_RETRIES. RETRY_COUNT is incremented when the application issues a rollback after executing the dequeue. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then RETRY_COUNT is not incremented. Note that max_retries is supported for all single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.</td>
</tr>
<tr>
<td>retry_delay</td>
<td>Delay time in seconds before this message is scheduled for processing again after an application rollback. The default is NULL, which means that the value will not be altered. Note that retry_delay is supported for single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.</td>
</tr>
<tr>
<td>retention_time</td>
<td>Retention time in seconds for which a message is retained in the queue table after being dequeued. The default is NULL, which means that the value will not be altered.</td>
</tr>
</tbody>
</table>
Table 22-15  (Cont.) ALTER_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_commit</td>
<td>TRUE causes the current transaction, if any, to commit before the ALTER_QUEUE operation is carried out. The ALTER_QUEUE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.</td>
</tr>
<tr>
<td></td>
<td>Caution: This parameter has been deprecated.</td>
</tr>
<tr>
<td>comment</td>
<td>User-specified description of the queue. This user comment is added to the queue catalog. The default value is NULL, which means that the value will not be changed.</td>
</tr>
</tbody>
</table>

22.4.6 ALTER_QUEUE_TABLE Procedure

This procedure alters the existing properties of a queue table.

Syntax

```sql
DBMS_AQADM.ALTER_QUEUE_TABLE (
    queue_table          IN   VARCHAR2,
    comment              IN   VARCHAR2       DEFAULT NULL,
    primary_instance     IN   BINARY_INTEGER DEFAULT NULL,
    secondary_instance   IN   BINARY_INTEGER DEFAULT NULL,
    replication_mode     IN   BINARY_INTEGER DEFAULT NULL);
```

Parameters

Table 22-16  ALTER_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Name of a queue table to be created.</td>
</tr>
<tr>
<td>comment</td>
<td>Modifies the user-specified description of the queue table. This user comment is added to the queue catalog. The default value is NULL, which means that the value will not be changed.</td>
</tr>
<tr>
<td>primary_instance</td>
<td>This is the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table will be done in this instance. The default value is NULL, which means that the current value will not be changed.</td>
</tr>
<tr>
<td>secondary_instance</td>
<td>The queue table fails over to the secondary instance if the primary instance is not available. The default value is NULL, which means that the current value will not be changed.</td>
</tr>
<tr>
<td>replication_mode</td>
<td>Reserved for future use. DBMS_AQADM.REPLICATION_MODE if queue is being altered to be in the Replication Mode. DBMS_AQADM.NONE if queue is being altered to not be replicated. The default value is NULL.</td>
</tr>
</tbody>
</table>
22.4.7 ALTER_SHARDED_QUEUE Procedure

This procedure provides user the ability to alter a sharded queue.

See Also:
Oracle® Database Advanced Queuing User's Guide for information about sharded queues

Syntax

PROCEDURE ALTER_SHARDED_QUEUE(
    queue_name             IN VARCHAR2,
    max_retries            IN NUMBER         DEFAULT NULL,
    comment                IN VARCHAR2       DEFAULT NULL,
    queue_properties       IN QUEUE_PROPS_T  DEFAULT NULL,
    replication_mode       IN BINARY_INTEGER DEFAULT NULL);

Parameters

Table 22-17  ALTR_SHARDED_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>This parameter specifies the name of the sharded queue. A maximum of 128 characters are allowed.</td>
</tr>
<tr>
<td>max_retries</td>
<td>The maximum number of retries allowed.</td>
</tr>
<tr>
<td>comment</td>
<td>The comment of the queue.</td>
</tr>
<tr>
<td>queue_properties</td>
<td>Properties such as Normal or Exception Queue, Retry delay, retention time, sort list and cache hint. Refer to QUEUE_PROPS_T Type for more information about queue_properties.</td>
</tr>
<tr>
<td>replication_mode</td>
<td>Reserved for future use. DBMS_AQADM.REPLICATION_MODE if queue is being altered to be in the Replication Mode or else DBMS_AQADM.NONE. Default is NULL.</td>
</tr>
</tbody>
</table>

22.4.8 ALTER_SUBSCRIBER Procedure

This procedure alters existing properties of a subscriber to a specified queue. Only the rule can be altered.

Syntax

DBMS_AQADM.ALTER_SUBSCRIBER (  
    queue_name     IN VARCHAR2,
    subscriber     IN sys.aq$_agent,
    rule           IN VARCHAR2
    transformation IN VARCHAR2);
Parameters

Table 22-18 ALTER_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>subscriber</td>
<td>Agent on whose behalf the subscription is being altered. See “AQ__AGENT Type”.</td>
</tr>
<tr>
<td>rule</td>
<td>A conditional expression based on the message properties, the message data properties and PL/SQL functions. The rule parameter cannot exceed 4000 characters. To eliminate the rule, set the rule parameter to NULL.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies a transformation that will be applied when this subscriber dequeues the message. The source type of the transformation must match the type of the queue. If the subscriber is remote, then the transformation is applied before propagation to the remote queue.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure alters both the rule and the transformation for the subscriber. If you want to retain the existing value for either of them, you must specify its old value. The current values for rule and transformation for a subscriber can be obtained from the schema.AQ$queue_table_R and schema.AQ$queue_table_S views.

22.4.9 CREATE_AQ_AGENT Procedure

This procedure registers an agent for Oracle Database Advanced Queuing Internet access using HTTP/SMTP protocols. It is also used to create an Oracle Database Advanced Queuing agent to access secure queues.

See Also:

Oracle Streams Concepts and Administration for information about secure queues

Syntax

```sql
DBMS_AQADM.CREATE_AQ_AGENT ( 
  agent_name IN VARCHAR2, 
  certificate_location IN VARCHAR2 DEFAULT NULL, 
  enable_http IN BOOLEAN DEFAULT FALSE, 
  enable_smtp IN BOOLEAN DEFAULT FALSE, 
  enable_anyp IN BOOLEAN DEFAULT FALSE )
```
Parameters

Table 22-19  CREATE_AQ_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the Oracle Database Advanced Queuing Internet agent.</td>
</tr>
<tr>
<td>certification_location</td>
<td>Agent's certificate location in LDAP (default is NULL). If the agent is allowed to access Oracle Database Advanced Queuing through SMTP, then its certificate must be registered in LDAP. If access through HTTP, the certificate location is not required.</td>
</tr>
<tr>
<td>enable_http</td>
<td>TRUE means the agent can access Oracle Database Advanced Queuing through HTTP. FALSE means the agent cannot access Oracle Database Advanced Queuing through HTTP.</td>
</tr>
<tr>
<td>enable_smtp</td>
<td>TRUE means the agent can access Oracle Database Advanced Queuing through SMTP (e-mail). FALSE means the agent cannot access Oracle Database Advanced Queuing through SMTP.</td>
</tr>
<tr>
<td>enable_anyp</td>
<td>TRUE means the agent can access Oracle Database Advanced Queuing through any protocol (HTTP or SMTP).</td>
</tr>
</tbody>
</table>

Usage Notes

The SYS.AQ$INTERNET_USERS view has a list of all Oracle Database Advanced Queuing Internet agents.

22.4.10 CREATE_NP_QUEUE Procedure

This procedure creates a nonpersistent RAW queue.

Note:

Nonpersistent queues are deprecated as of Release 10gR2. Oracle recommends using buffered messaging.

Syntax

```sql
DBMS_AQADM.CREATE_NP_QUEUE ( queue_name IN VARCHAR2, multiple_consumers IN BOOLEAN DEFAULT FALSE, comment IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 22-20  CREATE_NP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the nonpersistent queue that is to be created. The name must be unique within a schema and must follow object name guidelines in Oracle Database SQL Language Reference.</td>
</tr>
</tbody>
</table>
Table 22-20  (Cont.) CREATE_NP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple_consumers</td>
<td>FALSE means queues created in the table can only have one consumer for each message. This is the default. TRUE means queues created in the table can have multiple consumers for each message. Note that this parameter is distinguished at the queue level, because a nonpersistent queue does not inherit this characteristic from any user-created queue table.</td>
</tr>
<tr>
<td>comment</td>
<td>User-specified description of the queue. This user comment is added to the queue catalog.</td>
</tr>
</tbody>
</table>

Usage Notes

The queue may be either single-consumer or multiconsumer queue. All queue names must be unique within a schema. The queues are created in a 8.1-compatible or higher system-created queue table (AQ$_MEM_SC or AQ$_MEM_MC) in the same schema as that specified by the queue name.

If the queue name does not specify a schema name, the queue is created in the login user's schema. After a queue is created with CREATE_NP_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

You cannot dequeue from a nonpersistent queue. The only way to retrieve a message from a nonpersistent queue is by using the OCI notification mechanism. You cannot invoke the LISTEN call on a nonpersistent queue.

22.4.11 CREATE_QUEUE Procedure

This procedure creates a queue in the specified queue table.

Syntax

```sql
DBMS_AQADM.CREATE_QUEUE (  
    queue_name          IN       VARCHAR2,  
    queue_table         IN       VARCHAR2,  
    queue_type          IN       BINARY_INTEGER DEFAULT NORMAL_QUEUE,  
    max_retries         IN       NUMBER         DEFAULT NULL,  
    retry_delay         IN       NUMBER         DEFAULT 0,  
    retention_time      IN       NUMBER         DEFAULT 0,  
    dependency_tracking IN       BOOLEAN        DEFAULT FALSE,  
    comment             IN       VARCHAR2       DEFAULT NULL,  
    auto_commit         IN       BOOLEAN        DEFAULT TRUE);
```
### Parameters

**Table 22-21** CREATE_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>queue_name</code></td>
<td>Name of the queue that is to be created. The name must be unique within a schema and must follow object name guidelines in <em>Oracle Database SQL Language Reference</em> with regard to reserved characters.</td>
</tr>
<tr>
<td><code>queue_table</code></td>
<td>Name of the queue table that will contain the queue.</td>
</tr>
<tr>
<td><code>queue_type</code></td>
<td>Specifies whether the queue being created is an exception queue or a normal queue. <code>NORMAL_QUEUE</code> means the queue is a normal queue. This is the default. <code>EXCEPTION_QUEUE</code> means it is an exception queue. Only the dequeue operation is allowed on the exception queue.</td>
</tr>
<tr>
<td><code>max_retries</code></td>
<td>Limits the number of times a dequeue with the REMOVE mode can be attempted on a message. The maximum value of <code>max_retries</code> is $2^{31} - 1$. A message is moved to an exception queue if <code>RETRY_COUNT</code> is greater than <code>MAX_RETRIES</code>. Only the dequeue operation is allowed on the exception queue.</td>
</tr>
<tr>
<td><code>retry_delay</code></td>
<td>Delay time, in seconds, before this message is scheduled for processing again after an application rollback. The default is 0, which means the message can be retried as soon as possible. This parameter has no effect if <code>max_retries</code> is set to 0. Note that <code>retry_delay</code> is supported for single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.</td>
</tr>
<tr>
<td><code>retention_time</code></td>
<td>Number of seconds for which a message is retained in the queue table after being dequeued from the queue. <code>INFINITE</code> means the message is retained forever. <code>NUMBER</code> is the number of seconds for which to retain the messages. The default is 0, no retention.</td>
</tr>
<tr>
<td><code>dependency_tracking</code></td>
<td>Reserved for future use. <code>FALSE</code> is the default. <code>TRUE</code> is not permitted in this release.</td>
</tr>
<tr>
<td><code>comment</code></td>
<td>User-specified description of the queue. This user comment is added to the queue catalog.</td>
</tr>
<tr>
<td><code>auto_commit</code></td>
<td><code>TRUE</code> causes the current transaction, if any, to commit before the CREATE_QUEUE operation is carried out. The CREATE_QUEUE operation becomes persistent when the call returns. This is the default. <code>FALSE</code> means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.</td>
</tr>
</tbody>
</table>

Caution: This parameter has been deprecated.
Usage Notes

All queue names must be unique within a schema. After a queue is created with \texttt{CREATE\_QUEUE}, it can be enabled by calling \texttt{START\_QUEUE}. By default, the queue is created with both enqueue and dequeue disabled.

22.4.12 CREATE\_QUEUE\_TABLE Procedure

This procedure creates a queue table for messages of a predefined type.

Syntax

\begin{verbatim}
DBMS_AQADM.CREATE_QUEUE_TABLE (  
  queue_table          IN      VARCHAR2,  
  queue_payload_type   IN      VARCHAR2,  
  storage_clause       IN      VARCHAR2 DEFAULT NULL,  
  sort_list            IN      VARCHAR2 DEFAULT NULL,  
  multiple_consumers   IN      BOOLEAN DEFAULT FALSE,  
  message_grouping     IN      BINARY_INTEGER DEFAULT NONE,  
  comment              IN      VARCHAR2 DEFAULT NULL,  
  auto_commit          IN      BOOLEAN DEFAULT TRUE,  
  primary_instance     IN      BINARY_INTEGER DEFAULT 0,  
  secondary_instance   IN      BINARY_INTEGER DEFAULT 0,  
  compatible           IN      VARCHAR2 DEFAULT NULL,  
  secure               IN      BOOLEAN DEFAULT FALSE  
  replication_mode     IN      BINARY_INTEGER DEFAULT NONE); 
\end{verbatim}

Parameters

Table 22-22 CREATE\_QUEUE\_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Name of a queue table to be created</td>
</tr>
<tr>
<td>queue_payload_type</td>
<td>Type of the user data stored. See Type Name in DBMS_AQ Data Types for valid values for this parameter.</td>
</tr>
<tr>
<td>storage_clause</td>
<td>Storage parameter. The storage parameter is included in the CREATE TABLE statement when the queue table is created. The storage_clause argument can take any text that can be used in a standard CREATE TABLE storage_clause argument. The storage parameter can be made up of any combinations of the following parameters: PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLESPACE, LOB, and a table storage clause. If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. See Oracle Database SQL Language Reference for the usage of these parameters.</td>
</tr>
</tbody>
</table>
Table 22-22  (Cont.) CREATE_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort_list</td>
<td>The columns to be used as the sort key in ascending order. This parameter has the following format:</td>
</tr>
<tr>
<td></td>
<td>&quot;sort_column_1,sort_column_2&quot;</td>
</tr>
<tr>
<td></td>
<td>The allowed column names are priority, enq_time, and commit_time. If both columns are specified, then sort_column_1 defines the most significant order.</td>
</tr>
<tr>
<td></td>
<td>After a queue table is created with a specific ordering mechanism, all queues in the queue table inherit the same defaults. The order of a queue table cannot be altered after the queue table has been created.</td>
</tr>
<tr>
<td></td>
<td>If no sort list is specified, then all the queues in this queue table are sorted by the enqueue time in ascending order. This order is equivalent to FIFO order.</td>
</tr>
<tr>
<td></td>
<td>Even with the default ordering defined, a dequeeuer is allowed to choose a message to dequeue by specifying its msgid or correlation.msgid,correlation, and sequence_deviation take precedence over the default dequeueing order, if they are specified.</td>
</tr>
<tr>
<td></td>
<td>When commit_time is specified for the sort_list parameter the resulting queue table uses commit-time ordering. See &quot;Commit-Time Queues&quot; in Oracle Streams Concepts and Administration for more information about commit-time ordering.</td>
</tr>
<tr>
<td></td>
<td>See also &quot;Priority and Ordering of Messages&quot; in Oracle Database Advanced Queuing User's Guide for information about message ordering in Oracle Database Advanced Queuing.</td>
</tr>
<tr>
<td>multiple_consumers</td>
<td>FALSE means queues created in the table can only have one consumer for each message. This is the default. TRUE means queues created in the table can have multiple consumers for each message.</td>
</tr>
<tr>
<td>message_grouping</td>
<td>Message grouping behavior for queues created in the table. NONE means each message is treated individually. TRANSACTIONAL means messages enqueued as part of one transaction are considered part of the same group and can be dequeued as a group of related messages.</td>
</tr>
<tr>
<td>comment</td>
<td>User-specified description of the queue table. This user comment is added to the queue catalog.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>TRUE causes the current transaction, if any, to commit before the CREATE_QUEUE_TABLE operation is carried out. The CREATE_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.</td>
</tr>
<tr>
<td></td>
<td>Note: This parameter has been deprecated.</td>
</tr>
<tr>
<td>primary_instance</td>
<td>The primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance.</td>
</tr>
<tr>
<td></td>
<td>The default value for primary instance is 0, which means queue monitor scheduling and propagation will be done in any available instance.</td>
</tr>
</tbody>
</table>
Table 22-22  (Cont.) CREATE_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondary_instance</td>
<td>The queue table fails over to the secondary instance if the primary instance is not available. The default value is 0, which means that the queue table will fail over to any available instance.</td>
</tr>
<tr>
<td>compatible</td>
<td>The lowest database version with which the queue is compatible. Currently the possible values are either 8.0, 8.1, or 10.0. If the database is in 10.1-compatible mode, the default value is 10.0. If the database is in 8.1-compatible or 9.2-compatible mode, the default value is 8.1. If the database is in 8.0 compatible mode, the default value is 8.0.</td>
</tr>
<tr>
<td>secure</td>
<td>This parameter must be set to TRUE if you want to use the queue table for secure queues. Secure queues are queues for which AQ agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue. The owner of a secure queue can perform all queue operations on the queue, but other users cannot perform queue operations on a secure queue, unless they are configured as secure queue users.</td>
</tr>
<tr>
<td>replication_mode</td>
<td>Reserved for future use. DBMS_AQADM.REPLICATION_MODE if queue is being created in the Replication Mode or else DBMS_AQADM.NONE. Default is DBMS_AQADM.NONE.</td>
</tr>
</tbody>
</table>

Usage Notes

The sort keys for dequeue ordering, if any, must be defined at table creation time. The following objects are created at this time:

- **aq$$_queue_table_name_e**, a default exception queue associated with the queue table
- **aq$queue_table_name**, a read-only view, which is used by Oracle Database Advanced Queuing applications for querying queue data
- **aq$ queue_table_name_t**, an index (or an index organized table (IOT) in the case of multiple consumer queues) for the queue monitor operations
- **aq$ queue_table_name_i**, an index (or an index organized table in the case of multiple consumer queues) for dequeue operations

For 8.1-compatible or higher queue tables, the following index-organized tables are created:

- **aq$$_queue_table_name_s**, a table for storing information about the subscribers
- **aq$$_queue_table_name_r**, a table for storing information about rules on subscriptions
- **aq$$_queue_table_name_h**, an index-organized table for storing the dequeue history data

CLOB, BLOB, and BFILE are valid attributes for Oracle Database Advanced Queuing object type payloads. However, only CLOB and BLOB can be propagated using Oracle Database Advanced Queuing propagation in Oracle8i release 8.1.5 or later. See the Oracle Database Advanced Queuing User’s Guide for more information.
The default value of the compatible parameter depends on the database compatibility mode in the `init.ora`. If the database is in 10.1-compatible mode, the default value is 10.0. If the database is in 8.1-compatible or 9.2-compatible mode, the default value is 8.1. If the database is in 8.0 compatible mode, the default value is 8.0.

You can specify and modify the `primary_instance` and `secondary_instance` only in 8.1-compatible or higher mode. You cannot specify a secondary instance unless there is a primary instance.

22.4.13 CREATE_SHARDED_QUEUE Procedure

The `CREATE_SHARDED_QUEUE` API creates a queue and its queue table as appropriate for a sharded queue. This API cannot be used to create unsharded queues. Sharded queues must be created using this single integrated API that will automatically set AQ properties as needed.

Sharded queues may be either a single consumer or a multi-consumer queue.

Syntax

```sql
PROCEDURE CREATE_SHARDED_QUEUE (  
    queue_name             IN VARCHAR2,  
    storage_clause         IN VARCHAR2       DEFAULT NULL,  
    multiple_consumers     IN BOOLEAN        DEFAULT FALSE,  
    max_retries            IN NUMBER         DEFAULT NULL,  
    comment                IN VARCHAR2       DEFAULT NULL,  
    queue_payload_type     IN VARCHAR2       DEFAULT JMS_TYPE,  
    queue_properties       IN QUEUE_PROPS_T  DEFAULT NULL,  
    replication_mode       IN BINARY_INTEGER DEFAULT NONE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>This required parameter specifies the name of the new queue. Maximum of 128 characters allowed.</td>
</tr>
<tr>
<td>storage_clause</td>
<td>The storage parameter is included in the <code>CREATE TABLE</code> statement when the queue table is created. The <code>storage_clause</code> argument can take any text that can be used in a standard <code>CREATE TABLE</code> <code>storage_clause</code> argument. The storage parameter can be made up of any combinations of the following parameters: <code>PCTFREE</code>, <code>PCTUSED</code>, <code>INITRANS</code>, <code>MAXTRANS</code>, <code>TABLESPACE</code>, <code>LOB</code>, and a table storage clause. If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. See Oracle Database SQL Language Reference for the usage of these parameters.</td>
</tr>
<tr>
<td>multiple_consumers</td>
<td><code>FALSE</code> means queues can only have one consumer for each message. <code>TRUE</code> means queues created in the table can have multiple consumers for each message.</td>
</tr>
</tbody>
</table>
Table 22-23  (Cont.) CREATE_SHARDED_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_retries</td>
<td>This optional parameter limits the number of times that a dequeue can reattempted on a message after a failure. The maximum value of max_retries is $2^{31} - 1$. After the retry limit has been exceeded, the message will be purged from the queue. RETRY_COUNT is incremented when the application issues a rollback after executing the dequeue. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then RETRY_COUNT is not incremented.</td>
</tr>
<tr>
<td>comment</td>
<td>This optional parameter is a user-specified description of the queue table. This user comment is added to the queue catalog.</td>
</tr>
<tr>
<td>queue_payload_type</td>
<td>Payload can be RAW, DBMS_AQADM.JMS_TYPE, or an object type. Default is DBMS_AQADM.JMS_TYPE.</td>
</tr>
<tr>
<td>queue_properties</td>
<td>Properties such as Normal or Exception Queue, Retry delay, retention time, sort list and cache hint. Refer to QUEUE_PROPS_T Type for more information about queue_properties.</td>
</tr>
<tr>
<td>replication_mode</td>
<td>Reserved for future use. DBMS_AQADM.REPLICATION_MODE if Queue is being created in the Replication Mode or else DBMS_AQADM.NONE. Default is DBMS_AQADM.NONE.</td>
</tr>
</tbody>
</table>

22.4.14 CREATE_EXCEPTION_QUEUE Procedure

The CREATE_EXCEPTION_QUEUE API creates an exception queue for a sharded queue.

Syntax

```
PROCEDURE CREATE_EXCEPTION_QUEUE(
    sharded_queue_name     IN VARCHAR2,
    exception_queue_name   IN VARCHAR2 DEFAULT NULL
);
```

Parameters

Table 22-24  CREATE_EXCEPTION_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sharded_queue_name</td>
<td>The name of the sharded queue.</td>
</tr>
<tr>
<td>exception_queue_name</td>
<td>The name of the exception queue.</td>
</tr>
</tbody>
</table>

22.4.15 DEL_ALIAS_FROM_LDAP Procedure

This procedure drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP.

Syntax

```
DBMS_AQADM.DEL_ALIAS_FROM_LDAP(
    alias IN VARCHAR2);
```
Parameters

Table 22-25  DEL_ALIAS_FROM_LDAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>The alias to be removed.</td>
</tr>
</tbody>
</table>

22.4.16 DISABLE_DB_ACCESS Procedure

This procedure revokes the privileges of a specific database user from an Oracle Database Advanced Queuing Internet agent.

Syntax

```
DBMS_AQADM.DISABLE_DB_ACCESS (
    agent_name                IN VARCHAR2,
    db_username               IN VARCHAR2)
```

Parameters

Table 22-26  DISABLE_DB_ACCESS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the Oracle Database Advanced Queuing Internet agent.</td>
</tr>
<tr>
<td>db_username</td>
<td>Specifies the database user whose privileges are to be revoked from the Oracle Database Advanced Queuing Internet agent.</td>
</tr>
</tbody>
</table>

Usage Notes

The Oracle Database Advanced Queuing Internet agent should have been previously granted those privileges using the ENABLE_DB_ACCESS Procedure.

22.4.17 DISABLE_PROPAGATION_SCHEDULE Procedure

This procedure disables a propagation schedule.

Syntax

```
DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
    queue_name            IN   VARCHAR2,
    destination           IN   VARCHAR2 DEFAULT NULL,
    destination_queue  IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 22-27  DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.</td>
</tr>
</tbody>
</table>
Table 22-27  (Cont.) DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.</td>
</tr>
<tr>
<td>destination_queue</td>
<td>Name of the target queue to which messages are to be propagated in the form of a dblink</td>
</tr>
</tbody>
</table>

22.4.18 DROP_AQ_AGENT Procedure

This procedure drops an agent that was previously registered for Oracle Database Advanced Queuing Internet access.

Syntax

```sql
DBMS_AQADM.DROP_AQ_AGENT ( 
    agent_name    IN VARCHAR2)
```

Parameters

Table 22-28  DROP_AQ_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the Oracle Database Advanced Queuing Internet agent</td>
</tr>
</tbody>
</table>

22.4.19 DROP_QUEUE Procedure

This procedure drops an existing queue.

Syntax

```sql
DBMS_AQADM.DROP_QUEUE ( 
    queue_name    IN VARCHAR2, 
    auto_commit   IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 22-29  DROP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue that is to be dropped.</td>
</tr>
</tbody>
</table>
Table 22-29 (Cont.) DROP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_commit</td>
<td>TRUE causes the current transaction, if any, to commit before the DROP_QUEUE operation is carried out. The DROP_QUEUE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.</td>
</tr>
</tbody>
</table>

Usage Notes

DROP_QUEUE is not allowed unless STOP_QUEUE has been called to disable the queue for both enqueuing and dequeuing. All the queue data is deleted as part of the drop operation.

22.4.20 DROP_QUEUE_TABLE Procedure

This procedure drops an existing queue table.

Syntax

```
DBMS_AQADM.DROP_QUEUE_TABLE ( 
    queue_table       IN    VARCHAR2, 
    force             IN    BOOLEAN DEFAULT FALSE, 
    auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 22-30 DROP_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Name of a queue table to be dropped.</td>
</tr>
<tr>
<td>force</td>
<td>FALSE means the operation does not succeed if there are any queues in the table. This is the default. TRUE means all queues in the table are stopped and dropped automatically.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>TRUE causes the current transaction, if any, to commit before the DROP_QUEUE_TABLE operation is carried out. The DROP_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.</td>
</tr>
</tbody>
</table>

Usage Notes

All the queues in a queue table must be stopped and dropped before the queue table can be dropped. You must do this explicitly unless the force option is used, in which case this is done automatically.
22.4.21 DROP_SHARDED_QUEUE Procedure

This procedure drops an existing sharded queue from the database queuing system.

You must stop the queue before calling DROP_SHARDED_QUEUE. User must stop the queue explicitly if force is set to FALSE before calling DROP_SHARDED_QUEUE. If force is set to TRUE then queue will be stopped internally and then dropped.

Syntax

DBMS_AQADM.DROP_SHARDED_QUEUE(
    queue_name IN VARCHAR2,
    force      IN BOOLEAN DEFAULT FALSE )

Parameters

Table 22-31  DROP_SHARDED_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>This required parameter specifies the name of the sharded queue.</td>
</tr>
<tr>
<td>force</td>
<td>The sharded queue is dropped even if the queue is not stopped.</td>
</tr>
</tbody>
</table>

22.4.22 ENABLE_DB_ACCESS Procedure

This procedure grants an Oracle Database Advanced Queuing Internet agent the privileges of a specific database user.

Syntax

DBMS_AQADM.ENABLE_DB_ACCESS (  
    agent_name                IN VARCHAR2,  
    db_username               IN VARCHAR2)  

Parameters

Table 22-32  ENABLE_DB_ACCESS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Specifies the username of the Oracle Database Advanced Queuing Internet agent.</td>
</tr>
<tr>
<td>db_username</td>
<td>Specified the database user whose privileges are to be granted to the Oracle Database Advanced Queuing Internet agent.</td>
</tr>
</tbody>
</table>

Usage Notes

The Oracle Database Advanced Queuing Internet agent should have been previously created using the CREATE_AQ_AGENT Procedure.

For secure queues, the sender and receiver agent of the message must be mapped to the database user performing the enqueue or dequeue operation.
The `SYS.AQ$INTERNET_USERS` view has a list of all Oracle Database Advanced Queuing Internet agents and the names of the database users whose privileges are granted to them.

### 22.4.23 ENABLE_JMS_TYPES Procedure

Enqueue of JMS types and XML types does not work with Oracle Streams `Sys.Anydata` queues unless you call this procedure after `DBMS_STREAMS_ADM.SET_UP_QUEUE`. Enabling an Oracle Streams queue for these types may affect import/export of the queue table.

**Syntax**

```sql
DBMS_AQADM.ENABLE_JMS_TYPES (
    queue_table   IN   VARCHAR2);
```

**Parameters**

**Table 22-33**  ENABLE_JMS_TYPES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Specifies name of the queue table to be enabled for JMS and XML types.</td>
</tr>
</tbody>
</table>

### 22.4.24 ENABLE_PROPAGATION_SCHEDULE Procedure

This procedure enables a previously disabled propagation schedule.

**Syntax**

```sql
DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE (
    queue_name         IN   VARCHAR2,
    destination        IN   VARCHAR2 DEFAULT NULL,
    destination_queue  IN   VARCHAR2  DEFAULT NULL);
```

**Parameters**

**Table 22-34**  ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.</td>
</tr>
<tr>
<td>destination</td>
<td>Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is <code>NULL</code>, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.</td>
</tr>
</tbody>
</table>
Table 22-34 (Cont.) ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_queue</td>
<td>Name of the target queue to which messages are to be propagated in the form of a dblink</td>
</tr>
</tbody>
</table>

22.4.25 GET_MAX_STREAMS_POOL Procedure

This procedure retrieves the value of Oracle Database Advanced Queuing maximum streams pool memory limit.

Syntax

```sql
DBMS_AQADM.GET_MAX_STREAMS_POOL (value OUT NUMBER);
```

Parameters

Table 22-35 GET_MAX_STREAMS_POOL Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value in megabytes.</td>
</tr>
</tbody>
</table>

22.4.26 GET_MIN_STREAMS_POOL Procedure

This procedure retrieves the value of Oracle Database Advanced Queuing minimum streams pool memory limit.

Syntax

```sql
DBMS_AQADM.GET_MIN_STREAMS_POOL (value OUT NUMBER);
```

Parameters

Table 22-36 GET_MIN_STREAMS_POOL Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value in megabytes.</td>
</tr>
</tbody>
</table>

22.4.27 GET_QUEUE_PARAMETER Procedure

This procedure allows user to get different parameters for sharded queues at queue or database level.

For database level the `queue_name` should be NULL. Note that queue overrides database level parameter values.
See Also:

*Oracle® Database Advanced Queuing User's Guide* for information about sharded queues

Syntax

```sql
PROCEDURE GET_QUEUE_PARAMETER(
    queue_name          IN VARCHAR2,
    param_name          IN VARCHAR2,
    param_value         OUT NUMBER);
```

Parameters

**Table 22-37 GET_QUEUE_PARAMETER Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the sharded queue.</td>
</tr>
<tr>
<td>param_name</td>
<td>The name of the parameter.</td>
</tr>
<tr>
<td>param_value</td>
<td>The value of the parameter.</td>
</tr>
</tbody>
</table>

22.4.28 GET_WATERMARK Procedure

This procedure retrieves the value of watermark set by `SET_WATERMARK`.

Syntax

```sql
DBMS_AQADM.GET_WATERMARK (
    wmvalue     IN      NUMBER);
```

Parameters

**Table 22-38 GET_WATERMARK Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmvalue</td>
<td>Watermark value in megabytes.</td>
</tr>
</tbody>
</table>

22.4.29 GRANT_QUEUE_PRIVILEGE Procedure

This procedure grants privileges on a queue to users and roles. The privileges are `ENQUEUE` or `DEQUEUE`. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

Syntax

```sql
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
    privilege        IN    VARCHAR2,
    queue_name       IN    VARCHAR2,
    grantee          IN    VARCHAR2,
    grant_option     IN    BOOLEAN := FALSE);
```
Parameters

Table 22-39  GRANT_QUEUE_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The Oracle Database Advanced Queuing queue privilege to grant. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.</td>
</tr>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.</td>
</tr>
<tr>
<td>grant_option</td>
<td>Specifies if the access privilege is granted with the GRANT option or not. If the privilege is granted with the GRANT option, then the grantee is allowed to use this procedure to grant the access privilege to other users or roles, regardless of the ownership of the queue table. The default is FALSE.</td>
</tr>
</tbody>
</table>

22.4.30 GRANT_SYSTEM_PRIVILEGE Procedure

This procedure grants Oracle Database Advanced Queuing system privileges to users and roles.

The privileges are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. Initially, only SYS and SYSTEM can use this procedure successfully.

Note:

Starting from Oracle Database 12c Release 2, MANAGE_ANY, ENQUEUE_ANY, and DEQUEUE_ANY privileges will not allow access to SYS owned queues by users other than SYS.

Syntax

```sql
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
  privilege IN VARCHAR2,
  grantee IN VARCHAR2,
  admin_option IN BOOLEAN := FALSE);
```
Parameters

Table 22-40  GRANT_SYSTEM_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The Oracle Database Advanced Queuing system privilege to grant. The options are ENQUEUE ANY, DEQUEUE ANY, and MANAGE ANY. ENQUEUE ANY means users granted this privilege are allowed to enqueue messages to any queues in the database. DEQUEUE ANY means users granted this privilege are allowed to dequeue messages from any queues in the database. MANAGE ANY means users granted this privilege are allowed to run DBMS_AQADM calls on any schemas in the database.</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.</td>
</tr>
<tr>
<td>admin_option</td>
<td>Specifies if the system privilege is granted with the ADMIN option or not. If the privilege is granted with the ADMIN option, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles. The default is FALSE.</td>
</tr>
</tbody>
</table>

Note:
Starting from Oracle Database 12c Release 2, MANAGE ANY, ENQUEUE ANY, and DEQUEUE ANY privileges will not allow access to SYS owned queues by users other than SYS.

22.4.31 MIGRATE_QUEUE_TABLE Procedure

This procedure upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table.

Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE (  
  queue_table   IN   VARCHAR2,  
  compatible    IN   VARCHAR2);  
```

Parameters

Table 22-41  MIGRATE_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Specifies name of the queue table to be migrated.</td>
</tr>
<tr>
<td>compatible</td>
<td>Set this to 8.1 to upgrade an 8.0-compatible queue table, or set this to 8.0 to downgrade an 8.1-compatible queue table.</td>
</tr>
</tbody>
</table>
22.4.32 PURGE_QUEUE_TABLE Procedure

This procedure purges messages from queue tables. You can perform various purge operations on both single-consumer and multiconsumer queue tables for persistent and buffered messages.

Syntax

```sql
DBMS_AQADM.PURGE_QUEUE_TABLE(
    queue_table        IN   VARCHAR2,
    purge_condition    IN   VARCHAR2,
    purge_options      IN   aq$_purge_options_t);
```

where type aq$_purge_options_t is described in Oracle Database Advanced Queuing (AQ) Types.

Parameters

Table 22-42  PURGE_QUEUE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>Specifies the name of the queue table to be purged.</td>
</tr>
<tr>
<td>purge_condition</td>
<td>Specifies the purge condition to use when purging the queue table. The purge condition must be in the format of a SQL WHERE clause, and it is case-sensitive. The condition is based on the columns of aq$queue_table_name view.</td>
</tr>
<tr>
<td>purge_options</td>
<td>Type aq$_purge_options_t contains a block parameter and a delivery_mode parameter.</td>
</tr>
<tr>
<td></td>
<td>- If block is TRUE, then an exclusive lock on all the queues in the queue table is held while purging the queue table. This will cause concurrent enqueuers and dequeuers to block while the queue table is purged. The purge call always succeeds if block is TRUE. The default for block is FALSE. This will not block enqueuers and dequeuers, but it can cause the purge to fail with an error during high concurrency times.</td>
</tr>
<tr>
<td></td>
<td>- delivery_mode is used to specify whether DBMS_AQADM.PERSISTENT, DBMS_AQADM.BUFFERED or DBMS_AQADM.PERSISTENT_OR_BUFFERED types of messages are to be purged. You cannot implement arbitrary purge conditions if buffered messages have to be purged.</td>
</tr>
</tbody>
</table>

T o purge all queues in a queue table, set purge_condition to either NULL (a bare null word, no quotes) or '' (two single quotes).
Usage Notes

- You can purge selected messages from the queue table by specifying a purge_condition. 
  Table 22-42 describes these parameters. Messages can be enqueued to and dequeued 
  from the queue table while the queue table is being purged.
- A trace file is generated in the udump destination when you run this procedure. It details 
  what the procedure is doing.
- This procedure commits batches of messages in autonomous transactions. Several such 
  autonomous transactions may get executed as a part of one purge_queue_table call 
  depending on the number of messages in the queue table.

22.4.33 QUEUE_SUBSCRIBERS Function

This function returns the subscribers to an 8.0-compatible multiconsumer queue in the 
PL/SQL index by table collection type DBMS_AQADM.AQ$_subscriber_list_t.

Each element of the collection is of type sys.aq$_agent. This functionality is provided for 8.1-
compatible queues by the AQ$_queue_table_name_S view.

Syntax

```sql
DBMS_AQADM.QUEUE_SUBSCRIBERS (queue_name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Specifies the queue whose subscribers are to be printed.</td>
</tr>
</tbody>
</table>

22.4.34 REMOVE_SUBSCRIBER Procedure

This procedure removes a default subscriber from a queue. This operation takes effect 
immediately, and the containing transaction is committed. All references to the subscriber in 
existing messages are removed as part of the operation.

Syntax

```sql
DBMS_AQADM.REMOVE_SUBSCRIBER (queue_name IN VARCHAR2,
                              subscriber IN sys.aq$_agent);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>subscriber</td>
<td>Agent who is being removed. See AQ$_AGENT Type.</td>
</tr>
</tbody>
</table>
22.4.35 REVOKE_QUEUE_PRIVILEGE Procedure

This procedure revokes privileges on a queue from users and roles. The privileges are
ENQUEUE or DEQUEUE.

Syntax

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (  
    privilege         IN      VARCHAR2,  
    queue_name        IN      VARCHAR2,  
    grantee           IN      VARCHAR2);
```

Parameters

Table 22-45  REVOKE_QUEUE_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The Oracle Database Advanced Queuing queue privilege to revoke. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.</td>
</tr>
<tr>
<td>queue_name</td>
<td>Name of the queue.</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role. If the privilege has been propagated by the grantee through the GRANT option, then the propagated privilege is also revoked.</td>
</tr>
</tbody>
</table>

Usage Notes

To revoke a privilege, the revoker must be the original grantor of the privilege. The privileges propagated through the GRANT option are revoked if the grantor’s privileges are revoked.

22.4.36 REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure revokes Oracle Database Advanced Queuing system privileges from users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked. Starting from Oracle Database 12c Release 2, MANAGE_ANY, ENQUEUE_ANY, and DEQUEUE_ANY privileges will not allow access to SYS owned queues by users other than SYS.

Syntax

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE (  
    privilege         IN      VARCHAR2,  
    grantee           IN      VARCHAR2);
```
Parameters

Table 22-46  REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The Oracle Database Advanced Queuing system privilege to revoke. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.</td>
</tr>
</tbody>
</table>

Note: Starting from Oracle Database 12c Release 2, MANAGE_ANY, ENQUEUE_ANY, and DEQUEUE_ANY privileges will not allow access to SYS owned queues by users other than SYS.

grantee  Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.

22.4.37 SCHEDULE_PROPAGATION Procedure

This procedure schedules propagation of messages from a queue to a destination identified by a specific database link.

Syntax

```sql
DBMS_AQADM.SCHEDULE_PROPAGATION (
    queue_name          IN    VARCHAR2,
    destination         IN    VARCHAR2 DEFAULT NULL,
    start_time          IN    DATE     DEFAULT SYSDATE,
    duration            IN    NUMBER   DEFAULT NULL,
    next_time           IN    VARCHAR2 DEFAULT NULL,
    latency             IN    NUMBER   DEFAULT 60,
    destination_queue   IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 22-47  SCHEDULE_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.</td>
</tr>
</tbody>
</table>
### Table 22-47  (Cont.) SCHEDULE_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 390 bytes, and if the name is not fully qualified, then the default domain name is used. The pattern <code>schema.queue@dblink</code> is used.</td>
</tr>
<tr>
<td>start_time</td>
<td>Initial start time for the propagation window for messages from the source queue to the destination.</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.</td>
</tr>
<tr>
<td>next_time</td>
<td>Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, <code>next_time</code> should be specified as <code>SYSDATE + 1 - duration/86400</code>.</td>
</tr>
<tr>
<td>latency</td>
<td>Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. For example, if the latency is 60 seconds and there are no messages to be propagated during the propagation window, then messages from that queue for the destination are not propagated for at least 60 more seconds. It is at least 60 seconds before the queue is checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue is not checked for 10 minutes, and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination. As soon as a message is enqueued, it is propagated.</td>
</tr>
<tr>
<td>destination_queue</td>
<td>Name of the target queue to which messages are to be propagated in the form of a <code>dblink</code></td>
</tr>
</tbody>
</table>

### Usage Notes

Messages may also be propagated to other queues in the same database by specifying a NULL destination. If a message has multiple recipients at the same destination in either the same or different queues, the message is propagated to all of them at the same time.

Oracle extensions for JMS such as JMS propagation and remote subscribers are not currently supported for sharded queues. Propagation between sharded and non-sharded queues is not supported.
Related Topics

- Oracle Database Advanced Queuing User’s Guide

22.4.38 SET_WATERMARK Procedure

This procedure is used for Oracle Database Advanced Queuing notification to specify and limit memory use.

Syntax

```sql
DBMS_AQADM.SET_WATERMARK (wmvalue IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmvalue</td>
<td>Watermark value in megabytes.</td>
</tr>
</tbody>
</table>

22.4.39 SET_MAX_STREAMS_POOL Procedure

This procedure is used for Oracle Database Advanced Queuing to specify and limit maximum streams pool memory use.

Syntax

```sql
DBMS_AQADM.SET_MAX_STREAMS_POOL (value IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value in megabytes.</td>
</tr>
</tbody>
</table>

22.4.40 SET_MIN_STREAMS_POOL Procedure

This procedure is used for Oracle Database AQ to specify and limit minimum streams pool memory use.

Syntax

```sql
DBMS_AQADM.SET_MIN_STREAMS_POOL (value IN NUMBER);
```
Table 22-50  SET_MIN_STREAMS_POOL Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value in megabytes.</td>
</tr>
</tbody>
</table>

22.4.41 SET_QUEUE_PARAMETER Procedure

This procedure allows users to set different parameters for sharded queues at queue or database level. For database level the queue_name should be NULL. Note that queue overrides database level parameter values.

See Also:

Oracle® Database Advanced Queuing User's Guide for information about sharded queues

Syntax

```sql
PROCEDURE  SET_QUEUE_PARAMETER(
    queue_name          IN VARCHAR2,
    param_name          IN VARCHAR2,
    param_value         IN NUMBER);
```

Parameters

Table 22-51  SET_QUEUE_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the sharded queue.</td>
</tr>
<tr>
<td>param_name</td>
<td>The name of the parameter.</td>
</tr>
<tr>
<td>param_value</td>
<td>The value of the parameter.</td>
</tr>
</tbody>
</table>

22.4.42 START_QUEUE Procedure

This procedure enables the specified queue for enqueuing or dequeuing.

Syntax

```sql
DBMS_AQADM.START_QUEUE (    queue_name      IN     VARCHAR2,
    enqueue         IN     BOOLEAN DEFAULT TRUE,
    dequeue         IN     BOOLEAN DEFAULT TRUE);
```
Parameters

Table 22-52  START_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue to be enabled</td>
</tr>
<tr>
<td>enqueue</td>
<td>Specifies whether ENQUEUE should be enabled on this queue. TRUE means enable ENQUEUE. This is the default. FALSE means do not alter the current setting.</td>
</tr>
<tr>
<td>dequeue</td>
<td>Specifies whether DEQUEUE should be enabled on this queue. TRUE means enable DEQUEUE. This is the default. FALSE means do not alter the current setting.</td>
</tr>
</tbody>
</table>

Usage Notes

After creating a queue, the administrator must use START_QUEUE to enable the queue. The default is to enable it for both ENQUEUE and DEQUEUE. Only dequeue operations are allowed on an exception queue. This operation takes effect when the call completes and does not have any transactional characteristics.

22.4.43 STOP_QUEUE Procedure

This procedure disables enqueuing or dequeuing on the specified queue.

Syntax

```
DBMS_AQADM.STOP_QUEUE (|
    queue_name      IN   VARCHAR2,
    enqueue         IN   BOOLEAN DEFAULT TRUE,
    dequeue         IN   BOOLEAN DEFAULT TRUE,
    wait            IN   BOOLEAN DEFAULT TRUE,
    free_memory     IN   BOOLEAN DEFAULT FALSE|
);
```

Parameters

Table 22-53  STOP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue to be disabled</td>
</tr>
<tr>
<td>enqueue</td>
<td>Specifies whether ENQUEUE should be disabled on this queue. TRUE means disable ENQUEUE. This is the default. FALSE means do not alter the current setting.</td>
</tr>
<tr>
<td>dequeue</td>
<td>Specifies whether DEQUEUE should be disabled on this queue. TRUE means disable DEQUEUE. This is the default. FALSE means do not alter the current setting.</td>
</tr>
<tr>
<td>wait</td>
<td>Specifies whether to wait for the completion of outstanding transactions. TRUE means wait if there are any outstanding transactions. In this state no new transactions are allowed to enqueue to or dequeue from this queue. FALSE means return immediately either with a success or an error.</td>
</tr>
<tr>
<td>free_memory</td>
<td>Specifies whether the queue should be stopped.</td>
</tr>
</tbody>
</table>
Usage Notes

By default, this call disables both \texttt{ENQUEUE} and \texttt{DEQUEUE}. A queue cannot be stopped if there are outstanding transactions against the queue. This operation takes effect when the call completes and does not have any transactional characteristics.

22.4.44 UNSCHEDULE_PROPAGATION Procedure

This procedure unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link.

Syntax

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION (
  queue_name IN VARCHAR2,
  destination IN VARCHAR2 DEFAULT NULL,
  destination_queue IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.</td>
</tr>
<tr>
<td>destination</td>
<td>Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.</td>
</tr>
<tr>
<td>destination_queue</td>
<td>Name of the target queue to which messages are to be propagated in the form of a \texttt{dblink}</td>
</tr>
</tbody>
</table>

22.4.45 UNSET_QUEUE_PARAMETER Procedure

This procedure allows user to unset different parameters for sharded queues at queue or database level.

For database level the \texttt{queue_name} should be \texttt{NULL}. Note that queue overrides database level parameter values.

\textbf{See Also:}

Oracle® Database Advanced Queuing User’s Guide for information about sharded queues
Syntax

PROCEDURE UNSET_QUEUE_PARAMETER(
    queue_name           IN VARCHAR2,
    param_name           IN VARCHAR2);

Parameters

Table 22-55  UNSET_QUEUE_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the sharded queue.</td>
</tr>
<tr>
<td>param_name</td>
<td>The name of the parameter.</td>
</tr>
</tbody>
</table>

22.4.46 VERIFY_QUEUE_TYPES Procedure

This procedure verifies that the source and destination queues have identical types.

The result of the verification is stored in the table `sys.aq$_message_types`, overwriting all previous output of this command.

Syntax

DBMS_AQADM.VERIFY_QUEUE_TYPES (
    src_queue_name    IN    VARCHAR2,
    dest_queue_name   IN    VARCHAR2,
    destination       IN    VARCHAR2 DEFAULT NULL,
    rc                OUT   BINARY_INTEGER);

Parameters

Table 22-56  VERIFY_QUEUE_TYPES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_queue_name</td>
<td>Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.</td>
</tr>
<tr>
<td>dest_queue_name</td>
<td>Name of the destination queue where messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.</td>
</tr>
<tr>
<td>destination</td>
<td>Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.</td>
</tr>
<tr>
<td>rc</td>
<td>Return code for the result of the procedure. If there is no error, and if the source and destination queue types match, then the result is 1. If they do not match, then the result is 0. If an Oracle error is encountered, then it is returned in rc.</td>
</tr>
</tbody>
</table>
Note:

- **SYS.AQ$_MESSAGE_TYPES** can have multiple entries for the same source queue, destination queue, and database link, but with different transformations.
- **VERIFY_QUEUE_TYPES** check happens once per AQ propagation schedule and not for every propagated message send.
- In case the payload of the queue is modified then the existing propagation schedule between source and destination queue needs to be dropped and recreated.
The package provides subprograms to manage the configuration of Oracle Streams Advanced Queuing (AQ) asynchronous notification by e-mail and HTTP.

See Also:

Oracle Database Advanced Queuing User's Guide for detailed information about DBMS_AQELM

This chapter contains the following topics:

- Security Model
- Summary of DBMS_AQELM Subprograms

23.1 DBMS_AQELM Security Model

You need the administrator role AQ_ADMINISTRATOR_ROLE to run all procedures in DBMS_AQELM.

23.2 Summary of DBMS_AQELM Subprograms

This section lists and briefly describes the DBMS_AQELM subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_MAILHOST Procedure</td>
<td>Sets the host name for the SMTP server that the database will use to send</td>
</tr>
<tr>
<td></td>
<td>notifications</td>
</tr>
<tr>
<td>SET_MAILPORT Procedure</td>
<td>Sets the port number for the SMTP server</td>
</tr>
<tr>
<td>SET_SENDFROM Procedure</td>
<td>Sets the sent-from e-mail address</td>
</tr>
</tbody>
</table>

23.2.1 SET_MAILHOST Procedure

This procedure sets the host name for the SMTP server. The database uses this SMTP server host name to send out e-mail notifications.

Syntax

DBMS_AQELM.SET_MAILHOST (|
    mailhost    IN VARCHAR2);
Parameters

Table 23-2 SET_MAILHOST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailhost</td>
<td>SMTP server host name.</td>
</tr>
</tbody>
</table>

Usage Notes

As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on the DBMS_AQELM package needs to set the host name before registering for e-mail notifications.

23.2.2 SET_MAILPORT Procedure

This procedure sets the port number for the SMTP server.

Syntax

```sql
DBMS_AQELM.SET_MAILPORT (mailport  IN NUMBER);
```

Parameters

Table 23-3 SET_MAILPORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailport</td>
<td>SMTP server port number.</td>
</tr>
</tbody>
</table>

Usage Notes

As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on DBMS_AQELM package needs to set the port number before registering for e-mail notifications. The database uses this SMTP server port number to send out e-mail notifications. If not set, the SMTP mailport defaults to 25.

23.2.3 SET_SENDFROM Procedure

This procedure sets the sent-from e-mail address. This e-mail address is used in the sent-from field in all the e-mail notifications sent out by the database to the registered e-mail addresses.

Syntax

```sql
DBMS_AQELM.SET_SENDFROM (sendfrom IN VARCHAR2);
```
Parameters

Table 23-4  SET_SENDFROM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendfrom</td>
<td>The sent-from e-mail address.</td>
</tr>
</tbody>
</table>

Usage Notes

As part of the configuration for e-mail notifications, a user with AQ_ADMINISTRATOR_ROLE or with EXECUTE permissions on the DBMS_AQELM package should set the sent-from address before registering for e-mail notifications.
DBMS_AQIN

The DBMS_AQIN package plays a part in providing secure access to the Oracle JMS interfaces.

See Also:

Oracle Database Advanced Queuing User's Guide for detailed information about DBMS_AQIN

This chapter contains the following topic:

- Security Model

24.1 DBMS_AQIN Security Model

While you should not call any subprograms in the DBMS_AQIN package directly, you must have the EXECUTE privilege on the DBMS_AQIN and DBMS_AQJMS packages to use the Oracle JMS interfaces.

Use the following syntax to accomplish this with regard to the DBMS_AQIN package:

```
GRANT EXECUTE ON DBMS_AQIN to user;
```

You must have EXECUTE privilege on the DBMS_AQIN and DBMS_AQJMS packages to use the Oracle JMS interfaces. You can also acquire these rights through the AQ_USER_ROLE or the AQ_ADMINSTRATOR_ROLE.

See Also:

Oracle Database Advanced Queuing User's Guide for more information on accessing standard and Oracle JMS applications.
The DBMS_ASSERT package provides an interface to validate properties of the input value.

**See Also:**

*Oracle Database PL/SQL Language Reference* for more information about "Avoiding SQL Injection in PL/SQL"

This chapter contains the following topics:

- Operational Notes
- Summary of DBMS_ASSERT Subprograms

### 25.1 DBMS_ASSERT Operational Notes

If the condition which determines the property asserted in a function is not met then a value error is raised. Otherwise the input value is returned through the return value. Most functions return the value unchanged, however, several functions modify the value.

### 25.2 Summary of DBMS_ASSERT Subprograms

This section describes the subprograms of the DBMS_ASSERT package.

**Table 25-1  DBMS_ASSERT Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENQUOTE_LITERAL Function</td>
<td>Enquotes a string literal</td>
</tr>
<tr>
<td>ENQUOTE_NAME Function</td>
<td>Ensures that a string is enclosed by quotation marks, then checks that the result is a valid SQL identifier.</td>
</tr>
<tr>
<td>NOOP Functions</td>
<td>Returns the value without any checking</td>
</tr>
<tr>
<td>QUALIFIED_SQL_NAME Function</td>
<td>Verifies that the input string is a qualified SQL name</td>
</tr>
<tr>
<td>SCHEMA_NAME Function</td>
<td>Verifies that the input string is an existing schema name</td>
</tr>
<tr>
<td>SIMPLE_SQL_NAME Function</td>
<td>Verifies that the input string is a simple SQL name</td>
</tr>
<tr>
<td>SQL_OBJECT_NAME Function</td>
<td>Verifies that the input parameter string is a qualified SQL identifier of an existing SQL object</td>
</tr>
</tbody>
</table>
25.2.1 ENQUOTE_LITERAL Function

This function adds leading and trailing single quotes to a string literal.

Syntax

```sql
DBMS_ASSERT.ENQUOTE_LITERAL (
    str      VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table 25-2  ENQUOTE_LITERAL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>String to enquote</td>
</tr>
</tbody>
</table>

Usage Notes

- Verify that all single quotes except leading and trailing characters are paired with adjacent single quotes.
- No additional quotes are added if the name was already in quotes.

25.2.2 ENQUOTE_NAME Function

This function encloses the provided string in double quotes (quotation marks). No additional quotes are added if the string was already in quotes (quotation marks). The quoted string is then checked to see if it is a valid (quoted) simple SQL name.

For more information on Database object names and qualifiers, see Oracle Database SQL Language Reference.

Syntax

```sql
DBMS_ASSERT.ENQUOTE_NAME (
    str      VARCHAR2,
    capitalize BOOLEAN DEFAULT TRUE)
RETURN VARCHAR2;
```

Parameters

Table 25-3  ENQUOTE_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>String to enquote</td>
</tr>
<tr>
<td>capitalize</td>
<td>If TRUE or defaulted, alphabetic characters of str which was not in quotes are translated to upper case</td>
</tr>
</tbody>
</table>

Usage Notes

- No additional quotes are added if the name was already in quotes.
- Verify that all other double quotes in the string are adjacent pairs of double quotes.
Examples

-- This procedure creates a single column table in the createOneColumnTable's schema.
create or replace procedure createOneColumnTable(proposedTableName varchar2) is
BEGIN
  IF
    (proposedTableName is NULL)
  THEN
    raise value_error;
  END IF;
  -- The use of ENQUOTE_NAME ensures that the table will be created in the
  -- definer's schema and not in some other schema even if the definer has
  -- privileges to create tables in other schemas.
  execute immediate 'create table ' ||
    DBMS_ASSERT.ENQUOTE_NAME(proposedTableName) || ' (c1 number)';
exception
  when
    others
  then
    dbms_output.put_line('Table creation failed due to: ' || SQLERRM);
  end;
/

-- Examples of ENQUOTE_NAME showing input/output relationships
begin
  -- 'eMp' becomes "EMP" since it is unquoted
  dbms_output.put_line(DBMS_ASSERT.ENQUOTE_NAME('eMp'));
end;
/

begin
  -- For quoted strings, the case is preserved
  dbms_output.put_line(DBMS_ASSERT.ENQUOTE_NAME('"EmP"'));
end;
/

-- Invalid identifier example
begin
  dbms_output.put_line(DBMS_ASSERT.ENQUOTE_NAME('SCOTT."EMP"'));
end;
/

-- CHR(0) examples
-- The following examples illustrates that CHR(0), the NULL character, cannot appear
-- in the string; such a string poses a SQL injection risk.
begin
  dbms_output.put_line(DBMS_ASSERT.ENQUOTE_NAME('BAD' || CHR(0) || 'IDENTIFIER'));
end;
/

begin
  dbms_output.put_line(DBMS_ASSERT.ENQUOTE_NAME('"SCOTT" || CHR(0) || '.EMP"'));
end;
/

-- Oracle allows a period (.) to be a part of a quoted string
begin
  dbms_output.put_line(DBMS_ASSERT.ENQUOTE_NAME('"SCOTT.EMP"'));
end;
-- The single quotation mark ('), as opposed to a double quotation mark, can
appear in the string
-- Note: In Oracle, a single quotation mark is specified in a literal using two
single quotes. The first quotation mark escapes the second quotation mark in the
same way that
-- backslash (\) in POSIX is an escape character.
BEGIN
    dbms_output.put_line(DDBMS_ASSERT.ENQUOTE_NAME('"O''LEARY"'));
END;
/

25.2.3 NOOP Functions

This function returns the value without any checking.

Syntax

DBMS_ASSERT.NOOP (str VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2 CHARACTER SET str%CHARSET;

DBMS_ASSERT.NOOP (str CLOB CHARACTER SET ANY_CS)
RETURN CLOB CHARACTER SET str%CHARSET;

Parameters

Table 25-4 NOOP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Input value</td>
</tr>
</tbody>
</table>

25.2.4 QUALIFIED_SQL_NAME Function

This function verifies that the input string is a qualified SQL name.

Syntax

DBMS_ASSERT.QUALIFIED_SQL_NAME (str VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2 CHARACTER SET str%CHARSET;

Parameters

Table 25-5 QUALIFIED_SQL_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Input value</td>
</tr>
</tbody>
</table>

Exceptions

ORA44004: string is not a qualified SQL name
Usage Notes

A qualified SQL name <qualified name> can be expressed by the following grammar:

<local qualified name> ::= <simple name> {'.' <simple name>}
<database link name> ::= <local qualified name> ['@' <connection string>]
<connection string> ::= <simple name>
<qualified name> ::= <local qualified name> ['@' <database link name>]

25.2.5 SCHEMA_NAME Function

This function verifies that the input string is an existing schema name.

Syntax

```
DBMS_ASSERT.SCHEMA_NAME (str      VARCHAR2 CHARACTER SET ANY_CS)
RETURN     VARCHAR2 CHARACTER SET str%CHARSET;
```

Parameters

Table 25-6  SCHEMA_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Input value</td>
</tr>
</tbody>
</table>

Exceptions

ORA44001: Invalid schema name

Usage Notes

By definition, a schema name need not be just a simple SQL name. For example, "FIRST LAST" is a valid schema name. As a consequence, care must be taken to quote the output of schema name before concatenating it with SQL text.

25.2.6 SIMPLE_SQL_NAME Function

This function verifies that the input string is a simple SQL name.

Syntax

```
DBMS_ASSERT.SIMPLE_SQL_NAME (str      VARCHAR2 CHARACTER SET ANY_CS)
RETURN     VARCHAR2 CHARACTER SET str%CHARSET;
```

Parameters

Table 25-7  SIMPLE_SQL_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Input value</td>
</tr>
</tbody>
</table>
Exceptions

ORA44003: string is not a simple SQL name

Usage Notes

• The input value must be meet the following conditions:
  – The name must begin with an alphabetic character. It may contain alphanumeric characters as well as the characters _, $, and # in the second and subsequent character positions.
  – Quoted SQL names are also allowed.
  – Quoted names must be enclosed in double quotes.
  – Quoted names allow any characters between the quotes.
  – Quotes inside the name are represented by two quote characters in a row, for example, "a name with "" inside" is a valid quoted name.
  – The input parameter may have any number of leading and/or trailing white space characters.
• The length of the name is not checked.

25.2.7 SQL_OBJECT_NAME Function

This function verifies that the input parameter string is a qualified SQL identifier of an existing SQL object.

Syntax

DBMS_ASSERT.SQL_OBJECT_NAME (str VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2 CHARACTER SET str%CHARSET;

Parameters

Table 25-8 SQL_OBJECT_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Input value</td>
</tr>
</tbody>
</table>

Exceptions

ORA44002: Invalid object name

Usage Notes

The use of synonyms requires that the base object exists.
26
DBMS_AUDIT_MGMT

The `DBMS_AUDIT_MGMT` package provides subprograms to manage audit trail records. These subprograms enable audit administrators to manage the audit trail. In a mixed-mode environment, these audit trails comprise the database, operating system (OS), and XML audit trails. In a unified auditing environment, this comprises the unified audit trail.

This chapter contains the following topics:

- DBMS_AUDIT_MGMT Overview
- DBMS_AUDIT_MGMT Deprecated Subprograms
- DBMS_AUDIT_MGMT Security Model
- DBMS_AUDIT_MGMT Constants
- DBMS_AUDIT_MGMT Views
- Subprogram Groups
- Summary of DBMS_AUDIT_MGMT Subprograms

See Also:

- Oracle Database Security Guide regarding verifying security access with auditing
- Oracle Database Upgrade Guide regarding migrating to unified auditing

26.1 DBMS_AUDIT_MGMT Overview

Database auditing helps meet your database security and compliance requirements. In a mixed mode environment, audit records are written to database tables, operating system (OS) files, or XML files depending on the AUDIT_TRAIL initialization parameter setting. If you have upgraded to unified auditing, then the audit records are written to the unified audit trail.

In a mixed mode environment, when `AUDIT_TRAIL` is set to `DB`, database records are written to the `AUD$` table. In a unified auditing environment, audit records are written to a read-only table in the `AUDSYS` schema. The contents of this table are available from the `UNIFIED_AUDIT_TRAIL` data dictionary view. When `AUDIT_TRAIL` is set to `OS`, audit records are written to operating system files. When `AUDIT_TRAIL` is set to `XML`, audit records are written to operating system files in XML format.

With Unified Auditing facility, all audit records are written to the unified audit trail in a uniform format and are made available through the `UNIFIED_AUDIT_TRAIL` views.

It is important to manage your audit records properly in order to ensure efficient performance and disk space management. The `DBMS_AUDIT_MGMT` subprograms enable you to efficiently manage your audit trail records.
If you have not yet migrated to unified auditing, the DBMS_AUDIT_MGMT package provides a subprogram that allows you to move the database audit trail tables out of the SYSTEM tablespace. This improves overall database performance by reducing the load on the SYSTEM tablespace. It also enables you to dedicate an optimized tablespace for audit records.

For a mixed mode environment, the DBMS_AUDIT_MGMT subprograms also enable you to manage your operating system and XML audit files. You can define properties like the maximum size and age of an audit file. This enables you to keep the file sizes of OS and XML audit files in check.

The DBMS_AUDIT_MGMT subprograms enable you to perform cleanup operations on all audit trail types. Audit trails can be cleaned based on the last archive timestamp value. The last archive timestamp represents the timestamp of the most recent audit record that was securely archived.

The DBMS_AUDIT_MGMT package provides a subprogram that enables audit administrators to set the last archive timestamp for archived audit records. This subprogram can also be used by external archival systems to set the last archive timestamp.

The DBMS_AUDIT_MGMT subprograms also enable you to configure jobs that periodically delete audit trail records. The frequency with which these jobs should run can be controlled by the audit administrator.

See Also:
Oracle Database Security Guide for a detailed description of unified auditing

26.2 DBMS_AUDIT_MGMT Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms in new applications. Support for deprecated features is for backward compatibility only.

The following have been deprecated from the DBMS_AUDIT_MGMT package starting in Oracle Database 12c release 12.2:

- DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL procedure
- AUDIT_TRAIL_WRITE mode of the AUDIT_TRAIL_PROPERTY parameter of the DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY procedure

These are no longer necessary because audit records now bypass the common logging infrastructure queues and are directly written to a new internal relational table.

26.3 DBMS_AUDIT_MGMT Security Model

All DBMS_AUDIT_MGMT subprograms require the user to have EXECUTE privilege over the DBMS_AUDIT_MGMT package. The SYSDBA and AUDIT_ADMIN roles have EXECUTE privileges on the package by default.

Oracle strongly recommends that only audit administrators should have the EXECUTE privilege on the DBMS_AUDIT_MGMT package and be granted the AUDIT_ADMIN role.
Executions of the DBMS_AUDIT_MGMT subprograms are always audited.

26.4 DBMS_AUDIT_MGMT Constants

The DBMS_AUDIT_MGMT package defines several constants that can be used for specifying parameter values.

These constants shown in the following tables:

- Table 26-1
- Table 26-2
- Table 26-3

Audit trails can be classified based on whether audit records are written to database tables, operating system files, or XML files. The following table lists the audit trail type constants.

### Table 26-1  DBMS_AUDIT_MGMT Constants - Audit Trail Types

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT_TRAIL_ALL</td>
<td>PLS_INTEGER</td>
<td>All audit trail types. This includes the standard database audit trail (SYS.AUD$) and unified audit trail tables, operating system (OS) audit trail, and XML audit trail.</td>
</tr>
<tr>
<td>AUDIT_TRAIL_AUD_STD</td>
<td>PLS_INTEGER</td>
<td>Standard database audit records in the SYS.AUD$ table</td>
</tr>
<tr>
<td>AUDIT_TRAIL_DB_STD</td>
<td>PLS_INTEGER</td>
<td>Both standard audit (SYS.AUD$) and FGA audit (SYS.FGA_LOG$) records</td>
</tr>
<tr>
<td>AUDIT_TRAIL_FGA_STD</td>
<td>PLS_INTEGER</td>
<td>Standard database fine-grained auditing (FGA) records in the SYS.FGA_LOG$ table</td>
</tr>
<tr>
<td>AUDIT_TRAIL_FILES</td>
<td>PLS_INTEGER</td>
<td>Both operating system (OS) and XML audit trails</td>
</tr>
<tr>
<td>AUDIT_TRAIL_OS</td>
<td>PLS_INTEGER</td>
<td>Operating system audit trail. This refers to the audit records stored in operating system files.</td>
</tr>
<tr>
<td>AUDIT_TRAIL_UNIFIED</td>
<td>PLS_INTEGER</td>
<td>Unified audit trail. In unified auditing, all audit records are written to the unified audit trail and are made available through the unified audit trail views, such as UNIFIED_AUDIT_TRAIL.</td>
</tr>
<tr>
<td>AUDIT_TRAIL_XML</td>
<td>PLS_INTEGER</td>
<td>XML audit trail. This refers to the audit records stored in XML files.</td>
</tr>
</tbody>
</table>

Audit trail properties determine the audit configuration settings. The following table lists the constants related to audit trail properties.
Table 26-2  DBMS_AUDIT_MGMT Constants - Audit Trail Properties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT_TRAIL_WRITE_MODE</td>
<td>PLS_INTEGER</td>
<td>Note: This procedure has been deprecated starting in Oracle Database 12c release 12.2. A value of AUDIT_TRAIL_IMMEDIATE_WRITE indicates that the audit record must be immediately persisted and not to be queued. By contrast, AUDIT_TRAIL_QUEUED_WRITE indicates that the audit record can be queued and persisting can be done according to the database's flushing strategy. See Also Oracle Database Security Guide</td>
</tr>
<tr>
<td>CLEAN_UP_INTERVAL</td>
<td>PLS_INTEGER</td>
<td>Interval, in hours, after which the cleanup procedure is called to clear audit records in the specified audit trail.</td>
</tr>
<tr>
<td>DB_DELETE_BATCH_SIZE</td>
<td>PLS_INTEGER</td>
<td>Specifies the batch size to be used for deleting audit records in database audit tables. The audit records are deleted in batches of size equal to DB_DELETE_BATCH_SIZE.</td>
</tr>
<tr>
<td>FILE_DELETE_BATCH_SIZE</td>
<td>PLS_INTEGER</td>
<td>Specifies the batch size to be used for deleting audit files in the audit directory. The audit files are deleted in batches of size equal to FILE_DELETE_BATCH_SIZE.</td>
</tr>
<tr>
<td>OS_FILE_MAX_AGE</td>
<td>PLS_INTEGER</td>
<td>Specifies the maximum number of days for which an operating system (OS) or XML audit file can be kept open before a new audit file gets created.</td>
</tr>
<tr>
<td>OS_FILE_MAX_SIZE</td>
<td>PLS_INTEGER</td>
<td>Specifies the maximum size, in kilobytes (KB), to which an operating system (OS) or XML audit file can grow before a new file is opened.</td>
</tr>
</tbody>
</table>

The audit trail purge job cleans the audit trail. The following table lists the constants related to purge job status values.

Table 26-3  DBMS_AUDIT_MGMT Constants - Purge Job Status

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PURGE_JOB_DISABLE</td>
<td>PLS_INTEGER</td>
<td>Disables a purge job</td>
</tr>
<tr>
<td>PURGE_JOB_ENABLE</td>
<td>PLS_INTEGER</td>
<td>Enables a purge job</td>
</tr>
</tbody>
</table>

26.5 DBMS_AUDIT_MGMT Views

DBMS_AUDIT_MGMT views are used to display DBMS_AUDIT_MGMT configuration and cleanup events.

These views listed in the following table.
Table 26-4  Views used by DBMS_AUDIT_MGMT

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_AUDIT_MGMT_CLEAN_EVENTS</td>
<td>Displays the cleanup event history</td>
</tr>
<tr>
<td>DBA_AUDIT_MGMT_CLEANUP_JOBS</td>
<td>Displays the currently configured audit trail purge jobs</td>
</tr>
<tr>
<td>DBA_AUDIT_MGMT_CONFIG_PARAMS</td>
<td>Displays the currently configured audit trail properties</td>
</tr>
<tr>
<td>DBA_AUDIT_MGMT_LAST_ARCH TS</td>
<td>Displays the last archive timestamps set for the audit trails</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database Reference for more information on these views

26.6 DBMS_AUDIT_MGMT Subprogram Groups

The DBMS_AUDIT_MGMT package subprograms can be grouped into two categories: Audit Trail Management Subprograms and Audit Trail Cleanup Subprograms.

- Audit Trail Management Subprograms
- Audit Trail Cleanup Subprograms

26.6.1 DBMS_AUDIT_MGMT Audit Trail Management Subprograms

Audit trail management subprograms enable you to manage audit trail properties.

Table 26-5  Audit Trail Management Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_PARTITION_INTERVAL Procedure</td>
<td>Changes the unified audit internal relational table’s partition interval</td>
</tr>
<tr>
<td>CLEAR_AUDIT_TRAILPROPERTY Procedure</td>
<td>Clears the value for the audit trail property that you specify</td>
</tr>
<tr>
<td>FLUSH_UNIFIED_AUDIT_TRAIL Procedure</td>
<td>Writes the unified audit trail records in the SGA queue to disk (deprecated starting with Oracle Database 12c release 12.2)</td>
</tr>
<tr>
<td>GET_AUDIT_TRAILPROPERTY_VALUE Function</td>
<td>Returns the property value set by the SET_AUDIT_TRAIL_PROPERTY Procedure</td>
</tr>
<tr>
<td>GET_LAST_ARCHIVE_TIMESTAMP Function</td>
<td>Returns the timestamp set by the SET_LAST_ARCHIVE_TIMESTAMP Procedure in that database instance</td>
</tr>
<tr>
<td>LOAD_UNIFIED_AUDIT_FILES Procedure</td>
<td>Loads the data from the spillover OS audit files in a unified audit trail into the designated unified audit trail tablespace</td>
</tr>
<tr>
<td>SET_AUDIT_TRAIL_LOCATION Procedure</td>
<td>Moves the audit trail tables from their current tablespace to a user-specified tablespace</td>
</tr>
</tbody>
</table>
Table 26-5  (Cont.) Audit Trail Management Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_AUDIT_TRAIL_PROPERTY Procedure</td>
<td>Sets an audit trail property for the audit trail type that you specify</td>
</tr>
<tr>
<td>SET_LAST_ARCHIVE_TIMESTAMP Procedure</td>
<td>Sets a timestamp indicating when the audit records or files were last archived</td>
</tr>
<tr>
<td>TRANSFER_UNIFIED_AUDIT_RECORDS Procedure</td>
<td>Transfers audit records from the common logging infrastructure (CLI) swap table to the AUDSYS.AUD$UNIFIED relational table</td>
</tr>
</tbody>
</table>

The Summary of DBMS_AUDIT_MGMT Subprograms contains a complete listing of all subprograms in the package.

26.6.2 DBMS_AUDIT_MGMT Audit Trail Cleanup Subprograms

Audit trail cleanup subprograms help you perform cleanup related operations on the audit trail records.

Table 26-6  Audit Trail Cleanup Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAN_AUDIT_TRAIL Procedure</td>
<td>Deletes audit trail records or files that have been archived</td>
</tr>
<tr>
<td>CLEAR_LAST_ARCHIVE_TIMESTAMP Procedure</td>
<td>Clears the timestamp set by the SET_LAST_ARCHIVE_TIMESTAMP Procedure</td>
</tr>
<tr>
<td>CREATE_PURGE_JOB Procedure</td>
<td>Creates a purge job for periodically deleting the audit trail records or files</td>
</tr>
<tr>
<td>DEINIT_CLEANUP Procedure</td>
<td>Undoes the setup and initialization performed by the INIT_CLEANUP Procedure</td>
</tr>
<tr>
<td>DROP_OLD_UNIFIED_AUDIT_TABLES Procedure</td>
<td>Drops old unified audit tables following the cloning of a pluggable database (PDB)</td>
</tr>
<tr>
<td>DROP_PURGE_JOB Procedure</td>
<td>Drops the purge job created using the CREATE_PURGE_JOB Procedure</td>
</tr>
<tr>
<td>INIT_CLEANUP Procedure</td>
<td>Sets up the audit management infrastructure and sets a default cleanup interval for audit trail records or files</td>
</tr>
<tr>
<td>IS_CLEANUP_INITIALIZED Function</td>
<td>Checks to see if the INIT_CLEANUP Procedure has been run for an audit trail type</td>
</tr>
<tr>
<td>SET_PURGE_JOB_INTERVAL Procedure</td>
<td>Sets the interval at which the CLEAN_AUDIT_TRAIL Procedure is called for the purge job that you specify</td>
</tr>
<tr>
<td>SET_PURGE_JOB_STATUS Procedure</td>
<td>Enables or disables the purge job that you specify</td>
</tr>
</tbody>
</table>

The Summary of DBMS_AUDIT_MGMT Subprograms contains a complete listing of all subprograms in the package.
26.7 Summary of DBMS_AUDIT_MGMT Subprograms

This table lists and describes the subprograms of the `DBMS_AUDIT_MGMT` package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ALTER_PARTITION_INTERVAL</code> Procedure</td>
<td>Changes the unified audit internal relational table’s partition interval</td>
</tr>
<tr>
<td><code>CLEAN_AUDIT_TRAIL</code> Procedure</td>
<td>Deletes audit trail records that have been archived</td>
</tr>
<tr>
<td><code>CLEAR_AUDIT_TRAILPROPERTY</code> Procedure</td>
<td>Clears the value for the audit trail property that you specify</td>
</tr>
<tr>
<td><code>CLEAR_LAST_ARCHIVE_TIMESTAMP</code> Procedure</td>
<td>Clears the timestamp set by the set by the <code>SET_LAST_ARCHIVE_TIMESTAMP</code> Procedure</td>
</tr>
<tr>
<td><code>CREATE_PURGE_JOB</code> Procedure</td>
<td>Creates a purge job for periodically deleting the audit trail records</td>
</tr>
<tr>
<td><code>DEINIT_CLEANUP</code> Procedure</td>
<td>Undoes the setup and initialization performed by the <code>INIT_CLEANUP</code> Procedure</td>
</tr>
<tr>
<td><code>DROP_OLD_UNIFIED_AUDIT_TABLES</code> Procedure</td>
<td>Drops old unified audit tables following the cloning of a pluggable database (PDB)</td>
</tr>
<tr>
<td><code>DROP_PURGE_JOB</code> Procedure</td>
<td>Drops the purge job created using the <code>CREATE_PURGE_JOB</code> Procedure</td>
</tr>
<tr>
<td><code>FLUSH_UNIFIED_AUDIT_TRAIL</code> Procedure</td>
<td>Writes the unified audit trail records in the SGA queue to disk</td>
</tr>
<tr>
<td><code>GET_AUDIT_COMMIT_DELAY</code> Function</td>
<td>Returns the audit commit delay time as the number of seconds. This is the maximum time that it takes to COMMIT an audit record to the database audit trail.</td>
</tr>
<tr>
<td><code>GET_AU__TIT_TRAIL_PROPERTY_VALUE</code> Function</td>
<td>Returns the property value set by the <code>SET_AUDIT_TRAIL_PROPERTY</code> Procedure</td>
</tr>
<tr>
<td><code>GET_LAST_ARCHIVE_TIMESTAMP</code> Function</td>
<td>Returns the timestamp set by the <code>SET_LAST_ARCHIVE_TIMESTAMP</code> Procedure in that database instance</td>
</tr>
<tr>
<td><code>INIT_CLEANUP</code> Procedure</td>
<td>Sets up the audit management infrastructure and sets a default cleanup interval for audit trail records</td>
</tr>
<tr>
<td><code>IS_CLEANUP_INITIALIZED</code> Function</td>
<td>Checks to see if the <code>INIT_CLEANUP</code> Procedure has been run for an audit trail type</td>
</tr>
<tr>
<td><code>LOAD_UNIFIED_AUDIT_FILES</code> Procedure</td>
<td>Loads the data from the spillover OS audit files in a unified audit trail into the designated unified audit trail tablespace</td>
</tr>
<tr>
<td><code>SET_AUDIT_TRAIL_LOCATION</code> Procedure</td>
<td>Moves the audit trail tables from their current tablespace to a user-specified tablespace</td>
</tr>
<tr>
<td><code>SET_AUDIT_TRAIL_PROPERTY</code> Procedure</td>
<td>Sets the audit trail properties for the audit trail type that you specify</td>
</tr>
<tr>
<td><code>SET_LAST_ARCHIVE_TIMESTAMP</code> Procedure</td>
<td>Sets a timestamp indicating when the audit records were last archived</td>
</tr>
<tr>
<td><code>SET_PURGE_JOB_INTERVAL</code> Procedure</td>
<td>Sets the interval at which the <code>CLEAN_AUDIT_TRAIL</code> Procedure is called for the purge job that you specify</td>
</tr>
<tr>
<td><code>SET_PURGE_JOB_STATUS</code> Procedure</td>
<td>Enables or disables the purge job that you specify</td>
</tr>
</tbody>
</table>
Table 26-7  (Cont.) DBMS_AUDIT_MGMT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSFER_UNIFIED_AUDIT_RECORD S Procedure</td>
<td>Transfers audit records from the common logging infrastructure (CLI) swap table to the AUDSYS.AUD$UNIFIED relational table</td>
</tr>
</tbody>
</table>

26.7.1 ALTER_PARTITION_INTERVAL Procedure

This procedure changes the unified audit internal relational table’s partition interval.

Syntax

```sql
DBMS_AUDIT_MGMT.ALTER_PARTITION_INTERVAL(
    interval_number          IN BINARY_INTEGER,
    interval_frequency       IN VARCHAR2);
```

Parameters

Table 26-8  ALTER_PARTITION_INTERVAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval_number</td>
<td>Sets how often the database creates partitions for the unified audit internal relational table. For example, to specify that the partition is created every two days, you must enter 2.</td>
</tr>
<tr>
<td>interval_frequency</td>
<td>Sets the frequency for the value that was set in the interval_number setting. For example, for a partition to be created every two days, with interval_number set to 2, you must set interval_frequency to DAY. Supported values are YEAR, MONTH, and DAY.</td>
</tr>
</tbody>
</table>

Usage Notes

- The interval frequency that you choose depends on the rate of audit records that are generated in your database. The default setting is for one month. If you have a high audit record rate and are using the default, then too many audit records may be generated in the same partition. In this case, you should change the interval frequency to a more frequent interval, such as one month or one day. If the audit record rate generation is not so high, then you may want to keep it at the default of one month.

Example

The following example sets the partition interval to occur every two months.

```sql
BEGIN
    DBMS_AUDIT_MGMT.ALTER_PARTITION_INTERVAL(
        interval_number       => 2,
        interval_frequency    => 'MONTH');
END;
```
26.7.2 CLEAN_AUDIT_TRAIL Procedure

This procedure deletes audit trail records.

The CLEAN_AUDIT_TRAIL procedure is usually called after the SET_LAST_ARCHIVE_TIMESTAMP Procedure has been used to set the last archived timestamp for the audit records.

Syntax

```
DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
    audit_trail_type         IN PLS_INTEGER,
    use_last_arch_timestamp  IN BOOLEAN DEFAULT TRUE,
    container                IN PLS_INTEGER DEFAULT CONTAINER_CURRENT,
    database_id              IN NUMBER DEFAULT NULL,
    container_guid           IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26-9  CLEAN_AUDIT_TRAIL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the cleanup operation needs to be performed. Audit trail types are listed in Table 26-1.</td>
</tr>
<tr>
<td>use_last_arch_timestamp</td>
<td>Specifies whether the last archived timestamp should be used for deciding on the records that should be deleted.</td>
</tr>
<tr>
<td></td>
<td>A value of TRUE indicates that only audit records created before the last archive timestamp should be deleted.</td>
</tr>
<tr>
<td></td>
<td>A value of FALSE indicates that all audit records should be deleted.</td>
</tr>
<tr>
<td></td>
<td>The default value is TRUE. Oracle recommends using this value, as this helps guard against inadvertent deletion of records.</td>
</tr>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs). When CONTAINER is set to CONTAINER_ALL, this purges the audit trail in all the PDBs, otherwise it only purges from the connected PDB.</td>
</tr>
<tr>
<td>database_id</td>
<td>Database ID (DBID) of the audit records to cleanup</td>
</tr>
<tr>
<td>container_guid</td>
<td>Container GUID of the audit records to cleanup</td>
</tr>
</tbody>
</table>

Note: This parameter has been deprecated but is currently retained for backward compatibility.

Usage Notes

The following usage notes apply:

- When cleaning up operating system (OS) or XML audit files, only files in the current audit directory, specified by the AUDIT_FILE_DEST parameter, are cleaned up.
- For Windows platforms, no cleanup is performed when the audit_trail_type parameter is set to DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS. This is because operating system (OS) audit records on Windows are written to the Windows Event Viewer.
• For Unix platforms, no cleanup is performed for cases where the operating system (OS) audit records are written to the syslog. When the `audit_trail_type` parameter is set to `DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS`, it removes only the `.aud` files under the directory specified by the `AUDIT_FILE_DEST` initialization parameter.

See Also:

"AUDIT_SYSLOG_LEVEL" in the Oracle Database Reference

• When the `audit_trail_type` parameter is set to `DBMS_AUDIT_MGMT.AUDIT_TRAIL_XML`, this procedure only removes XML audit files (`*.xml`) from the current audit directory. Oracle database maintains a book-keeping file (`adx_${ORACLE_SID}.txt`) for the XML audit files. This file is not removed by the cleanup procedure.

• If the cleanup of the unified audit trail is performed when the `use_last_arch_timestamp` parameter is set to TRUE:

  – If you set the `database_id` value for the cleanup operation, then this value is used with the last archive timestamp while `CLEAN_AUDIT_TRAIL` runs. However, for the unified audit records that are present during the cleanup of spillover operating system audit files, the `database_id` value is ignored. Cleanup for operating system audit files is based on the last archive timestamp only. If you want to have the `database_id` value used for the cleanup operation of unified audit trail records that are present in the spillover operating system audit files, then load the contents of these files into database tables by using the `DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES` procedure before you run `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL`.

  – If you do not set the `database_id` value for the cleanup operation, then `CLEAN_AUDIT_TRAIL` uses the database ID of the current database container. This ID is used along with the last archive timestamp value while cleaning up the unified audit records that are present in the database tables. However, for unified audit records that are present in the spillover operating system audit records, this database ID is not used and cleanup of these records is based on the last archive timestamp value only. If you want to include the database ID value in the cleanup of unified audit records that are present in the spillover operating system audit files, then load the contents of these files by using the `DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES` procedure before you run `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL`.

• If the cleanup of the unified audit trail is performed when the `use_last_arch_timestamp` parameter is set to FALSE:

  – If you set the `database_id` value for the cleanup operation, then this value is used while `CLEAN_AUDIT_TRAIL` cleans up the unified audit trail records that are present in database tables. However, the `database_id` value is not used for the cleanup of unified audit trail records that are present in spillover operating system files. If want the `database_id` value to be used in the `CLEAN_AUDIT_TRAIL` operation of unified audit records that are present in spillover operating system audit files, then load the contents of these files to the database tables by using the `DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES` procedure before you run `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL`.  


If you do not set the `database_id` value when you invoke the `CLEAN_AUDIT_TRAIL` procedure, then Oracle Database purges all unified audit records irrespective of `database_id` values and irrespective of the location (that is, database tables or spillover operating system audit files) where the unified audit records reside.

**Examples**

The following example calls the `CLEAN_AUDIT_TRAIL` procedure to clean up the operating system (OS) audit trail records that were updated before the last archive timestamp.

```sql
BEGIN
    DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
        audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
        use_last_arch_timestamp => TRUE);
END;
```

## 26.7.3 CLEAR_AUDIT_TRAIL_PROPERTY Procedure

This procedure clears the value for the specified audit trail property.

Audit trail properties are set using the `SET_AUDIT_TRAIL_PROPERTY` Procedure.

The `CLEAR_AUDIT_TRAIL_PROPERTY` procedure can optionally reset the property value to its default value through the `use_default_values` parameter.

**Syntax**

```sql
DBMS_AUDIT_MGMT.CLEAR_AUDIT_TRAIL_PROPERTY(
    audit_trail_type        IN PLS_INTEGER,
    audit_trail_property    IN PLS_INTEGER,
    use_default_values      IN BOOLEAN DEFAULT FALSE) ;
```

**Parameters**

**Table 26-10  CLEAR_AUDIT_TRAIL_PROPERTY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>audit_trail_type</code></td>
<td>The audit trail type for which the property needs to be cleared. Audit trail types are listed in Table 26-1</td>
</tr>
<tr>
<td><code>audit_trail_property</code></td>
<td>The audit trail property whose value needs to be cleared. You cannot clear the value for the <code>CLEANUP_INTERVAL</code> property. Audit trail properties are listed in Table 26-2</td>
</tr>
<tr>
<td><code>use_default_values</code></td>
<td>Specifies whether the default value of the <code>audit_trail_property</code> should be used in place of the cleared value. A value of <code>TRUE</code> causes the default value of the parameter to be used. A value of <code>FALSE</code> causes the <code>audit_trail_property</code> to have no value. The default value for this parameter is <code>FALSE</code>.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The following usage notes apply:
• You can use this procedure to clear the value for an audit trail property that you do not wish to use. For example, if you do not want a restriction on the operating system audit file size, then you can use this procedure to reset the OS_FILE_MAX_SIZE property.

You can also use this procedure to reset an audit trail property to its default value. You need to set use_default_values to TRUE when invoking the procedure.

• The DB_DELETE_BATCH_SIZE property needs to be individually cleared for the AUDIT_TRAIL_AUD_STD and AUDIT_TRAIL_PGA_STD audit trail types. You cannot clear this property collectively using the AUDIT_TRAIL_DB_STD and AUDIT_TRAIL_ALL audit trail types.

• If you clear the value of the DB_DELETE_BATCH_SIZE property with use_default_value set to FALSE, the default value of DB_DELETE_BATCH_SIZE is still assumed. This is because audit records are always deleted in batches.

• The FILE_DELETE_BATCH_SIZE property needs to be individually cleared for the AUDIT_TRAIL_OS and AUDIT_TRAIL_XML audit trail types. You cannot clear this property collectively using the AUDIT_TRAIL_FILES and AUDIT_TRAIL_ALL audit trail types.

• If you clear the value of the FILE_DELETE_BATCH_SIZE property with use default value set to FALSE, the default value of FILE_DELETE_BATCH_SIZE is still assumed. This is because audit files are always deleted in batches.

• You cannot clear the value for the CLEANUP_INTERVAL property.

• You cannot clear the value for the AUDIT_TRAIL_WRITE_MODE property.

Examples

The following example calls the CLEAR_AUDIT_TRAIL_PROPERTY procedure to clear the value for the audit trail property, OS_FILE_MAX_SIZE. The procedure uses a value of FALSE for the USE_DEFAULT_VALUES parameter. This means that there will be no maximum size threshold for operating system (OS) audit files.

```
BEGIN
    DBMS_AUDIT_MGMT.CLEAR_AUDIT_TRAIL_PROPERTY(
        AUDIT_TRAIL_TYPE        =>  DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
        AUDIT_TRAIL_PROPERTY    =>  DBMS_AUDIT_MGMT.OS_FILE_MAX_SIZE,
        USE_DEFAULT_VALUES      =>  FALSE );
END;
```

26.7.4 CLEAR_LAST_ARCHIVE_TIMESTAMP Procedure

This procedure clears the timestamp set by the SET_LAST_ARCHIVE_TIMESTAMP Procedure.

Syntax

```
DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP(
    audit_trail_type IN PLS_INTEGER,
    rac_instance_number IN PLS_INTEGER DEFAULT NULL,
    container IN PLS_INTEGER DEFAULT CONTAINER_CURRENT,
    database_id IN NUMBER DEFAULT NULL,
    container_guid IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 26-1  CLEAR_LAST_ARCHIVE_TIMESTAMP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the timestamp needs to be cleared. Audit trail types are listed in Table 26-1.</td>
</tr>
<tr>
<td>rac_instance_number</td>
<td>The instance number for the Oracle Real Application Clusters (Oracle RAC) instance. The default value is NULL. The rac_instance_number is not relevant for single instance databases. You can find the instance number by issuing the SHOW PARAMETER INSTANCE_NUMBER command in SQL*Plus.</td>
</tr>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs). When CONTAINER is set to CONTAINER_ALL, this clears the last archive timestamp from all the PDBs, otherwise it clears from only the connected PDB.</td>
</tr>
<tr>
<td>database_id</td>
<td>Database ID (DBID) of the audit records to cleanup</td>
</tr>
<tr>
<td>container_guid</td>
<td>Container GUID of the audit records to cleanup</td>
</tr>
<tr>
<td>Note: This parameter has been deprecated but is currently retained for backward compatibility.</td>
<td></td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply:

- The timestamp for only one audit_trail_type can be cleared at a time.
- The following are invalid audit_trail_type values for this procedure and cannot be used:
  - AUDIT_TRAIL_ALL
  - AUDIT_TRAIL_DB_STD
  - AUDIT_TRAIL_FILES

Examples

The following example calls the CLEAR_LAST_ARCHIVE_TIMESTAMP procedure to clear the timestamp value for the operating system (OS) audit trail type.

```sql
BEGIN
DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
    rac_instance_number => 1);
END;
```

Related Topics

- **SET_LAST_ARCHIVE_TIMESTAMP Procedure**
  This procedure sets a timestamp indicating when the audit records were last archived.
  The audit administrator provides the timestamp to be attached to the audit records.
26.7.5 CREATE_PURGE_JOB Procedure

This procedure creates a purge job for periodically deleting the audit trail records.

This procedure carries out the cleanup operation at intervals specified by the user. It calls the CLEAN_AUDIT_TRAIL Procedure to perform the cleanup operation.

The SET_PURGE_JOB_INTERVAL Procedure is used to modify the frequency of the purge job.

The SET_PURGE_JOB_STATUS Procedure is used to enable or disable the purge job.

The DROP_PURGE_JOB Procedure is used to drop a purge job created with the CREATE_PURGE_JOB procedure.

Syntax

```sql
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB(
    audit_trail_type            IN PLS_INTEGER,
    audit_trail_purge_interval  IN PLS_INTEGER,
    audit_trail_purge_name      IN VARCHAR2,
    use_last_arch_timestamp     IN BOOLEAN DEFAULT TRUE,
    container                   IN PLS_INTEGER DEFAULT CONTAINER_CURRENT) ;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the purge job needs to be created. Audit trail types are listed in Table 26-1.</td>
</tr>
<tr>
<td>audit_trail_purge_interval</td>
<td>The interval, in hours, at which the cleanup procedure is called. A lower value means that the cleanup is performed more often.</td>
</tr>
<tr>
<td>audit_trail_purge_name</td>
<td>A name to identify the purge job.</td>
</tr>
<tr>
<td>use_last_arch_timestamp</td>
<td>Specifies whether the last archived timestamp should be used for deciding on the records that should be deleted.</td>
</tr>
<tr>
<td></td>
<td>A value of TRUE indicates that only audit records created before the last archive timestamp should be deleted.</td>
</tr>
<tr>
<td></td>
<td>A value of FALSE indicates that all audit records should be deleted.</td>
</tr>
<tr>
<td></td>
<td>The default value is TRUE.</td>
</tr>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs). When CONTAINER is set to CONTAINER_ALL, it creates one job in the Root PDB and the invocation of this job will invoke cleanup in all the PDBs.</td>
</tr>
</tbody>
</table>

Usage Notes

Use this procedure to schedule the CLEAN_AUDIT_TRAIL Procedure for your audit trail records.
Examples

The following example calls the CREATE_PURGE_JOB procedure to create a cleanup job called CLEANUP, for all audit trail types. It sets the audit_trail_purge_interval parameter to 100. This means that the cleanup job is invoked every 100 hours. It also sets the use_last_arch_timestamp parameter value to TRUE. This means that all audit records older than the last archive timestamp are deleted.

BEGIN
  DBMS_AUDIT_MGMT.CREATE_PURGE_JOB(
    audit_trail_type             =>  DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL,
    audit_trail_purge_interval   =>  100 /* hours */,
    audit_trail_purge_name       =>  'CLEANUP',
    use_last_arch_timestamp      =>  TRUE);
END;

26.7.6 DEINIT_CLEANUP Procedure

This procedure undoes the setup and initialization performed by the INIT_CLEANUP Procedure. The DEINIT_CLEANUP procedure clears the value of the default_cleanup_interval parameter. However, when used for audit tables, it does not move the audit trail tables back to their original tablespace.

Syntax

DBMS_AUDIT_MGMT.DEINIT_CLEANUP(
    audit_trail_type  IN PLS_INTEGER,
    container         IN PLS_INTEGER DEFAULT CONTAINER_CURRENT);

Parameters

Table 26-13 DEINIT_CLEANUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the procedure needs to be called. Audit trail types are listed in Table 26-1</td>
</tr>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs). When CONTAINER is set to CONTAINER_ALL, this de-initializes the audit trail from cleanup in all the pluggable databases, otherwise it de-initializes the audit trail from cleanup in the connected PDB only.</td>
</tr>
</tbody>
</table>

Usage Notes

You cannot invoke this procedure for AUDIT_TRAIL_UNIFIED. Doing so will raise ORA-46250: Invalid value for argument 'AUDIT_TRAIL_TYPE'.

Examples

The following example clears the default_cleanup_interval parameter setting for the standard database audit trail:

BEGIN
  DBMS_AUDIT_MGMT.DEINIT_CLEANUP(
    audit_trail_type  =>  DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL,  
    default_cleanup_interval =>  0,
    container         =>  CONTAINER_CURRENT);
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD);
END;

Related Topics

- INIT_CLEANUP Procedure
  This procedure sets up the audit management infrastructure and a default cleanup interval for the audit trail records.

26.7.7 DROP_OLD_UNIFIED_AUDIT_TABLES Procedure

This procedure drops old unified audit tables following the cloning of a pluggable database (PDB).

Syntax

DBMS_AUDIT_MGMT.DROP_OLD_UNIFIED_AUDIT_TABLES(
    container_guid   IN VARCHAR2) ;

Parameters

Table 26-14  DROP_OLD_UNIFIED_AUDIT_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container_guid</td>
<td>Container GUID of the old unified audit tables</td>
</tr>
</tbody>
</table>

Usage Notes

When a pluggable database gets cloned, the unified audit tables get newly created in the new pluggable database. To drop the old unified audit tables, use the DROP_OLD_UNIFIED_AUDIT_TABLES by specifying the old GUID of the PDB from which the clone was created. You can query the historical GUIDs from the DBA_PDB_HISTORY view for the given PDB.

Examples

BEGIN
    DBMS_AUDIT_MGMT.DROP_OLD_UNIFIED_AUDIT_TABLES (
        container_guid => 'E4721865A9321CB5E043EFA9E80A2D77');
END;

26.7.8 DROP_PURGE_JOB Procedure

This procedure drops the purge job created using the CREATE_PURGE_JOB Procedure. The name of the purge job is passed as an argument.

Syntax

DBMS_AUDIT_MGMT.DROP_PURGE_JOB(
    audit_trail_purge_name   IN VARCHAR2) ;
Parameters

Table 26-15 DROP_PURGE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_purge_name</td>
<td>The name of the purge job which is being deleted. This is the purge job name that you specified with the CREATE_PURGE_JOB Procedure.</td>
</tr>
</tbody>
</table>

Examples

The following example calls the DROP_PURGE_JOB procedure to drop the purge job called CLEANUP.

BEGIN
  DBMS_AUDIT_MGMT.DROP_PURGE_JOB(
    AUDIT_TRAIL_PURGE_NAME  => 'CLEANUP');
END;

Related Topics

- CREATE_PURGE_JOB Procedure
  This procedure creates a purge job for periodically deleting the audit trail records.

26.7.9 FLUSH_UNIFIED_AUDIT_TRAIL Procedure

This deprecated procedure writes the unified audit trail records in the SGA queue to disk.

Note:

This procedure has been deprecated starting in Oracle Database 12c release 12.2.

Syntax

DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL (  
  flush_type      IN BINARY_INTEGER   DEFAULT FLUSH_CURRENT_INSTANCE,  
  container       IN BINARY_INTEGER   DEFAULT CONTAINER_CURRENT);

Parameters

Table 26-16 FLUSH_UNIFIED_AUDIT_TRAIL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flush_type</td>
<td>Takes one of the following two arguments:</td>
</tr>
<tr>
<td></td>
<td>• FLUSH_CURRENT_INSTANCE - Flushes the audit records from SGA queues in that particular RAC instance</td>
</tr>
<tr>
<td></td>
<td>• FLUSH_ALL_INSTANCES - Flushes the audit records from SGA queues in all the RAC instances</td>
</tr>
</tbody>
</table>
Table 26-16 (Cont.) FLUSH_UNIFIED_AUDIT_TRAIL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>The containers where the SGA queues should be flushed. It takes one of the following two arguments:</td>
</tr>
<tr>
<td></td>
<td>· CONTAINER_CURRENT -Flushes the audit records from SGA queues in that particular PDB</td>
</tr>
<tr>
<td></td>
<td>· CONTAINER_ALL -Flushes the audit records from SGA queues in all the active PDBs.</td>
</tr>
</tbody>
</table>

26.7.10 GET_AUDIT_COMMIT_DELAY Function

This function returns the audit commit delay time as the number of seconds. audit commit delay time is the maximum time that it takes to COMMIT an audit record to the database audit trail. If it takes more time to COMMIT an audit record than defined by the audit commit delay time, then a copy of the audit record is written to the operating system (OS) audit trail.

The audit commit delay time value is useful when determining the last archive timestamp for database audit records.

Syntax

```
DBMS_AUDIT_MGMT.GET_AUDIT_COMMIT_DELAY
RETURN NUMBER;
```

26.7.11 GET_AUDIT_TRAIL_PROPERTY_VALUE Function

This procedure returns the property value set by the SET_AUDIT_TRAIL_PROPERTY Procedure.

Syntax

```
DBMS_AUDIT_MGMT.GET_AUDIT_TRAIL_PROPERTY_VALUE(
    audit_trail_type       IN PLS_INTEGER,
    audit_trail_property   IN PLS_INTEGER)
RETURN NUMBER;
```

Parameters

Table 26-17 GET_AUDIT_TRAIL_PROPERTY_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for the timestamp to be retrieved. Audit trail types are listed in Table 26-1.</td>
</tr>
<tr>
<td>audit_trail_property</td>
<td>The audit trail property that is being queried. Audit trail properties are listed in Table 26-2.</td>
</tr>
</tbody>
</table>

Return Values

If the property value is cached in SGA memory, this function will return the value set by the SET_AUDIT_TRAIL_PROPERTY Procedure. Else it will return NULL.
The `GET_AUDIT_TRAIL_PROPERTY_VALUE` function may return an ORA-46250 error if the audit trail property value has been set to `DBMS_AUDIT_MGMT.CLEAN_UP_INTERVAL`. To find the cleanup interval of the purge job, query `SYS.DAM_CLEANUP_JOBS$`.

Examples

The following example prints the property value of `OS_FILE_MAX_AGE` set by the `SET_AUDIT_TRAIL_PROPERTY` Procedure.

```sql
SET_AUDIT_TRAIL_PROPERTY.
SET SERVEROUTPUT ON
DECLARE
    OS_MAX_AGE_VAL NUMBER;
BEGIN
    OS_MAX_AGE_VAL := DBMS_AUDIT_MGMT.GET_AUDIT_TRAIL_PROPERTY_VALUE(
        audit_trail_type          => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
        audit_trail_property      => DBMS_AUDIT_MGMT. OS_FILE_MAX_AGE);
    IF OS_MAX_AGE_VAL is not NULL THEN
        DBMS_OUTPUT.PUT_LINE('The Maximum Age configured for OS Audit files is: ' ||
            OS_MAX_AGE_VAL);
    END IF;
END;
```

Related Topics

- **SET_AUDIT_TRAIL_PROPERTY** Procedure
  This procedure sets an audit trail property for the audit trail type that is specified.

### 26.7.12 GET_LAST_ARCHIVE_TIMESTAMP Function

This procedure returns the timestamp set by the `SET_LAST_ARCHIVE_TIMESTAMP` Procedure in that database instance.

**Syntax**

```sql
DBMS_AUDIT_MGMT.GET_LAST_ARCHIVE_TIMESTAMP(
    audit_trail_type IN PLS_INTEGER)
RETURN TIMESTAMP;
```

**Parameters**

**Table 26-18**  
GET_LAST_ARCHIVE_TIMESTAMP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>audit_trail_type</code></td>
<td>The audit trail type for the timestamp to be retrieved. Audit trail types are listed in Table 26-1.</td>
</tr>
</tbody>
</table>

**Return Values**

In a database that is opened for READ WRITE, since there will no timestamp stored in SGA memory, this function will return NULL. But in a database that is opened for READ ONLY, if a timestamp is set by the `SET_LAST_ARCHIVE_TIMESTAMP` Procedure, the timestamp will be returned. Else it will return NULL.
Usage Notes

This function will return NULL on a database that is opened READ WRITE. Use DBA_AUDIT_MGMT_LAST_ARCH_TS view to check the timestamp set in such a case.

Examples

The following example prints the timestamp set by the SET_LAST_ARCHIVE_TIMESTAMP Procedure on a READ ONLY database.

```
SET SERVEROUTPUT ON
DECLARE
  LAT_TS TIMESTAMP;
BEGIN
  LAT_TS := DBMS_AUDIT_MGMT.GET_LAST_ARCHIVE_TIMESTAMP(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS);
  IF LAT_TS IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('The Last Archive Timestamp is: ' || to_char(LAT_TS));
  END IF;
END;
```

Related Topics

- SET_LAST_ARCHIVE_TIMESTAMP Procedure
  This procedure sets a timestamp indicating when the audit records were last archived. The audit administrator provides the timestamp to be attached to the audit records.

26.7.13 INIT_CLEANUP Procedure

This procedure sets up the audit management infrastructure and a default cleanup interval for the audit trail records.

If the audit trail tables are in the SYSTEM tablespace, then the procedure moves them to the SYSAUX tablespace. If you are using unified auditing, you do not need to run this procedure because the unified audit trail tables are in the SYSAUX tablespace by default. If you are not using unified auditing, refer to Oracle Database Upgrade Guide for documentation which references an environment without unified auditing.

Moving the audit trail tables out of the SYSTEM tablespace enhances overall database performance. The INIT_CLEANUP procedure moves the audit trail tables to the SYSAUX tablespace. If the SET_AUDIT_TRAIL_LOCATION Procedure has already moved the audit tables elsewhere, then no tables are moved.

The SET_AUDIT_TRAIL_LOCATION Procedure enables you to specify an alternate target tablespace for the database audit tables.

The INIT_CLEANUP procedure is currently not relevant for the AUDIT_TRAIL_OS, AUDIT_TRAIL_XML, and AUDIT_TRAIL_FILES audit trail types. No preliminary set up is required for these audit trail types.

See Also:

Table 26-1 for a list of all audit trail types
This procedure also sets a default cleanup interval for the audit trail records.

Syntax

```
DBMS_AUDIT_MGMT.INIT_CLEANUP(
    audit_trail_type          IN PLS_INTEGER,
    default_cleanup_interval  IN PLS_INTEGER,
    container                 IN PLS_INTEGER DEFAULT CONTAINER_CURRENT);
```

Parameters

Table 26-19 INIT_CLEANUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the clean up operation needs to be initialized. Audit trail types are listed in Table 26-1 except AUDIT_TRAIL_UNIFIED</td>
</tr>
<tr>
<td>default_cleanup_interval</td>
<td>The default time interval, in hours, after which the cleanup procedure should be called. The minimum value is 1 and the maximum is 999.</td>
</tr>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs). When CONTAINER is set to CONTAINER_ALL, this initializes the audit trails for clean up in all the pluggable databases, otherwise this initializes the audit trail in the connected PDB only.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply:

- This procedure may involve data movement across tablespaces. This can be a resource intensive operation especially if your database audit trail tables are already populated. Oracle recommends that you invoke the procedure during non-peak hours.
- You should ensure that the SYSAUX tablespace, into which the audit trail tables are being moved, has sufficient space to accommodate the audit trail tables. You should also optimize the SYSAUX tablespace for frequent write operations.
- You can change the default_cleanup_interval later using the SET_AUDIT_TRAIL_PROPERTY Procedure.
- If you do not wish to move the audit trail tables to the SYSAUX tablespace, then you should use the DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION procedure to move the audit trail tables to another tablespace before calling the INIT_CLEANUP procedure.
- Invoking this procedure with AUDIT_TRAIL_UNIFIED results in ORA-46250. It requires no initializations for cleanup since it is cleanup-ready by default.

See Also:

"SET_AUDIT_TRAIL_LOCATION Procedure"
Examples

The following example calls the INIT_CLEANUP procedure to set a default_cleanup_interval of 12 hours for all audit trail types:

```
BEGIN
  DBMS_AUDIT_MGMT.INIT_CLEANUP(
    audit_trail_type   => DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL,
    default_cleanup_interval   => 12 /* hours */);
END;
```

See Also:
- Table 26-1 for a list of all audit trail types

26.7.14 IS_CLEANUP_INITIALIZED Function

This function checks to see if the INIT_CLEANUP Procedure has been run for an audit trail type.

The IS_CLEANUP_INITIALIZED function returns TRUE if the procedure has already been run for the audit trail type. It returns FALSE if the procedure has not been run for the audit trail type.

This function is currently not relevant for the AUDIT_TRAIL_OS, AUDIT_TRAIL_XML, and AUDIT_TRAIL_FILES audit trail types. The function always returns TRUE for these audit trail types. No preliminary set up is required for these audit trail types.

See Also:
- Table 26-1 for a list of all audit trail types

Syntax

```
DBMS_AUDIT_MGMT.IS_CLEANUP_INITIALIZED(
  audit_trail_type  IN PLS_INTEGER
  container         IN PLS_INTEGER DEFAULT CONTAINER_CURRENT)
RETURN BOOLEAN;
```

Parameters

Table 26-20  IS_CLEANUP_INITIALIZED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the function needs to be called. Note that this does not apply to AUDIT_TRAIL_UNIFIED. Audit trail types are listed in Table 26-1</td>
</tr>
</tbody>
</table>
Table 26-20  (Cont.) IS_CLEANUP_INITIALIZED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs).</td>
</tr>
<tr>
<td></td>
<td>· When CONTAINER is set to CONTAINER_ALL, this function returns the initialization status of all the pluggable databases. The function returns FALSE even if one of the PDBs is not initialized.</td>
</tr>
<tr>
<td></td>
<td>· When CONTAINER is set to CONTAINER_CURRENT, this returns the initialization status of the connected PDB.</td>
</tr>
</tbody>
</table>

Examples

The following example checks to see if the standard database audit trail type has been initialized for cleanup operation. If the audit trail type has not been initialized, then it calls the INIT_CLEANUP Procedure to initialize the audit trail type.

```sql
BEGIN
  IF NOT DBMS_AUDIT_MGMT.IS_CLEANUP_INITIALIZED(DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD) THEN
    DBMS_AUDIT_MGMT.INIT_CLEANUP(
      audit_trail_type          => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD,
      default_cleanup_interval  => 12 /* hours */);
  END IF;
END;
```

Related Topics

- **INIT_CLEANUP Procedure**
  This procedure sets up the audit management infrastructure and a default cleanup interval for the audit trail records.

26.7.15 LOAD_UNIFIED_AUDIT_FILES Procedure

This procedure loads the data from the spillover OS audit files in a unified audit trail into the designated unified audit trail tablespace.

See Also:

Oracle Database Security Guide for information about moving the OS audit trail records into the unified audit trail

Syntax

```sql
DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES (
  container        IN BINARY_INTEGER);
```
**Parameters**

Table 26-21  LOAD_UNIFIED_AUDIT_FILES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs).</td>
</tr>
<tr>
<td></td>
<td>• CONTAINER_CURRENT - loads the unified audit files from $ORACLE_BASE/audit/$ORACLE_SID OS directory to the tables in only current PDB</td>
</tr>
<tr>
<td></td>
<td>• CONTAINER_ALL - loads the unified audit files from $ORACLE_BASE/audit/$ORACLE_SID OS directory to the tables in the respective PDBs, but for all the active PDBs</td>
</tr>
</tbody>
</table>

26.7.16 SET_AUDIT_TRAIL_LOCATION Procedure

This procedure moves the audit trail tables from their current tablespace to a user-specified tablespace.

The **SET_AUDIT_TRAIL_LOCATION** procedure is not relevant for the **AUDIT_TRAIL_OS**, **AUDIT_TRAIL_XML**, and **AUDIT_TRAIL_FILES** audit trail types. The **AUDIT_FILE_DEST** initialization parameter is the only way you can specify the destination directory for these audit trail types.

**See Also:**
- Table 26-1 for a list of all audit trail types
- **AUDIT_FILE_DEST** in the *Oracle Database Reference*

**Syntax**

```sql
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION(
    audit_trail_type            IN PLS_INTEGER,
    audit_trail_location_value  IN VARCHAR2);
```

**Parameters**

Table 26-22  SET_AUDIT_TRAIL_LOCATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the audit trail location needs to be set.</td>
</tr>
<tr>
<td>audit_trail_location_value</td>
<td>Target location or tablespace for the audit trail records</td>
</tr>
</tbody>
</table>
Usage Notes

The following usage notes apply:

• By default, audit trail records are written to the SYSAUX tablespace. You can change the tablespace location to a user tablespace using this procedure. However the designated tablespace must be an automatic storage space management (ASSM) tablespace.

• This procedure is valid for the following audit_trail_type values:
  - AUDIT_TRAIL_AUD_STD
  - AUDIT_TRAIL_FGA_STD
  - AUDIT_TRAIL_DB_STD
  - AUDIT_TRAIL_UNIFIED

• For the audit_trail_type values AUDIT_TRAIL_AUD_STD, AUDIT_TRAIL_FGA_STD, and AUDIT_TRAIL_DB_STD, you should ensure that the target tablespace, into which the audit trail tables are being moved, has sufficient space to accommodate the audit trail tables. You should also optimize the target tablespace for frequent write operations.

• This procedure involves data movement across tablespaces. For the audit_trail_type values AUDIT_TRAIL_AUD_STD, AUDIT_TRAIL_FGA_STD, and AUDIT_TRAIL_DB_STD, this can be a resource intensive operation especially if your database audit trail tables are already populated. Oracle recommends that you invoke the procedure during non-peak hours.

• You optionally can specify an encrypted tablespace for the audit trail location.

• When AUDIT_TRAIL_TYPE is AUDIT_TRAIL_UNIFIED, this procedure sets the tablespace for newer audit records in the unified audit trail but does not move the older audit records. Thus, it is not resource intensive for the unified audit trail.

• The UNIFIED_AUDIT_TRAIL data dictionary view is built on top of an internal relational table. This table is an interval partitioned table (irrespective of database editions) with a default interval of 1 month. This setting means that when you execute the DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION procedure, only newly created partitions of the internal table are created in the new tablespace that is set as part of this procedure. Existing partitions of this table remain in the earlier tablespace (SYSAUX is the default tablespace for this internal table). If you want to change this table’s partition interval, then use the DBMS_AUDIT_MGMT.ALTER_PARTITION_INTERVAL procedure.

Examples

The following example moves the database audit trail tables, AUD$ and FGA_LOG$, from the current tablespace to a user-created tablespace called RECORDS:

```sql
BEGIN
  DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD,
    audit_trail_location_value => 'RECORDS');
END;
```

26.7.17 SET_AUDIT_TRAIL_PROPERTY Procedure

This procedure sets an audit trail property for the audit trail type that is specified.

The procedure sets the properties OS_FILE_MAX_SIZE, OS_FILE_MAX_AGE, and FILE_DELETE_BATCH_SIZE for operating system (OS) and XML audit trail types. The
OS_FILE_MAX_SIZE and OS_FILE_MAX_AGE properties determine the maximum size and age of an audit trail file before a new audit trail file gets created. The FILE_DELETE_BATCH_SIZE property specifies the number of audit trail files that are deleted in one batch.

The procedure sets the properties DB_DELETE_BATCH_SIZE and CLEANUP_INTERVAL for the database audit trail type. DB_DELETE_BATCH_SIZE specifies the batch size in which records get deleted from audit trail tables. This ensures that if a cleanup operation gets interrupted midway, the process does not need to start afresh the next time it is invoked. This is because all batches before the last processed batch are already committed.

The CLEANUP_INTERVAL specifies the frequency, in hours, with which the cleanup procedure is called.

**Syntax**

```sql
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
    audit_trail_type            IN PLS_INTEGER,
    audit_trail_property        IN PLS_INTEGER,
    audit_trail_property_value  IN PLS_INTEGER)
```

**Parameters**

**Table 26-23**  SET_AUDIT_TRAIL_PROPERTY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the property needs to be set. Audit trail types are listed in Table 26-1</td>
</tr>
<tr>
<td>audit_trail_property</td>
<td>The audit trail property that is being set. Audit trail properties are listed in Table 26-2</td>
</tr>
</tbody>
</table>
Table 26-23  (Cont.) SET_AUDIT_TRAIL_PROPERTY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_property_value</td>
<td>The value of the property specified using audit_trail_property. The following are valid values for audit trail properties:</td>
</tr>
<tr>
<td></td>
<td>• OS_FILE_MAX_SIZE can have a minimum value of 1 and maximum value of 2000000. The default value is 10000. OS_FILE_MAX_SIZE is measured in kilobytes (KB).</td>
</tr>
<tr>
<td></td>
<td>• OS_FILE_MAX_AGE can have a minimum value of 1 and a maximum value of 497. The default value is 5. OS_FILE_MAX_AGE is measured in days.</td>
</tr>
<tr>
<td></td>
<td>• DB_DELETE_BATCH_SIZE can have a minimum value of 100 and a maximum value of 1000000. The default value is 10000. DB_DELETE_BATCH_SIZE is measured as the number of audit records that are deleted in one batch.</td>
</tr>
<tr>
<td></td>
<td>• FILE_DELETE_BATCH_SIZE can have a minimum value of 100 and a maximum value of 1000000. The default value is 1000. FILE_DELETE_BATCH_SIZE is measured as the number of audit files that are deleted in one batch.</td>
</tr>
<tr>
<td></td>
<td>• CLEANUP_INTERVAL can have a minimum value of 1 and a maximum value of 999. The default value is set using the INIT_CLEANUP Procedure. CLEANUP_INTERVAL is measured in hours.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply:

- The audit trail properties for which you do not explicitly set values use their default values.

- If you have set both the OS_FILE_MAX_SIZE and OS_FILE_MAX_AGE properties for an operating system (OS) or XML audit trail type, then a new audit trail file gets created depending on which of these two limits is reached first.

  For example, let us take a scenario where OS_FILE_MAX_SIZE is 10000 and OS_FILE_MAX_AGE is 5. If the operating system audit file is already more than 5 days old and has a size of 9000 KB, then a new audit file is opened. This is because one of the limits has been reached.

- The DB_DELETE_BATCH_SIZE property needs to be individually set for the AUDIT_TRAIL_AUD_STD and AUDIT_TRAIL_FGA_STD audit trail types. You cannot set this property collectively using the AUDIT_TRAIL_DB_STD and AUDIT_TRAIL_ALL audit trail types.

- The DB_DELETE_BATCH_SIZE property enables you to control the number of audit records that are deleted in one batch. Setting a large value for this parameter requires increased allocation for the undo log space.
• The `FILE_DELETE_BATCH_SIZE` property needs to be individually set for the `AUDIT_TRAIL_OS` and `AUDIT_TRAIL_XML` audit trail types. You cannot set this property collectively using the `AUDIT_TRAIL_FILES` and `AUDIT_TRAIL_ALL` audit trail types.

• The `FILE_DELETE_BATCH_SIZE` property enables you to control the number of audit files that are deleted in one batch. Setting a very large value may engage the `GEN0` background process for a long time.

• In Oracle Database Standard Edition, you can only associate the tablespace for unified auditing once. You should perform this association before you generate any audit records for the unified audit trail. The default tablespace is `SYSAUX`. After you have associated the tablespace, you cannot modify it on the Standard Edition because the partitioning feature is not supported in the Standard Edition.

Examples

The following example calls the `SET_AUDIT_TRAIL_PROPERTY` procedure to set the `OS_FILE_MAX_SIZE` property for the operating system (OS) audit trail. It sets this property value to 102400. This means that a new audit file gets created every time the current audit file size reaches 100 MB.

```sql
BEGIN
  DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
    audit_trail_type          => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
    audit_trail_property      =>  DBMS_AUDIT_MGMT.OS_FILE_MAX_SIZE,
    audit_trail_property_value => 102400 /* 100MB*/ );
END;
```

The following example calls the `SET_AUDIT_TRAIL_PROPERTY` procedure to set the `OS_FILE_MAX_AGE` property for the operating system (OS) audit trail. It sets this property value to 5. This means that a new audit file gets created every sixth day.

```sql
BEGIN
  DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
    audit_trail_type          => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
    audit_trail_property      =>  DBMS_AUDIT_MGMT.OS_FILE_MAX_AGE,
    audit_trail_property_value => 5 /* days */);
END;
```

The following example calls the `SET_AUDIT_TRAIL_PROPERTY` procedure to set the `DB_DELETE_BATCH_SIZE` property for the `AUDIT_TRAIL_AUD_STD` audit trail. It sets this property value to 100000. This means that during a cleanup operation, audit records are deleted from the `SYS.AUD$` table in batches of size 100000.

```sql
BEGIN
  DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
    audit_trail_type          => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD,
    audit_trail_property      =>  DBMS_AUDIT_MGMT.DB_DELETE_BATCH_SIZE,
    audit_trail_property_value => 100000 /* delete batch size */);
END;
```
26.7.18 SET_LAST_ARCHIVE_TIMESTAMP Procedure

This procedure sets a timestamp indicating when the audit records were last archived. The audit administrator provides the timestamp to be attached to the audit records.

The CLEAN_AUDIT_TRAIL Procedure uses this timestamp to decide on the audit records to be deleted.

Syntax

```
DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
    audit_trail_type         IN PLS_INTEGER,
    last_archive_time        IN TIMESTAMP,
    rac_instance_number      IN PLS_INTEGER DEFAULT NULL,
    container                IN PLS_INTEGER DEFAULT CONTAINER_CURRENT,
    database_id              IN NUMBER DEFAULT NULL,
    container_guid           IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_type</td>
<td>The audit trail type for which the timestamp needs to be set. Audit trail types are listed in Table 26-1.</td>
</tr>
<tr>
<td>last_archive_time</td>
<td>The TIMESTAMP value based on which the audit records or files should be deleted. This indicates the last time when the audit records or files were archived.</td>
</tr>
<tr>
<td>rac_instance_number</td>
<td>The instance number for the Oracle Real Application Clusters (Oracle RAC) instance. The default value is NULL.</td>
</tr>
<tr>
<td>container</td>
<td>Values: CONTAINER_CURRENT for the connected pluggable database (PDB) or CONTAINER_ALL for all pluggable databases (PDBs). When CONTAINER is set to CONTAINER_ALL, this sets the value for last archive timestamp in all the pluggable databases, otherwise it sets the value in the connected PDB only.</td>
</tr>
<tr>
<td>database_id</td>
<td>Database ID (DBID) of the audit records to cleanup</td>
</tr>
<tr>
<td>container_guid</td>
<td>Container GUID of the audit records to cleanup</td>
</tr>
</tbody>
</table>

Note: This parameter has been deprecated but is currently retained for backward compatibility.

Usage Notes

The following usage notes apply:

- The last_archive_time must be specified in Coordinated Universal Time (UTC) when the audit trail types are AUDIT_TRAIL_AUD_STD, AUDIT_TRAIL_FGA_STD, or AUDIT_TRAIL_UNIFIED. This is because the database audit trails store the timestamps in UTC. UTC is also known as Greenwich Mean Time (GMT).
• The last_archive_time must be specified as the local time zone time when the audit trail types are AUDIT_TRAIL_OS or AUDIT_TRAIL_XML. The time zone must be the time zone of the machine where the OS or XML audit files were created. This is because the operating system audit files are cleaned based on the audit file's Last Modification Timestamp property. The Last Modification Timestamp property value is stored in the local time zone of the machine.

• When using an Oracle Real Application Clusters (Oracle RAC) database, Oracle recommends that you use the Network Time Protocol (NTP) to synchronize individual Oracle RAC nodes.

Examples

The following example calls the SET_LAST_ARCHIVE_TIMESTAMP procedure to set the last archive timestamp for the operating system (OS) audit trail type on Oracle RAC instance 1. It uses the TO_TIMESTAMP function to convert a character string into a timestamp value.

A subsequent call to the CLEAN_AUDIT_TRAIL Procedure, with use_last_arch_timestamp set to TRUE, will delete all those OS audit files from the current AUDIT_FILE_DEST directory that were modified before 10-Sep-2012 14:10:10.0.

BEGIN
  DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
    audit_trail_type     => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
    last_archive_time    => TO_TIMESTAMP('12-SEP-0714:10:10.0','DD-MON-RRHH24:MI:SS.FF'),
    rac_instance_number  => 1);
END;

26.7.19 SET_PURGE_JOB_INTERVAL Procedure

This procedure sets the interval at which the CLEAN_AUDIT_TRAIL Procedure is called for the purge job specified.

The purge job must have already been created using the CREATE_PURGE_JOB Procedure.

Syntax

DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL(
  audit_trail_purge_name IN VARCHAR2,
  audit_trail_interval_value IN PLS_INTEGER) ;

Parameters

Table 26-25  SET_PURGE_JOB_INTERVAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_trail_purge_name</td>
<td>The name of the purge job for which the interval is being set. This is the purge job name that you specified with the CREATE_PURGE_JOB Procedure.</td>
</tr>
<tr>
<td>audit_trail_interval_value</td>
<td>The interval, in hours, at which the clean up procedure should be called. This value modifies the audit_trail_purge_interval parameter set using the CREATE_PURGE_JOB Procedure</td>
</tr>
</tbody>
</table>
**Usage Notes**

Use this procedure to modify the `audit_trail_purge_interval` parameter set using the `CREATE_PURGE_JOB` Procedure.

**Examples**

The following example calls the `SET_PURGE_JOB_INTERVAL` procedure to change the frequency at which the purge job called `CLEANUP` is invoked. The new interval is set to 24 hours.

```sql
BEGIN
  DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL(
    AUDIT_TRAIL_PURGE_NAME       => 'CLEANUP',
    AUDIT_TRAIL_INTERVAL_VALUE   => 24 );
END;
```

**Related Topics**

- **CLEAN_AUDIT_TRAIL Procedure**
  This procedure deletes audit trail records.

## 26.7.20 SET_PURGE_JOB_STATUS Procedure

This procedure enables or disables the specified purge job.

The purge job must have already been created using the `CREATE_PURGE_JOB` Procedure.

**Syntax**

```sql
DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS(
  audit_trail_purge_name    IN VARCHAR2,
  audit_trail_status_value  IN PLS_INTEGER) ;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>audit_trail_purge_name</code></td>
<td>The name of the purge job for which the status is being set. This is the purge job name that you specified with the <code>CREATE_PURGE_JOB</code> Procedure.</td>
</tr>
<tr>
<td><code>audit_trail_status_value</code></td>
<td>One of the values specified in Table 26-3. The value <code>PURGE_JOB_ENABLE</code> enables the specified purge job. The value <code>PURGE_JOB_DISABLE</code> disables the specified purge job.</td>
</tr>
</tbody>
</table>

**Examples**

The following example calls the `SET_PURGE_JOB_STATUS` procedure to enable the `CLEANUP` purge job.

```sql
BEGIN
  DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS(
    audit_trail_purge_name => 'CLEANUP',
    audit_trail_status_value => DBMS_AUDIT_MGMT.PURGE_JOB_ENABLE);
END;
```
26.7.21 TRANSFER_UNIFIED_AUDIT_RECORDS Procedure

This procedure transfers unified audit records that were in a pre-upgraded Oracle database to an internal relational table that is designed to improve read performance.

In the pre-upgraded Oracle database, these records resided in the common logging infrastructure (CLI) SGA back-end tables.

**Syntax**

```sql
DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS (
    container_guid       IN VARCHAR2   DEFAULT NULL);
```

**Parameters**

**Table 26-27   TRANSFER_UNIFIED_AUDIT_RECORDS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container_guid</td>
<td>The GUID of the container of the associated CLI back-end table. This back-end table contains the audit records from the pre-upgraded Oracle database. If you omit this setting, then the GUID of the current container is used.</td>
</tr>
</tbody>
</table>

**Usage Notes**

- It is not mandatory to run `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` after an upgrade, but for better read performance of the unified audit trail, Oracle highly recommends that you run this procedure.

- The `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` is designed to be a one-time operation, to be performed after you upgrade from Oracle Database 12c release 12.1.

- There is one CLI back-end table per GUID of the container. You can find the GUIDs for containers by querying the `PDB_GUID` column of the `DBA_PDB_HISTORY` data dictionary view. Execute the `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` procedure by passing each of these GUIDs one by one to ensure that you move the unified audit records from these CLI back-end tables to the `AUDSYS.AUD$UNIFIED` table.

- In a multitenant environment, you must run the `DBMS_AUDIT_MGMT.TRANSFER_UNIFIED_AUDIT_RECORDS` procedure only in the container to which the transfer operation applies, whether it is the root or an individual PDB. You cannot run this procedure in the root, for example, to transfer audit records in a PDB.

- If you have a high rate of audit record generation and your database supports partitioning, then you may want to use the `DBMS_AUDIT_MGMT.ALTER_PARTITION_INTERVAL` procedure to alter the partition interval setting for the internal relational table. See `ALTER_PARTITION_INTERVAL Procedure` for more information.
The `DBMS_AUDIT_UTIL` package provides functions that enable you to format the output of queries to the `DBA_FGA_AUDIT_TRAIL`, `DBA_AUDIT_TRAIL`, `UNIFIED_AUDIT_TRAIL`, and `V$XML_AUDIT_TRAIL` views.

This chapter contains the following topics:

• DBMS_AUDIT_UTIL Overview
• DBMS_AUDIT_UTIL Security Model
• DBMS_AUDIT_UTIL Views
• Summary of DBMS_AUDIT_UTIL Subprograms

27.1 DBMS_AUDIT_UTIL Overview

The functions in the `DBMS_AUDIT_UTIL` package enable you to format the output of queries to the `RLS_INFO` column of several audit trail views so that the output appear in separate rows.

These functions use a cursor to find and format each row of the corresponding view. To use the functions in this package, include the function in a query to one of the following views:

• `DBA_FGA_AUDIT_TRAIL` data dictionary view, for the `DECODE_RLS_INFO_ATRAIL_FGA` function
• `DBA_AUDIT_TRAIL` data dictionary view, for the `DECODE_RLS_INFO_ATRAIL_STD` function
• `UNIFIED_AUDIT_TRAIL` data dictionary view, for the `DECODE_RLS_INFO_ATRAIL_UNI` function
• `V$XML_AUDIT_TRAIL` dynamic view, for the `DECODE_RLS_INFO_ATRAIL_XML` function

27.2 DBMS_AUDIT_UTIL Security Model

All `DBMS_AUDIT_UTIL` subprograms require the user to have `EXECUTE` privilege on the `DBMS_AUDIT_UTIL` package.

The `SYSDBA` administrative privilege and `AUDIT_ADMIN` and `AUDIT_VIEWER` roles have the `EXECUTE` privilege on the `DBMS_AUDIT_UTIL` package by default. An auditor can view audit data after being granted the `AUDIT_VIEWER` role.

Oracle strongly recommends that only audit administrators have the `EXECUTE` privilege on the `DBMS_AUDIT_UTIL` package and be granted the `AUDIT_VIEWER` role.
27.3 DBMS_AUDIT_UTIL Views

The views in this section display the audit information used by the DBMS_AUDIT_UTIL package function.

Table 27-1 displays the DBMS_AUDIT_UTIL views.

Table 27-1 Views Used by DBMS_AUDIT_UTIL

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_FGA_AUDIT_TRAIL</td>
<td>Displays fine-grained audit record information; used with the DECODE_RLS_INFO_ATRAIL_FGA function</td>
</tr>
<tr>
<td>DBA_AUDIT_TRAIL</td>
<td>Displays standard audit record information; used with the DECODE_RLS_INFO_ATRAIL_STD function</td>
</tr>
<tr>
<td>UNIFIED_AUDIT_TRAIL</td>
<td>Displays unified audit trail information; used with the DECODE_RLS_INFO_ATRAIL_XML function</td>
</tr>
<tr>
<td>V$XML_AUDIT_TRAIL</td>
<td>Displays XML audit record information; used with the DECODE_RLS_INFO_ATRAIL_UNI function</td>
</tr>
</tbody>
</table>

27.4 Summary of DBMS_AUDIT_UTIL Subprograms

This table lists the DBMS_AUDIT_UTIL subprograms and their descriptions.

Table 27-2 DBMS_AUDIT_UTIL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECODE_RLS_INFO_ATRAIL_FGA Function</td>
<td>Reformats the output for queries to the RLS_INFO column of the DBA_FGA_AUDIT_TRAIL data dictionary view</td>
</tr>
<tr>
<td>DECODE_RLS_INFO_ATRAIL_STD Function</td>
<td>Reformats the output for queries to the RLS_INFO column of the DBA_AUDIT_TRAIL data dictionary view</td>
</tr>
<tr>
<td>DECODE_RLS_INFO_ATRAIL_UNI Function</td>
<td>Reformats the output for queries to the RLS_INFO column of the UNIFIED_AUDIT_TRAIL data dictionary view</td>
</tr>
<tr>
<td>DECODE_RLS_INFO_ATRAIL_XML Function</td>
<td>Reformats the output for queries to the RLS_INFO column of the V$XML_AUDIT_TRAIL dynamic view</td>
</tr>
</tbody>
</table>

27.4.1 DECODE_RLS_INFO_ATRAIL_FGA Function

This function reformats the output for queries to the RLS_INFO column of the DBA_FGA_AUDIT_TRAIL data dictionary view so that the output is easily readable. It is used for the concatenated Oracle Virtual Private Database predicates for multiple fine-
grained audit policies in an environment that has not been enabled for unified auditing. It returns the output in separate rows.

See Also:

Oracle Database Security Guide regarding fine-grained auditing

Syntax

```
DECODE_RLS_INFO_ATRAIL_FGA(
    IN_CURSOR                  REF CURSOR               IN)
RETURN PIPELINED ROW;
```

Parameters

Except for the `IN_CURSOR` parameter, the parameters for the `DECODE_RLS_INFO_ATRAIL_FGA` function are the same as the columns in the `DBA_FGA_AUDIT_TRAIL` data dictionary view. See Oracle Database Reference for more information about this view.

Usage Notes

- To use this function, include it in a query to the `DBA_FGA_AUDIT_TRAIL` data dictionary view, using a cursor similar to the example shown in the following section.
- See Oracle Database Reference for more information about the `DBA_FGA_AUDIT_TRAIL` data dictionary view.

Example

```
SELECT DB_USER, OBJECT_NAME, SQL_TEXT
RLS_PREDICATE, RLS_POLICY_TYPE, RLS_POLICY_OWNER, RLS_POLICY_NAME
FROM TABLE (DBMS_AUDIT_UTIL.DECODE_RLS_INFO_ATRAIL_FGA
    (CURSOR (SELECT * FROM DBA_FGA_AUDIT_TRAIL)));
```

Return Values

A piped row with decoded values of `DBA_FGA_AUDIT_TRAIL.RLS_INFO` column.

27.4.2 DECODE_RLS_INFO_ATRAIL_STD Function

This function reformats the output for queries to the `RLS_INFO` column of the `DBA_AUDIT_TRAIL` data dictionary view so that the output is easily readable. It is used for the concatenated Oracle Virtual Private Database predicates for multiple standard audit records in an environment that has not been enabled for unified auditing. It returns the output in separate rows.

See Also:

Oracle Database Security Guide regarding auditing
Syntax

```
DECODE_RLS_INFO_ATRAIL_STD(
    IN_CURSOR            REF CURSOR   IN
) RETURN PIPELINED ROW;
```

Parameters

Except for the `IN_CURSOR` parameter, the parameters for the `DECODE_RLS_INFO_ATRAIL_STD` function are the same as the columns in the `DBA_AUDIT_TRAIL` data dictionary view. See Oracle Database Reference for more information about this view.

Usage Notes

- To use this function, include it in a query to the `DBA_AUDIT_TRAIL` data dictionary view, using a cursor similar to the example shown in the following section.
- See Oracle Database Reference for more information about the `DBA_AUDIT_TRAIL` data dictionary view.

Example

```
SELECT USERNAME, USERHOST, ACTION, OBJ_NAME, OBJ_PRIVILEGE,
       RLS_PREDICATE, RLS_POLICY_TYPE, RLS_POLICY_OWNER, RLS_POLICY_NAME
FROM TABLE (DBMS_AUDIT_UTIL.DECODE_RLS_INFO_ATRAIL_STD
            (CURSOR (SELECT * FROM DBA_AUDIT_TRAIL)));
```

Return Values

A piped row with decoded values of `DBA_AUDIT_TRAIL.RLS_INFO` column

27.4.3 DECODE_RLS_INFO_ATRAIL_UNI Function

This function reformats the output for queries to the `RLS_INFO` column of the `V$XML_AUDIT_TRAIL` dynamic view so that the output is easily readable. It is used for the concatenated Oracle Virtual Private Database predicates for multiple XML audit records in an environment that has not been enabled for unified auditing. It returns the output in separate rows.

See Also:

Oracle Database Security Guide regarding unified auditing

Syntax

```
DECODE_RLS_INFO_ATRAIL_UNI(
    IN_CURSOR            REF CURSOR   IN
) RETURN PIPELINED ROW;
```

Parameters

Except for the `IN_CURSOR` parameter, the parameters for the `DECODE_RLS_INFO_ATRAIL_UNI` function are the same as the columns in the
**V$XML_AUDIT_TRAIL** data dictionary view. See *Oracle Database Reference* for more information about this view.

**Usage Notes**

- To use this function, include it in a query to the **V$XML_AUDIT_TRAIL** dynamic view, using a cursor similar to the example shown in the following section.
- See *Oracle Database Reference* for more information about the **V$XML_AUDIT_TRAIL** dynamic view.

**Example**

```sql
SELECT OBJECT_NAME, SQL_TEXT
RLS_PREDICATE, RLS_POLICY_TYPE, RLS_POLICY_OWNER, RLS_POLICY_NAME
FROM TABLE (DBMS_AUDIT_UTIL.DECODE_RLS_INFO_ATRAIL_UNI
(CURSOR (SELECT * FROM V$XML_AUDIT_TRAIL)));
```

**Return Values**

A piped row with decoded values of **V$XML_AUDIT_TRAIL.RLS_INFO** column

### 27.4.4 DECODE_RLS_INFO_ATRAIL_XML Function

This function reformats the output for queries to the **RLS_INFO** column of the **UNIFIED_AUDIT_TRAIL** data dictionary view so that the output is easily readable. It is used for the concatenated Oracle Virtual Private Database predicates for multiple audit records from unified audit policies. It returns the output in separate rows.

**See Also:**

*Oracle Database Security Guide* regarding auditing

**Syntax**

```sql
DECODE_RLS_INFO_ATRAIL_XML(
   IN_CURSOR                  REF CURSOR               IN)
RETURN PIPELINED ROW;
```

**Parameters**

Except for the **IN_CURSOR** parameter, the parameters for the **DECODE_RLS_INFO_ATRAIL_XML** function are the same as the columns in the **UNIFIED_AUDIT_TRAIL** data dictionary view. See *Oracle Database Reference* for more information about this view.

**Usage Notes**

- To use this function, include it in a query to the **UNIFIED_AUDIT_TRAIL** data dictionary view, using a cursor similar to the example shown in the following section.
- See *Oracle Database Reference* for more information about the **UNIFIED_AUDIT_TRAIL** data dictionary view.
Example

SELECT DBUSERNAME, ACTION_NAME, OBJECT_NAME, SQL_TEXT,
       RLS_PREDICATE, RLS_POLICY_TYPE, RLS_POLICY_OWNER, RLS_POLICY_NAME
FROM TABLE (DBMS_AUDIT_UTIL.DECODE_RLS_INFO_ATRAIL_XML
            (CURSOR (SELECT * FROM UNIFIED_AUDIT_TRAIL)));

Return Values

A piped row with decoded values of UNIFIED_AUDIT_TRAIL.RLS_INFO column
The DBMS_AUTO_REPORT package provides an interface to view SQL Monitoring and Real-time Automatic Database Diagnostic Monitor (ADDM) data that has been captured into Automatic Workload Repository (AWR). It also provides subprograms to control the behavior of how these data are captured to AWR.

See Also:
Oracle Database SQL Tuning Guide for more information about reporting database operations

This chapter contains the following topics:

• Overview
• Security Model
• Summary of DBMS_AUTO_REPORT Subprograms

28.1 DBMS_AUTO_REPORT Overview

This package provides an interface to view SQL Monitoring and Real-time ADDM data that has been captured into AWR. It also provides subprograms to control the behavior of how these data are captured to AWR. Captured data are stored in AWR and exposed via 2 views: DBA_HIST_REPORTS and DBA_HIST_REPORTS_DETAILS.

28.2 DBMS_AUTO_REPORT Security Model

This package is available to PUBLIC and performs its own security checking.

28.3 Summary of DBMS_AUTO_REPORT Subprograms

This table describes the parameters of the DBMS_AUTO_REPORT package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FINISH_REPORT_CAPTURE Procedure</td>
<td>Ends the complete capture of SQL monitor data that was started with the START_REPORT_CAPTURE Procedure.</td>
</tr>
<tr>
<td>REPORT_REPOSITORY_DETAIL Function</td>
<td>Obtains the stored report for a given report ID</td>
</tr>
<tr>
<td>REPORT_REPOSITORY_DETAIL_XML Function</td>
<td>Obtains the stored XML report for a given report ID</td>
</tr>
</tbody>
</table>
Table 28-1  (Cont.) DBMS_AUTO_REPORT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT_REPOSITORY_LIST_XML Function</td>
<td>Obtains an XML report of the list of SQL Monitor and Real-time ADDM data captured in AWR</td>
</tr>
<tr>
<td>START_REPORT_CAPTURE Procedure</td>
<td>Captures SQL monitor data of any newly monitored SQLs every minute since the last run of the capture cycle, and stores it in AWR.</td>
</tr>
</tbody>
</table>

28.3.1 FINISH_REPORT_CAPTURE Procedure

This procedure ends the complete capture of SQL monitor data that was started with the START_REPORT_CAPTURE procedure.

After calling this subprogram, capture of data continues every minute except that it is not captured for all active SQLs but only for those deemed important, namely the top 5 SQLs (by elapsed time, or elapsed time*DOP in case of PQ) whose monitoring has completed.

Syntax

```
DBMS_AUTO_REPORT.FINISH_REPORT_CAPTURE;
```

Related Topics

- START_REPORT_CAPTURE Procedure
  This procedure captures SQL monitor data of any newly monitored SQLs every minute since the last run of the capture cycle, and stores it in AWR.

28.3.2 REPORT_REPOSITORY_DETAIL Function

This procedure obtains the stored report for a given report ID in the specified format such as XML or HTML.

Syntax

```
DBMS_AUTO_REPORT.REPORT_REPOSITORY_DETAIL (rid => IN NUMBER DEFAULT NULL, type => IN VARCHAR2 DEFAULT 'XML', base_path => IN VARCHAR2 DEFAULT NULL) RETURNS CLOB
```

Parameters

Table 28-2  REPORT_REPOSITORY_DETAIL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rid</td>
<td>ID of the stored report which returned by the function</td>
</tr>
<tr>
<td>type</td>
<td>Desired format of the report. Values can be 'XML', 'TEXT', 'HTML', 'EM' or 'ACTIVE'. The last two options generate a report in the same format called active HTML. Default value is 'XML'.</td>
</tr>
<tr>
<td>base_path</td>
<td>Unused/Non-operative</td>
</tr>
</tbody>
</table>
Return Values
The persisted report for the given record ID

28.3.3 REPORT_REPOSITORY_DETAIL_XML Function

This procedure obtains the stored XML report for a given report ID.

Syntax

```sql
DBMS_AUTO_REPORT.REPORT_REPOSITORY_DETAIL_XML (  
  rid              IN NUMBER    DEFAULT NULL,
  base_path        IN VARCHAR2  DEFAULT NULL)
RETURNS XMLTYPE
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rid</td>
<td>ID of the stored report which returned by the function</td>
</tr>
<tr>
<td>base_path</td>
<td>Unused/Non-operative</td>
</tr>
</tbody>
</table>

Return Values
The persisted XML report for the given record ID

28.3.4 REPORT_REPOSITORY_LIST_XML Function

This procedure obtains an XML report of the list of SQL Monitor and Real-time ADDM data captured in AWR.

The input parameters can be used to select and restrict which captured data will be included in the list report. All parameters are optional.

Syntax

```sql
DBMS_AUTO_REPORT.REPORT_REPOSITORY_LIST_XML (  
  active_since              IN DATE     DEFAULT NULL,
  active_upto               IN DATE     DEFAULT NULL,
  snapshot_id               IN NUMBER   DEFAULT NULL,
  dbid                      IN NUMBER   DEFAULT NULL,
  inst_id                   IN NUMBER   DEFAULT NULL,
  con_dbid                  IN NUMBER   DEFAULT NULL,
  session_id                IN NUMBER   DEFAULT NULL,
  session_serial            IN NUMBER   DEFAULT NULL,
  component_name            IN VARCHAR2 DEFAULT NULL,
  key1                      IN VARCHAR2 DEFAULT NULL,
  key2                      IN VARCHAR2 DEFAULT NULL,
  key3                      IN VARCHAR2 DEFAULT NULL,
  report_level              IN VARCHAR2 DEFAULT 'TYPICAL',
  base_path                 IN VARCHAR2 DEFAULT NULL)
RETURNS XMLTYPE
```
## Parameters

**Table 28-4  REPORT_REPOSITORY_LIST_XML Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active_since</td>
<td>Start of a time range used to select data. When a time range is specified, only those data are included in the list that were active during the time range. When no value is specified the time range is chosen as the last 24 hours ending at the current system time.</td>
</tr>
<tr>
<td>active_upto</td>
<td>Same as <code>active_since</code> except that it is the end of the time range</td>
</tr>
<tr>
<td>snapshot_id</td>
<td>If a value is specified, only those data captured during the specified snapshot ID are included in the list report. If no value is specified, no filtering is performed on snapshot ID.</td>
</tr>
<tr>
<td>dbid</td>
<td>If a value is specified, only those data captured for the specified database ID are included in the list report. If no value is specified, no filtering is performed on database ID.</td>
</tr>
<tr>
<td>inst_id</td>
<td>If a value is specified, only those data captured on the specified instance number are included in the list report. If no value is specified, no filtering is performed on the instance ID.</td>
</tr>
<tr>
<td>con_dbid</td>
<td>If a value is specified, only those data captured on the specified container DBID are included in the list report. If no value is specified, no filtering is performed on the container DBID.</td>
</tr>
<tr>
<td>session_id</td>
<td>If a value is specified, only those data captured for the specified session ID are included in the list report. If no value is specified, no filtering is performed on session ID.</td>
</tr>
<tr>
<td>session_serial</td>
<td>If a value is specified, only those data captured for the specified session are included in the list report. If no value is specified, no filtering is performed on session serial number. This parameter should be used in conjunction with the <code>session_id</code> parameter.</td>
</tr>
<tr>
<td>component_name</td>
<td>Can be 'sqlmonitor' for SQL Monitor data or 'rtaddm' for Real-time ADDM data. If a value is specified then data pertaining only to the specified component will be included in the list report. If no value is specified, no filtering is performed.</td>
</tr>
<tr>
<td>key1</td>
<td>Key value relevant to a component. For SQL Monitor, <code>key1</code> is the SQL ID of the captured SQL statement. If a value is specified, only those data having specified value for <code>key1</code> are included, else no filtering is performed on <code>key1</code>.</td>
</tr>
<tr>
<td>key2</td>
<td>Key value relevant to a component. For SQL Monitor, <code>key2</code> is the SQL execution ID of the captured SQL statement. If a value is specified, only those data having specified value for <code>key2</code> are included, else no filtering is performed on <code>key2</code>.</td>
</tr>
<tr>
<td>key3</td>
<td>Key value relevant to a component. For SQL Monitor, <code>key3</code> is the SQL execution start time of the captured SQL statement. If a value is specified, then only those data having specified value for <code>key3</code> are included, else no filtering is performed on <code>key3</code>.</td>
</tr>
<tr>
<td>report_level</td>
<td>Currently only 'TYPICAL' is used</td>
</tr>
<tr>
<td>base_path</td>
<td>Unused/Non-operative</td>
</tr>
</tbody>
</table>
28.3.5 START_REPORT_CAPTURE Procedure

This procedure captures SQL monitor data of any newly monitored SQLs every minute since the last run of the capture cycle, and stores it in AWR.

Every capture cycle attempts to capture data for SQLs that are not currently executing or queued. This is a complete capture since data of all newly monitored SQLs is captured. It continues to run every minute until it is explicitly ended with the FINISH_REPORT_CAPTURE Procedure. In the case of a RAC system, the capture will start on each node of the cluster.

Syntax

DBMS_AUTO_REPORT.START_REPORT_CAPTURE;
29

DBMS_AUTO_SQLTUNE

The DBMS_AUTO_SQLTUNE package is the interface for managing the Automatic SQL Tuning task. Unlike DBMS_SQLTUNE, the DBMS_AUTO_SQLTUNE package requires the DBA role.

The chapter contains the following topics:

• DBMS_AUTO_SQLTUNE Overview
• DBMS_AUTO_SQLTUNE Security Model
• Summary of DBMS_AUTO_SQLTUNE Subprograms

29.1 DBMS_AUTO_SQLTUNE Overview

The DBMS_AUTO_SQLTUNE package is the interface to SQL Tuning Advisor (DBMS_SQLTUNE) when run within the Autotask framework.

The database creates the automated system task SYS_AUTO_SQL_TUNING_TASK as part of the catalog scripts. This task automatically chooses a set of high-load SQL from AWR and runs the SQL Tuning Advisor on this SQL. The automated task performs the same comprehensive analysis as any other SQL Tuning task.

The automated task tests any SQL profiles it finds by executing both the old and new query plans. Automatic SQL Tuning differs from manual SQL tuning in one important way. If automatic implementation of SQL profiles is enabled (the default is disabled), then the database implements any SQL profiles that promise a great performance benefit. The implementation occurs at tuning time so that the database can immediately benefit from the new plan. You can enable or disable automatic implementation by using the SET_AUTO_TUNING_TASK_PARAMETER API to set the ACCEPT_SQL_PROFILES parameter.

In each maintenance window, the automated tuning task stores its results as a new execution. Each execution result has the same task name but a different execution name. Query the DBA_ADVISOR_EXECUTIONS view for information about task executions. Use the REPORT_AUTO_TUNING_TASK Function to view reports that span multiple executions.

29.2 DBMS_AUTO_SQLTUNE Security Model

This package is available to users with the DBA role. For other users, you must grant the EXECUTE privilege on the package explicitly. Note that the EXECUTE_AUTO_TUNING_TASK procedure is an exception: only SYS can invoke it.

Users can call APIs in this package to control how the automatic tuning task behaves when it runs, such as enabling automatic SQL profile creation and configuring the total and per-SQL time limits under which the task runs. Because these settings affect the overall performance of the database, it may not be appropriate for all users with the ADVISOR privilege to have access to this package.
29.3 Summary of DBMS_AUTO_SQLTUNE Subprograms

The DBMS_AUTO_SQLTUNE package contains EXECUTE, REPORT, and SET subprograms.

Table 29-1   DBMS_AUTO_SQLTUNE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE_AUTO_TUNING_TASK Function and Procedure</td>
<td>Executes the Automatic SQL Tuning task immediately (SYS only)</td>
</tr>
<tr>
<td>REPORT_AUTO_TUNING_TASK Function</td>
<td>Displays a text report of the automatic tuning task’s history</td>
</tr>
<tr>
<td>SET_AUTO_TUNING_TASK_PARAMETER Procedures</td>
<td>Changes a task parameter value for the daily automatic runs</td>
</tr>
</tbody>
</table>

29.3.1 EXECUTE_AUTO_TUNING_TASK Function and Procedure

This function and procedure executes the Automatic SQL Tuning task (SYS_AUTO_SQL_TUNING_TASK).

Both the function and the procedure run in the context of a new task execution. The difference is that the function returns the name of the new execution.

Syntax

DBMS_AUTO_SQLTUNE.EXECUTE_AUTO_TUNING_TASK(
    execution_name    IN VARCHAR2               := NULL,
    execution_params  IN DBMS_ADVISOR.argList   := NULL,
    execution_desc    IN VARCHAR2               := NULL)
RETURN VARCHAR2;

Parameters

Table 29-2   EXECUTE_AUTO_TUNING_TASK Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execution_name</td>
<td>A name to qualify and identify an execution. If not specified, it is generated by the advisor and returned by function.</td>
</tr>
<tr>
<td>execution_params</td>
<td>List of parameters (name, value) for the specified execution. The execution parameters have effect only on the execution for which they are specified. They override the values for the parameters stored in the task (set through the SET_AUTO_TUNING_TASK_PARAMETER Procedures).</td>
</tr>
<tr>
<td>execution_desc</td>
<td>A 256-length string describing the execution</td>
</tr>
</tbody>
</table>
Usage Notes

Only SYS can invoke this subprogram. A tuning task can be executed multiple times without having to reset it.

Examples

EXEC DBMS_AUTO_SQLTUNE.EXECUTE_AUTO_TUNING_TASK('SYS_AUTO_SQL_TUNING_TASK');

29.3.2 REPORT_AUTO_TUNING_TASK Function

This procedure displays the results of an Automatic SQL Tuning task.

Syntax

DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK(
    begin_exec      IN   VARCHAR2   := NULL,
    end_exec        IN   VARCHAR2   := NULL,
    type            IN   VARCHAR2   := 'TEXT',
    level           IN   VARCHAR2   := 'TYPICAL',
    section         IN   VARCHAR2   := ALL,
    object_id       IN   NUMBER     := NULL,
    result_limit    IN   NUMBER     := NULL)
RETURN CLOB;

Parameters

Table 29-3  REPORT_AUTO_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin_exec</td>
<td>Name of the beginning task execution to use. If NULL, the report is generated for the most recent task execution.</td>
</tr>
<tr>
<td>end_exec</td>
<td>Name of the ending task execution to use. If NULL, the report is generated for the most recent task execution.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the report to produce. Possible values are TEXT which produces a text report.</td>
</tr>
<tr>
<td>level</td>
<td>Level of detail in the report:</td>
</tr>
<tr>
<td></td>
<td>• BASIC: simple version of the report. Just show info about the actions taken by the advisor.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: show information about every statement analyzed, including requests not implemented.</td>
</tr>
<tr>
<td></td>
<td>• ALL: highly detailed report level, also provides annotations about statements skipped over.</td>
</tr>
<tr>
<td>section</td>
<td>Section of the report to include:</td>
</tr>
<tr>
<td></td>
<td>• SUMMARY: summary information</td>
</tr>
<tr>
<td></td>
<td>• FINDINGS: tuning findings</td>
</tr>
<tr>
<td></td>
<td>• PLAN: explain plans</td>
</tr>
<tr>
<td></td>
<td>• INFORMATION: general information</td>
</tr>
<tr>
<td></td>
<td>• ERROR: statements with errors</td>
</tr>
<tr>
<td></td>
<td>• ALL: all sections</td>
</tr>
</tbody>
</table>
Table 29-3  (Cont.) REPORT_AUTO_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>Advisor framework object id that represents a single statement to restrict reporting to. NULL for all statements. Only valid for reports that target a single execution.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Maximum number of SQL statements to show in the report</td>
</tr>
</tbody>
</table>

Return Values

A CLOB containing the desired report.

Examples

```sql
-- Get the whole report for the most recent execution
SELECT DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK
FROM DUAL;

-- Show the summary for a range of executions
SELECT DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK(:begin_exec, :end_exec, 'TEXT', 'TYPICAL', 'SUMMARY')
FROM DUAL;

-- Show the findings for the statement of interest
SELECT DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK(:exec, :exec, 'TEXT', 'TYPICAL', 'FINDINGS', 5)
FROM DUAL;
```

29.3.3 SET_AUTO_TUNING_TASK_PARAMETER Procedures

This procedure updates the value of a SQL tuning parameter of type VARCHAR2 or NUMBER for SYS_AUTO_SQL_TUNING_TASK.

Syntax

```sql
DBMS_AUTO_SQLTUNE.SET_AUTO_TUNING_TASK_PARAMETER(
    parameter  IN VARCHAR2,
    value      IN VARCHAR2);

DBMS_AUTO_SQLTUNE.SET_AUTO_TUNING_TASK_PARAMETER(
    parameter  IN VARCHAR2,
    value      IN NUMBER);
```
Parameters

Table 29-4  SET_AUTO_TUNING_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>Name of the parameter to set. The possible tuning parameters that can be set by this procedure using the parameter in the form VARCHAR2:</td>
</tr>
<tr>
<td></td>
<td>- MODE: tuning scope (comprehensive, limited)</td>
</tr>
<tr>
<td></td>
<td>- USERNAME: username under which the statement is parsed</td>
</tr>
<tr>
<td></td>
<td>- DAYS_TO_EXPIRE: number of days until the task is deleted</td>
</tr>
<tr>
<td></td>
<td>- EXECUTION_DAYS_TO_EXPIRE: number of days until the task’s executions is deleted (without deleting the task)</td>
</tr>
<tr>
<td></td>
<td>- DEFAULT_EXECUTION_TYPE: the task defaults to this type of execution when none is specified by the EXECUTE_AUTO_TUNING_TASK Function and Procedure.</td>
</tr>
<tr>
<td></td>
<td>- TIME_LIMIT: global time out (seconds)</td>
</tr>
<tr>
<td></td>
<td>- LOCAL_TIME_LIMIT: per-statement time out (seconds)</td>
</tr>
<tr>
<td></td>
<td>- TEST_EXECUTE: FULL/AUTO/OFF.</td>
</tr>
<tr>
<td></td>
<td>- FULL - test-execute for as much time as necessary, up to the local time limit for the SQL (or the global task time limit if no SQL time limit is set)</td>
</tr>
<tr>
<td></td>
<td>- AUTO - test-execute for an automatically-chosen time proportional to the tuning time</td>
</tr>
<tr>
<td></td>
<td>- OFF - do not test-execute</td>
</tr>
<tr>
<td></td>
<td>- BASIC_FILTER: basic filter for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>- OBJECT_FILTER: object filter for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>- PLAN_FILTER: plan filter for SQL tuning set (see SELECT_SQLSET for possible values)</td>
</tr>
<tr>
<td></td>
<td>- RANK_MEASURE1: first ranking measure for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>- RANK_MEASURE2: second possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>- RANK_MEASURE3: third possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>- RESUME_FILTER: a extra filter for SQL tuning sets besides BASIC_FILTER</td>
</tr>
<tr>
<td></td>
<td>- SQL_LIMIT: maximum number of SQL statements to tune</td>
</tr>
<tr>
<td></td>
<td>- SQL_PERCENTAGE: percentage filter of SQL tuning set statements</td>
</tr>
<tr>
<td></td>
<td>The following parameters are supported for the automatic tuning task only:</td>
</tr>
<tr>
<td></td>
<td>- ACCEPT_SQL_PROFILES: TRUE/FALSE: whether the task should accept SQL profiles automatically</td>
</tr>
<tr>
<td></td>
<td>- MAX_AUTO_SQL_PROFILES: maximum number of automatic SQL profiles allowed on the system, in sum</td>
</tr>
<tr>
<td></td>
<td>- MAX_SQL_PROFILES_PER_EXEC: maximum number of SQL profiles that can be automatically implemented per execution of the task.</td>
</tr>
<tr>
<td>value</td>
<td>New value of the specified parameter</td>
</tr>
</tbody>
</table>
The `DBMS_AUTO_TASK_ADMIN` package provides an interface to AUTOTASK functionality. It is used by the DBA as well as Enterprise Manager to access the AUTOTASK controls. Enterprise Manager also uses the AUTOTASK Advisor.

See Also:

`Oracle Database Administrator's Guide` for more information about "Configuring Automated Maintenance Task"

This chapter contains the following sections:

- Deprecated Subprograms
- Security Model
- Constants
- Summary of DBMS_AUTO_TASK_ADMIN Subprograms

### 30.1 DBMS_AUTO_TASK_ADMIN Deprecated Subprograms

The `DBMS_AUTO_TASK_ADMIN` OVERRIDE_PRIORITY subprogram has been deprecated.

**Note:**

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

- OVERRIDE_PRIORITY Procedures

### 30.2 DBMS_AUTO_TASK_ADMIN Security Model

`DBMS_AUTO_TASK_ADMIN` is a definer's rights package, and EXECUTE is automatically granted to DBA, IMP_FULL_DATABASE and DATAPUMP_IMP_FULL_DATABASE.

### 30.3 DBMS_AUTO_TASK_ADMIN Constants

The `DBMS_AUTO_TASK_ADMIN` package defines several constants that can be used for specifying parameter values.

These constants shown in the following table:
### Table 30-1  DBMS_AUTO_TASK_ADMIN Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY_MEDIUM</td>
<td>VARCHAR2</td>
<td>'MEDIUM'</td>
<td>Task with this priority should be executed as time permits</td>
</tr>
<tr>
<td>PRIORITY_HIGH</td>
<td>VARCHAR2</td>
<td>'HIGH'</td>
<td>Task with this priority should be executed within the current Maintenance Window</td>
</tr>
<tr>
<td>PRIORITY_URGENT</td>
<td>VARCHAR2</td>
<td>'URGENT'</td>
<td>Task with this priority is to be executed at the earliest opportunity</td>
</tr>
</tbody>
</table>

### 30.4 Summary of DBMS_AUTO_TASK_ADMIN Subprograms

This table lists the DBMS_AUTO_TASK_ADMIN subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE Procedures</td>
<td>Prevents AUTOTASK from executing any requests from a specified client or operation.</td>
</tr>
<tr>
<td>ENABLE Procedures</td>
<td>Allows a previously disabled client, operation, target type, or individual target to be enabled under AUTOTASK control</td>
</tr>
<tr>
<td>GET_CLIENT_ATTRIBUTES Procedure</td>
<td>Returns values of select client attributes</td>
</tr>
<tr>
<td>GET_P1_RESOURCES Procedure</td>
<td>Returns percent of resources allocated to each AUTOTASK High Priority Consumer Groups</td>
</tr>
<tr>
<td>OVERRIDE_PRIORITY Procedures</td>
<td>Manually overrides task priority.</td>
</tr>
<tr>
<td>SET_CLIENT_SERVICE Procedure</td>
<td>Associates an AUTOTASK Client with a specified Service</td>
</tr>
<tr>
<td>SET_P1_RESOURCES Procedure</td>
<td>Sets percentage-based resource allocation for each High Priority Consumer Group used by AUTOTASK Clients</td>
</tr>
</tbody>
</table>

### 30.4.1 DISABLE Procedures

This procedure prevents AUTOTASK from executing any requests from a specified client or operation.

**Syntax**

Disables all AUTOTASK functionality.

```sql
DBMS_AUTO_TASK_ADMIN.DISABLE;
```

Disables all tasks for the client or operation.

```sql
DBMS_AUTO_TASK_ADMIN.DISABLE (
    client_name IN VARCHAR2,
```
operation       IN VARCHAR2,
window_name     IN VARCHAR2);

Parameters

Table 30-3  DISABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_name</td>
<td>Name of the client, as found in DBA_AUTOTASK_CLIENT View</td>
</tr>
<tr>
<td>operation</td>
<td>Name of the operation as specified in DBA_AUTOTASK_OPERATION View</td>
</tr>
<tr>
<td>window_name</td>
<td>Optional name of the window in which client is to be disabled</td>
</tr>
</tbody>
</table>

Usage Notes

- If operation and window_name are both NULL, the client is disabled.
- If operation is not NULL, window_name is ignored and the operation is disabled.
- If operation is NULL and window_name is not NULL, the client is disabled in the specified window.

30.4.2 ENABLE Procedures

This procedure allows a previously disabled client, operation, target type, or individual target to be enabled under AUTOTASK control.

Specifying the DEFERRED option postpones the effect of the call until the start of the next maintenance window. If IMMEDIATE option is specified the effect of this call is immediate – as long as there is a currently open maintenance window.

Syntax

Re-enabling AUTOTASK. This version enables the specified client. Note that any explicitly disabled tasks or operations must be re-enabled individually.

DBMS_AUTO_TASK_ADMIN.ENABLE;

Re-enabling a client or operation. Note that any explicitly disabled tasks or operations must be re-enabled individually.

DBMS_AUTO_TASK_ADMIN.ENABLE (  
  client_name  IN VARCHAR2,  
  operation    IN VARCHAR2,  
  window_name  IN VARCHAR2);  

Parameters

Table 30-4  ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_name</td>
<td>Name of the client, as found in DBA_AUTOTASK_CLIENT View</td>
</tr>
<tr>
<td>operation</td>
<td>Name of the operation as specified in DBA_AUTOTASK_OPERATION View</td>
</tr>
</tbody>
</table>
Table 30-4  (Cont.) ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_name</td>
<td>Optional name of the window in which client is to be enabled</td>
</tr>
</tbody>
</table>

Usage Notes

- If operation and window_name are both NULL, the client is enabled.
- If operation is not NULL, window_name is ignored and the specified operation is enabled
- If operation is NULL and window_name is not NULL, the client is enabled in the specified window.

30.4.3 GET_CLIENT_ATTRIBUTES Procedure

This procedure returns values of select client attributes.

Syntax

```
DBMS_AUTO_TASK_ADMIN.GET_CLIENT_ATTRIBUTES(
    client_name        IN    VARCHAR2,   service_name       OUT   VARCHAR2,
    window_group       OUT   VARCHAR2);
```

Parameters

Table 30-5  GET_CLIENT_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_name</td>
<td>Name of the client, as found in DBA_AUTOTASK_CLIENT View</td>
</tr>
<tr>
<td>service_name</td>
<td>Service name for client, may be NULL</td>
</tr>
<tr>
<td>window_group</td>
<td>Name of the window group in which the client is active</td>
</tr>
</tbody>
</table>

30.4.4 GET_P1_RESOURCES Procedure

This procedure returns percent of resources allocated to each AUTOTASK High Priority Consumer Group.

Syntax

```
DBMS_AUTO_TASK_ADMIN.GET_P1_RESOURCES(
    stats_group_pct     OUT   NUMBER,
    seg_group_pct       OUT   NUMBER,
    tune_group_pct      OUT   NUMBER,
    health_group_pct    OUT   NUMBER);
```
Parameters

Table 30-6  GET_P1_RESOURCES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stats_group_pct</td>
<td>%resources for Statistics Gathering</td>
</tr>
<tr>
<td>seq_group_pct</td>
<td>%resources for Space Management</td>
</tr>
<tr>
<td>tune_group_pct</td>
<td>%resources for SQL Tuning</td>
</tr>
<tr>
<td>health_group_pct</td>
<td>%resources for Health Checks</td>
</tr>
</tbody>
</table>

Usage Notes

Values will add up to 100%.

30.4.5 OVERRIDE_PRIORITY Procedures

This deprecated procedure is used to manually override task priority.

Note:

This subprogram is deprecated and becomes nonoperative with Oracle Database 12c.

This can be done at the client, operation or individual task level. This priority assignment is honored during the next maintenance window in which the named client is active. Specifically, setting the priority to URGENT causes a high priority job to be generated at the start of the maintenance window. Setting priority to CLEAR removes the override.

Syntax

Override Priority for a Client.

DBMS_AUTO_TASK_ADMIN.OVERRIDE_PRIORITY (
  client_name       IN    VARCHAR2,
  priority          IN    VARCHAR2);

Override Priority for an Operation.

DBMS_AUTO_TASK_ADMIN.OVERRIDE_PRIORITY (
  client_name       IN    VARCHAR2,
  operation         IN    VARCHAR2,
  priority          IN    VARCHAR2);

Override Priority for a Task.

DBMS_AUTO_TASK_ADMIN.OVERRIDE_PRIORITY (
  client_name       IN    VARCHAR2,
  operation         IN    VARCHAR2,
  task_target_type  IN    VARCHAR2,
  task_target_name  IN    VARCHAR2,
  priority          IN    VARCHAR2);
Parameters

Table 30-7  OVERRIDE_PRIORITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_name</td>
<td>Name of the client, as found in DBA_AUTOTASK_CLIENT View</td>
</tr>
<tr>
<td>priority</td>
<td>URGENT, HIGH, MEDIUM or LOW</td>
</tr>
<tr>
<td>operation</td>
<td>Name of the operation as specified in DBA_AUTOTASK_OPERATION View</td>
</tr>
<tr>
<td>task_target_type</td>
<td>Type of target to be affected, as found in V$AUTOTASK_TARGET_TYPE View</td>
</tr>
<tr>
<td>task_target_name</td>
<td>Name of the specific target to be affected</td>
</tr>
</tbody>
</table>

30.4.6 SET_CLIENT_SERVICE Procedure

This procedure associates an AUTOTASK Client with a specified Service.

Syntax

```sql
DBMS_AUTO_TASK_ADMIN.SET_CLIENT_SERVICE(
    client_name        IN    VARCHAR2,
    service_name       IN   VARCHAR2);
```

Parameters

Table 30-8  SET_CLIENT_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_name</td>
<td>Name of the client, as found in DBA_AUTOTASK_CLIENT View</td>
</tr>
<tr>
<td>service_name</td>
<td>Service name for client, may be NULL</td>
</tr>
</tbody>
</table>

Usage Notes

All work performed on behalf of the Client takes place only on instances where the service is enabled.

30.4.7 SET_P1_RESOURCES Procedure

This procedure sets percentage-based resource allocation for each High Priority Consumer Group used by AUTOTASK Clients.

Syntax

```sql
DBMS_AUTO_TASK_ADMIN.SET_P1_RESOURCES(
    stats_group_pct     OUT   NUMBER,
    seg_group_pct       OUT   NUMBER,
    tune_group_pct      OUT   NUMBER,
    health_group_pct    OUT   NUMBER);
```
Parameters

Table 30-9  SET_P1_RESOURCES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stats_group_pct</td>
<td>%resources for Statistics Gathering</td>
</tr>
<tr>
<td>seq_group_pct</td>
<td>%resources for Space Management</td>
</tr>
<tr>
<td>tune_group_pct</td>
<td>%resources for SQL Tuning</td>
</tr>
<tr>
<td>health_group_pct</td>
<td>%resources for Health Checks</td>
</tr>
</tbody>
</table>

Usage Notes

Values must be integers in the range 0 to 100, and must add up to 100 (percent), otherwise, an exception is raised.
DBMS_AW_STATS contains subprograms for managing optimizer statistics for cubes and dimensions. Generating the statistics does not have a significant performance cost.

See Also:
Oracle OLAP User's Guide regarding use of the OLAP option to support business intelligence and analytical applications

This chapter contains the following topics:

• Using DBMS_AW_STATS
• Summary of DBMS_AW_STATS Subprograms

31.1 Using DBMS_AW_STATS

Cubes and dimensions are first class data objects that support multidimensional analytics. They are stored in a container called an analytic workspace. Multidimensional objects and analytics are available with the OLAP option to Oracle Database.

Optimizer statistics are used to create execution plans for queries that join two cube views or join a cube view to a table or a view of a table. They are also used for query rewrite to cube materialized views. You need to generate the statistics only for these types of queries.

Queries against a single cube do not use optimizer statistics. These queries are automatically optimized within the analytic workspace.

31.2 Summary of DBMS_AW_STATS Subprograms

DBMS_AW_STATS uses the ANALYZE and CLEAR procedures.

**Table 31-1  DBMS_AW_STATS Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZE Procedure</td>
<td>Generates optimizer statistics on cubes and cube dimensions.</td>
</tr>
<tr>
<td>CLEAR Procedure</td>
<td>Clears optimizer statistics from cubes and cube dimensions.</td>
</tr>
</tbody>
</table>

31.2.1 ANALYZE Procedure

This procedure generates optimizer statistics on a cube or a cube dimension.

These statistics are used to generate some execution plans, as described in "Using DBMS_AW_STATS".
For a cube, the statistics are for all of the measures and calculated measures associated with the cube. These statistics include:

- The average length of data values
- The length of the largest data value
- The minimum value
- The number of distinct values
- The number of null values

For a dimension, the statistics are for the dimension and its attributes, levels, and hierarchies. These statistics include:

- The average length of a value
- The length of the largest value
- The minimum value
- The maximum value

**Syntax**

```sql
DBMS_AW_STATS.ANALYZE
    (inname IN VARCHAR2);
```

**Parameters**

**Table 31-2  ANALYZE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inname</td>
<td>The qualified name of a cube or a dimension. For a cube, the format of a qualified name is <code>owner.cube_name</code>. For a dimension, the format is <code>owner.dimension_name</code>.</td>
</tr>
</tbody>
</table>

**Usage Notes**

Always analyze the dimensions first, then the cube.

After analyzing a dimension, analyze all cubes that use that dimension.

**Example**

This sample script generates optimizer statistics on `UNITS_CUBE` and its dimensions.

```sql
BEGIN
    DBMS_AW_STATS.ANALYZE('time');
    DBMS_AW_STATS.ANALYZE('customer');
    DBMS_AW_STATS.ANALYZE('product');
    DBMS_AW_STATS.ANALYZE('channel');
    DBMS_AW_STATS.ANALYZE('units_cube');
END;
/
```

The following statements create and display an execution plan for a `SELECT` statement that joins columns from `UNITS_CUBE_VIEW`, `CUSTOMER_PRIMARY_VIEW`, and the `ACCOUNTS` table:
EXPLAIN PLAN FOR SELECT
    cu.long_description customer,
    a.city city,
    a.zip_pc zip,
    cu.level_name "LEVEL",
    round(f.sales) sales
/* From dimension views and cube view */
FROM time_calendar_view t,
    product_primary_view p,
    customer_view cu,
    channel_view ch,
    units_cube_view f,
    account a
/* Create level filters instead of GROUP BY */
WHERE t.long_description = '2004'
    AND p.level_name = 'TOTAL'
    AND cu.customer_account_id like 'COMP%' 
    AND ch.level_name = 'TOTAL'
/* Join dimension views to cube view */
    AND t.dim_key = f.TIME 
    AND p.dim_key = f.product 
    AND cu.dim_key = f.customer 
    AND ch.dim_key = f.channel 
    AND a.account_id = cu.customer_account_id
ORDER BY zip;

SQL> SELECT plan_table_output FROM TABLE(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
Plan hash value: 3890178023

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>89</td>
<td>6 (34)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>SORT ORDER BY</td>
<td></td>
<td>1</td>
<td>89</td>
<td>6 (34)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td>89</td>
<td>5 (20)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>JOINED CUBE SCAN PARTIAL OUTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CUBE ACCESS</td>
<td>UNITS_CUBE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CUBE ACCESS</td>
<td>CHANNEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>CUBE ACCESS</td>
<td>CUSTOMER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CUBE ACCESS</td>
<td>PRODUCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 8</td>
<td>CUBE ACCESS</td>
<td>TIME</td>
<td>1</td>
<td>55</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 9</td>
<td>TABLE ACCESS FULL</td>
<td>ACCOUNT</td>
<td>3</td>
<td>102</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
---------------------------------------------------
2 - access("A","ACCOUNT_ID"=SYS_OP_ATG(VALUE(KOKBF$),39,40,2))
8 - filter(SYS_OP_ATG(VALUE(KOKBF$),16,17,2)='2004' AND
    SYS_OP_ATG(VALUE(KOKBF$),39,40,2) LIKE 'COMP%' AND
    SYS_OP_ATG(VALUE(KOKBF$),47,48,2)='TOTAL' AND
    SYS_OP_ATG(VALUE(KOKBF$),25,26,2)='TOTAL')
9 - filter("A","ACCOUNT_ID" LIKE 'COMP%')

Note
-----
- dynamic statistics used for this statement
31.2.2 CLEAR Procedure

This procedure clears the statistics generated by the ANALYZE procedure.

Syntax

DBMS_AW_STATS.CLEAR (
   inname IN VARCHAR2;
)

Parameters

Table 31-3 CLEAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inname</td>
<td>The qualified name of a cube or a dimension. For a cube, the format of a qualified name is owner.cube_name. For a dimension, the format is owner.dimension_name.</td>
</tr>
</tbody>
</table>

Examples

The following scripts clears the statistics from UNITS_CUBE and its dimensions.

BEGIN
   DBMS_AW_STATS.clear('units_cube');
   DBMS_AW_STATS.clear('time');
   DBMS_AW_STATS.clear('customer');
   DBMS_AW_STATS.clear('product');
   DBMS_AW_STATS.clear('channel');
END;
/

The DBMS_CAPTURE_ADM package, one of a set of Oracle Streams packages, provides subprograms for starting, stopping, and configuring a capture process. The source of the captured changes is the redo logs, and the repository for the captured changes is a queue.

See Also:

Oracle Streams Concepts and Administration and Oracle Streams Replication Administrator’s Guide for more information about this package and capture processes.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_CAPTURE_ADM Subprograms

32.1 DBMS_CAPTURE_ADM Overview

This package provides interfaces to start, stop, and configure a capture process or a synchronous capture. This package includes subprograms for preparing database objects for instantiation.

Capture processes can be used in an XStream configuration in a multitenant container database (CDB). A CDB is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs).

See Also:

Oracle Database Concepts for more information about CDBs and PDBs.

32.2 DBMS_CAPTURE_ADM Security Model

The DBMS_CAPTURE_ADM security can be controlled in one of two ways.

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.
When the `DBMS_CAPTURE_ADM` package is used to manage an Oracle Streams configuration, it requires that the user is granted the privileges of an Oracle Streams administrator.

When the `DBMS_CAPTURE_ADM` package is used to manage an XStream configuration, it requires that the user is granted the privileges of an XStream administrator.

**Note:**

- The user must be granted additional privileges to perform some administrative tasks using the subprograms in this package, such as setting a capture user. If additional privileges are required for a subprogram, then the privileges are documented in the section that describes the subprogram.
- Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

**See Also:**

- *Oracle Streams Concepts and Administration* for information about configuring an Oracle Streams administrator
- *Oracle Database XStream Guide* for information about configuring an XStream administrator

### 32.3 Summary of DBMS_CAPTURE_ADM Subprograms

This table lists the `DBMS_CAPTURE_ADM` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT_GLOBAL_INSTANTIATION Procedure</td>
<td>Reverses the effects of running the PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and PREPARE_TABLE_INSTANTIATION procedures</td>
</tr>
<tr>
<td>ABORT_SCHEMA_INSTANTIATION Procedure</td>
<td>Reverses the effects of running the PREPARE_SCHEMA_INSTANTIATION and PREPARE_TABLE_INSTANTIATION procedures</td>
</tr>
<tr>
<td>ABORT_SYNC_INSTANTIATION Procedure</td>
<td>Reverses the effects of running the PREPARE_SYNC_INSTANTIATION procedure</td>
</tr>
<tr>
<td>ABORT_TABLE_INSTANTIATION Procedure</td>
<td>Reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure</td>
</tr>
<tr>
<td>ALTER_CAPTURE Procedure</td>
<td>Alters a capture process</td>
</tr>
<tr>
<td>ALTER_SYNC_CAPTURE Procedure</td>
<td>Alters a synchronous capture</td>
</tr>
</tbody>
</table>
### Table 32-1  (Cont.) DBMS_CAPTURE_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BUILD Procedure</strong></td>
<td>Extracts the data dictionary of the current database to the redo logs and automatically specifies database supplemental logging for all primary key and unique key columns</td>
</tr>
<tr>
<td><strong>CREATE_CAPTURE Procedure</strong></td>
<td>Creates a capture process</td>
</tr>
<tr>
<td><strong>CREATE_SYNC_CAPTURE Procedure</strong></td>
<td>Creates a synchronous capture</td>
</tr>
<tr>
<td><strong>DROP_CAPTURE Procedure</strong></td>
<td>Drops a capture process</td>
</tr>
<tr>
<td><strong>INCLUDE_EXTRA_ATTRIBUTE Procedure</strong></td>
<td>Includes or excludes an extra attribute in logical change records (LCRs) captured by the specified capture process or synchronous capture</td>
</tr>
<tr>
<td><strong>PREPARE_GLOBAL_INSTANTIATION Procedure</strong></td>
<td>Performs the synchronization necessary for instantiating all the tables in the database at another database and can enable supplemental logging for key columns or all columns in these tables</td>
</tr>
<tr>
<td><strong>PREPARE_SCHEMA_INSTANTIATION Procedure</strong></td>
<td>Performs the synchronization necessary for instantiating all tables in the schema at another database and can enable supplemental logging for key columns or all columns in these tables</td>
</tr>
<tr>
<td><strong>PREPARE_SYNC_INSTANTIATION Function</strong></td>
<td>Performs the synchronization necessary for instantiating one or more tables at another database and returns the prepare SCN</td>
</tr>
<tr>
<td><strong>PREPARE_TABLE_INSTANTIATION Procedure</strong></td>
<td>Performs the synchronization necessary for instantiating the table at another database and can enable supplemental logging for key columns or all columns in the table</td>
</tr>
<tr>
<td><strong>SET_PARAMETER Procedure</strong></td>
<td>Sets a capture process parameter to the specified value</td>
</tr>
<tr>
<td><strong>START_CAPTURE Procedure</strong></td>
<td>Starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue</td>
</tr>
<tr>
<td><strong>STOP_CAPTURE Procedure</strong></td>
<td>Stops the capture process from mining redo logs</td>
</tr>
</tbody>
</table>

**Note:**

All subprograms commit unless specified otherwise.

### 32.3.1 ABORT_GLOBAL_INSTANTIATION Procedure

This procedure reverses the effects of running the `PREPARE_GLOBAL_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, and `PREPARE_TABLE_INSTANTIATION` procedures.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to the database, schema, and table instantiations
• Removes any supplemental logging enabled by the
  PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and
  PREPARE_TABLE_INSTANTIATION procedures

Syntax

DBMS_CAPTURE_ADM.ABORT_GLOBAL_INSTANTIATION(
    container IN VARCHAR2 DEFAULT 'CURRENT');

Parameter

Table 32-2  ABORT_GLOBAL_INSTANTIATION Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| container | Either CURRENT, ALL, or pdb_name.  
  If CURRENT is specified, then this procedure removes supplemental logging for the current container.  
  If ALL is specified, then this procedure removes supplemental logging for all of the containers in the current CDB.  
  If pdb_name is specified, then this procedure removes supplemental logging for the specified PDB.  
  ALL and pdb_name are valid only when you invoke the procedure from the root. |

32.3.2 ABORT_SCHEMA_INSTANTIATION Procedure

This procedure reverses the effects of running the PREPARE_SCHEMA_INSTANTIATION procedure. It also reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure on tables in the specified schema.

Specifically, this procedure performs the following actions:

• Removes data dictionary information related to schema instantiations and table instantiations of tables in the schema

• Removes any supplemental logging enabled by the PREPARE_SCHEMA_INSTANTIATION procedure

• Removes any supplemental logging enabled by the PREPARE_TABLE_INSTANTIATION procedure for tables in the specified schema

Syntax

DBMS_CAPTURE_ADM.ABORT_SCHEMA_INSTANTIATION(
    schema_name IN VARCHAR2,
    container IN VARCHAR2 DEFAULT 'CURRENT');

Parameter

Table 32-3  ABORT_SCHEMA_INSTANTIATION Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema for which to abort the effects of preparing instantiation</td>
</tr>
</tbody>
</table>
Table 32-3  (Cont.) ABORT_SCHEMA_INSTANTIATION Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Either CURRENT, ALL, or pdb_name. If CURRENT is specified, then this procedure removes supplemental logging for the current container. If ALL is specified, then this procedure removes supplemental logging for all of the containers in the current CDB. If pdb_name is specified, then this procedure removes supplemental logging for the specified PDB. ALL and pdb_name are valid only when you invoke the procedure from the root.</td>
</tr>
</tbody>
</table>

32.3.3 ABORT_SYNC_INSTANTIATION Procedure

This procedure reverses the effects of running the PREPARE_SYNC_INSTANTIATION procedure. Specifically, this procedure removes data dictionary information related to the table instantiation.

This procedure is overloaded. The table_names parameter is VARCHAR2 datatype in one version and DBMS_UTILITY.UNCL_ARRAY datatype in the other version.

Syntax

```sql
DBMS_CAPTURE_ADM.ABORT_SYNC_INSTANTIATION(
  table_names IN VARCHAR2);
```

```sql
DBMS_CAPTURE_ADM.ABORT_SYNC_INSTANTIATION(
  table_names IN DBMS_UTILITY.UNCL_ARRAY);
```

Parameters

Table 32-4  ABORT_SYNC_INSTANTIATION Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_names</td>
<td>When the table_names parameter is VARCHAR2 datatype, a comma-delimited list of the tables for which to abort the effects of preparing instantiation. There must be no spaces between entries. When the table_names parameter is DBMS_UTILITY.UNCL_ARRAY datatype, specify a PL/SQL associative array of this type that contains the names of the tables for which to abort the effects of preparing instantiation. The first table name is at position 1, the second at position 2, and so on. The table does not need to be NULL terminated. In either version of the procedure, specify the name of each table in the form [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
32.3.4 ABORT_TABLE_INSTANTIATION Procedure

This procedure reverses the effects of running the PREPARE_TABLE_INSTANTIATION procedure.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to the table instantiation
- Removes any supplemental logging enabled by the PREPARE_TABLE_INSTANTIATION procedure

Syntax

```
DBMS_CAPTURE_ADM.ABORT_TABLE_INSTANTIATION(
    table_name  IN  VARCHAR2;
    container   IN  VARCHAR2  DEFAULT 'CURRENT');
```

Parameter

**Table 32-5  ABORT_TABLE_INSTANTIATION Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table for which to abort the effects of preparing instantiation, specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>container</td>
<td>Either CURRENT, ALL, or pdb_name. If CURRENT is specified, then this procedure removes supplemental logging for the current container. If ALL is specified, then this procedure removes supplemental logging for all of the containers in the current CDB. If pdb_name is specified, then this procedure removes supplemental logging for the specified PDB. ALL and pdb_name are valid only when you invoke the procedure from the root.</td>
</tr>
</tbody>
</table>

32.3.5 ALTER_CAPTURE Procedure

This procedure alters a capture process.

**See Also:**

*Oracle Streams Concepts and Administration* for more information about altering a capture process

Syntax

```
DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name               IN  VARCHAR2,
    rule_set_name              IN  VARCHAR2   DEFAULT NULL,
    remove_rule_set            IN  BOOLEAN    DEFAULT FALSE,
)
Parameters

Table 32-6  ALTER_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the capture process being altered. You must specify an existing capture process name. Do not specify an owner.</td>
</tr>
<tr>
<td>rule_set_name</td>
<td>The name of the positive rule set for the capture process. The positive rule set contains the rules that instruct the capture process to capture changes.</td>
</tr>
<tr>
<td></td>
<td>To change the positive rule set for the capture process, specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the hr schema named job_capture_rules, enter hr.job_capture_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</td>
</tr>
<tr>
<td>remove_rule_set</td>
<td>If TRUE, then the procedure removes the positive rule set for the specified capture process. If you remove a positive rule set for a capture process, and the capture process does not have a negative rule set, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If you remove a positive rule set for a capture process, and the capture process has a negative rule set, then the capture process captures all supported changes that are not discarded by the negative rule set. If FALSE, then the procedure retains the positive rule set for the specified capture process. If the rule_set_name parameter is non-NULL, then ensure that this parameter is set to FALSE.</td>
</tr>
</tbody>
</table>

See Also: Oracle Streams Concepts and Administration for more information about the changes that can be captured by a capture process
Table 32-6  (Cont.) ALTER_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_scn</td>
<td>A valid SCN for the database from which the capture process starts capturing changes. The SCN value must be greater than or equal to the first SCN for the capture process. Also, the capture process must be stopped before resetting its start SCN. An error is returned if an invalid SCN is specified or if the capture process is enabled.</td>
</tr>
<tr>
<td>use_database_link</td>
<td>If <strong>TRUE</strong>, then the capture process at a downstream database uses a database link to the source database for administrative purposes relating to the capture process. If you want a capture process that is not using a database link currently to begin using a database link, then specify <strong>TRUE</strong>. In this case, a database link with the same name as the global name of the source database must exist at the downstream database. If <strong>FALSE</strong>, then either the capture process is running on the source database, or the capture process at a downstream database does not use a database link to the source database. If you want a capture process that is using a database link currently to stop using a database link, then specify <strong>FALSE</strong>. In this case, you must prepare source database objects for instantiation manually when you add or change capture process rules that pertain to these objects. If <strong>NULL</strong>, then the current value of this parameter for the capture process is not changed.</td>
</tr>
<tr>
<td>first_scn</td>
<td>The lowest SCN in the redo log from which a capture process can capture changes. If you specify a new first SCN for the capture process, then the specified first SCN must meet the following requirements:</td>
</tr>
<tr>
<td></td>
<td>• It must be greater than the current first SCN for the capture process.</td>
</tr>
<tr>
<td></td>
<td>• It must be less than or equal to the current applied SCN for the capture process. However, this requirement does not apply if the current applied SCN for the capture process is zero.</td>
</tr>
<tr>
<td></td>
<td>• It must be less than or equal to the required checkpoint SCN for the capture process.</td>
</tr>
</tbody>
</table>

An error is returned if the specified SCN does not meet the first three requirements. See "Usage Notes" for information about determining an SCN value that meets all of these conditions.

When the first SCN is modified, the capture process purges information from its LogMiner data dictionary that is required to restart it at an earlier SCN. See **BUILD Procedure** for more information about a LogMiner data dictionary.

If the specified first SCN is higher than the current start SCN for the capture process, then the start SCN is set automatically to the new value of the first SCN.
Table 32-6  (Cont.) ALTER_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>negative_rule_set_name</td>
<td>The name of the negative rule set for the capture process. The negative rule set contains the rules that instruct the capture process to discard changes. To change the negative rule set for the capture process, specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a negative rule set in the hr schema named neg_capture_rules, enter hr.neg_capture_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL and the remove_negative_rule_set parameter is set to FALSE, then the procedure retains any existing negative rule set. If you specify NULL and the remove_negative_rule_set parameter is set to TRUE, then the procedure removes any existing negative rule set. If you specify both a positive and a negative rule set for a capture process, then the negative rule set is always evaluated first. If the negative_rule_set_name parameter is non-NULL, then ensure that this parameter is set to FALSE.</td>
</tr>
<tr>
<td>remove_negative_rule_set</td>
<td>If TRUE, then the procedure removes the negative rule set for the specified capture process. If you remove a negative rule set for a capture process, and the capture process does not have a positive rule set, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If you remove a negative rule set for a capture process, and a positive rule set exists for the capture process, then the capture process captures all changes that are not discarded by the positive rule set. If FALSE, then the procedure retains the negative rule set for the specified capture process. If the negative_rule_set_name parameter is non-NULL, then ensure that this parameter is set to FALSE.</td>
</tr>
</tbody>
</table>
Table 32-6  (Cont.) ALTER_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| capture_user       | The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If NULL, then the capture user is not changed. To change the capture user, the user who invokes the ALTER_CAPTURE procedure must be granted the DBA role. Only the SYS user can set the capture_user to SYS. If you change the capture user, then this procedure grants the new capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user of the queue. In addition, ensure that the capture user has the following privileges:  
  - EXECUTE privilege on the rule sets used by the capture process  
  - EXECUTE privilege on all rule-based transformation functions used in the rule set  
These privileges can be granted directly to the capture user, or they can be granted through roles. In addition, the capture user must be granted EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the capture process. These privileges must be granted directly to the capture user. They cannot be granted through roles. The capture process is stopped and restarted automatically when you change the value of this parameter.  
**Note:** If the capture user for a capture process is dropped using DROP USER . . . CASCADE, then the capture process is also dropped automatically.  
| checkpoint_retention_time | Either the number of days that a capture process retains checkpoints before purging them automatically, or DBMS_CAPTURE_ADM.INFINITE if checkpoints should not be purged automatically. If NULL, then the checkpoint retention time is not changed.  
If a number is specified, then a capture process purges a checkpoint the specified number of days after the checkpoint was taken. Partial days can be specified using decimal values. For example, .25 specifies 6 hours.  
When a checkpoint is purged, LogMiner data dictionary information for the archived redo log file that corresponds to the checkpoint is purged, and the first scn of the capture process is reset to the SCN value corresponding to the first change in the next archived redo log file.  
**See Also:** Oracle Streams Concepts and Administration for more information about checkpoint retention time |
Table 32-6  (Cont.) ALTER_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_time</td>
<td>A valid time from which the capture process starts capturing changes. The capture process must be stopped before resetting its start time. An error is returned if an invalid time is specified or if the capture process is enabled. The start_scn and start_time parameters are mutually exclusive.</td>
</tr>
</tbody>
</table>

Usage Notes

If you want to alter the first SCN for a capture process, then the value specified must meet the conditions in the description for the first_scn parameter.

Examples

The following query determines the current first SCN, applied SCN, and required checkpoint SCN for each capture process in a database:

```
SELECT CAPTURE_NAME, FIRST_SCN, APPLIED_SCN, REQUIRED_CHECKPOINT_SCN
FROM DBA_CAPTURE;
```

32.3.6 ALTER_SYNC_CAPTURE Procedure

This procedure alters a synchronous capture.

See Also:

*Oracle Streams Concepts and Administration* for more information about altering a capture process

Syntax

```
DBMS_CAPTURE_ADM.ALTER_SYNC_CAPTURE(
  capture_name  IN VARCHAR2,
  rule_set_name IN VARCHAR2 DEFAULT NULL,
  capture_user  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 32-7  ALTER_SYNC_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the synchronous capture being altered. You must specify an existing synchronous capture name. Do not specify an owner.</td>
</tr>
</tbody>
</table>
Table 32-7  (Cont.) ALTER_SYNC_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_set_name</td>
<td>The name of the positive rule set for the synchronous capture. The positive rule set contains the rules that instruct the synchronous capture to capture changes. To change the rule set for the synchronous capture, specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the strmadmin schema named sync_cap_rules, enter strmadmin.sync_cap_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You must specify a rule set that was created using the DBMS_STREAMS_ADM package. If NULL, then the rule set is not changed. See Also: Oracle Streams Concepts and Administration for more information about the changes that can be captured by a synchronous capture.</td>
</tr>
<tr>
<td>capture_user</td>
<td>The user in whose security domain a synchronous capture captures changes that satisfy its rule set and runs custom rule-based transformations configured for synchronous capture rules. If NULL, then the capture user is not changed. To change the capture user, the user who invokes the ALTER_SYNC_CAPTURE procedure must be granted the DBA role. Only the SYS user can set the capture_user to SYS. If you change the capture user, then this procedure grants the new capture user enqueue privilege on the queue used by the synchronous capture and configures the user as a secure queue user of the queue. In addition, ensure that capture user has the following privileges:  † EXECUTE privilege on the rule sets used by the synchronous capture  † EXECUTE privilege on all rule-based transformation functions used in the rule set These privileges can be granted directly to the capture user, or they can be granted through roles. In addition, the capture user must be granted EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the synchronous capture. These privileges must be granted directly to the capture user. They cannot be granted through roles.</td>
</tr>
</tbody>
</table>

Usage Notes

If the capture user for a synchronous capture is dropped using DROP USER . . . CASCADE, then the synchronous capture is also dropped automatically.

32.3.7 BUILD Procedure

This procedure extracts the data dictionary of the current database to the redo log and automatically specifies database supplemental logging by running the SQL statement ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

This procedure is overloaded. One version of this procedure contains the OUT parameter first_scn, and the other does not.

Syntax

```
DBMS_CAPTURE_ADM.BUILD(
    first_scn OUT NUMBER);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_scn</td>
<td>Contains the lowest SCN value corresponding to the data dictionary extracted to the redo log that can be specified as a first SCN for a capture process</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- You can run this procedure multiple times at a source database.
- If you plan to capture changes originating at a source database with a capture process, then this procedure must be executed at the source database at least once. When the capture process is started, either at a local source database or at a downstream database, the capture process uses the extracted information in the redo log to create a LogMiner data dictionary.
- A LogMiner data dictionary is a separate data dictionary used by a capture process to determine the details of a change that it is capturing. The LogMiner data dictionary is necessary because the primary data dictionary of the source database might not be synchronized with the redo data being scanned by a capture process.
- After executing this procedure, you can query the FIRST_CHANGE# column of the V$ARCHIVED_LOG dynamic performance view where the DICTIONARY_BEGIN column is YES to determine the lowest SCN value for the database that can be specified as a first SCN for a capture process. The first SCN for a capture process is the lowest SCN in the redo log from which the capture process can capture changes. You can specify the first SCN for a capture process when you run the CREATE_CAPTURE or ALTER_CAPTURE procedure in the DBMS_CAPTURE_ADM package.
- In a CDB, the BUILD procedure must be executed from the root.

32.3.8 CREATE_CAPTURE Procedure

This procedure creates a capture process.

See Also:

- Oracle Streams Replication Administrator's Guide for more information about creating a capture process
- DBMS_RULE_ADM for more information about rules and rule sets

Syntax

```sql
DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name                IN  VARCHAR2,
```
Parameters

Table 32-9  CREATE_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the queue into which the capture process enqueues changes. You must specify an existing queue in the form <code>[schema_name.]queue_name</code>. For example, to specify a queue in the <code>hr</code> schema named <code>streams_queue</code>, enter <code>hr.streams_queue</code>. If the schema is not specified, then the current user is the default. <strong>Note:</strong> The <code>queue_name</code> setting cannot be altered after the capture process is created.</td>
</tr>
<tr>
<td>capture_name</td>
<td>The name of the capture process being created. A NULL specification is not allowed. Do not specify an owner. <strong>Note:</strong> The <code>capture_name</code> setting cannot be altered after the capture process is created.</td>
</tr>
</tbody>
</table>
Table 32-9  (Cont.) CREATE_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_set_name</td>
<td>The name of the positive rule set for the capture process. The positive rule set contains the rules that instruct the capture process to capture changes. If you want to use a positive rule set for the capture process, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the hr schema named job_capture_rules, enter hr.job_capture_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL, and no negative rule set is specified, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas. If you specify NULL, and a negative rule set exists for the capture process, then the capture process captures all changes that are not discarded by the negative rule set. <strong>See Also</strong>: Oracle Streams Concepts and Administration for more information about the changes that can be captured by a capture process.</td>
</tr>
<tr>
<td>start_scn</td>
<td>A valid SCN for the database from which the capture process starts capturing changes. An error is returned if an invalid SCN is specified. The start_scn and start_time parameters are mutually exclusive. <strong>See Also</strong>: &quot;Usage Notes&quot; for more information setting the start_scn parameter</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the changes to be captured originated. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. If NULL, or if the specified name is the same as the global name of the current database, then local capture is assumed and only the default values for use_database_link and first_scn can be specified.</td>
</tr>
</tbody>
</table>
Table 32-9  (Cont.) CREATE_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| use_database_link | If TRUE, then the capture process at a downstream database uses a database link to the source database for administrative purposes relating to the capture process. A database link with the same name as the global name of the source database must exist at the downstream database. The capture process uses the database link to prepare database objects for instantiation at the source database and run the \texttt{DBMS\_CAPTURE\_ADM.BUILD} procedure at the source database, if necessary. During the creation of a downstream capture process, if the \texttt{first\_scn} parameter is set to \texttt{NULL}, then the \texttt{use\_database\_link} parameter must be set to TRUE. Otherwise, an error is returned. If FALSE, then either the capture process is running on the source database, or the capture process at a downstream database does not use a database link to the source database. In this case, you must perform the following administrative tasks manually:  
- Run the \texttt{DBMS\_CAPTURE\_ADM.BUILD} procedure at the source database to extract the data dictionary at the source database to the redo log when a capture process is created.  
- Obtain the first SCN for the downstream capture process if the first SCN is not specified during capture process creation. The first SCN is needed to create and maintain a capture process.  
- Prepare source database objects for instantiation. |
<p>| first_scn         | The lowest SCN in the redo log from which a capture process can capture changes. A non-NULL value for this parameter is valid only if the \texttt{DBMS_CAPTURE_ADM.BUILD} procedure has been run at least once at the source database. You can query the \texttt{FIRST_CHANGE#} column of the \texttt{V\textbackslash$ARCHIVED_LOG} dynamic performance view where the \texttt{DICTIONARY_BEGIN} column is \texttt{YES} to determine whether the \texttt{DBMS_CAPTURE_ADM.BUILD} procedure has been run on a source database. Any of the values returned by such a query can be used as a \texttt{first_scn} value if the redo log containing that SCN value is still available. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information setting the \texttt{first_scn} parameter |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| logfile_assignment    | If *implicit*, which is the default, then the capture process at a downstream database scans all redo log files added by redo transport services or manually from the source database to the downstream database.<br>  
If *explicit*, then a redo log file is scanned by a capture process at a downstream database only if the capture process name is specified in the FOR logminer_session_name clause. If *explicit*, then the redo log file must be added manually to the downstream database, and redo transport services cannot be used to add redo log files to the capture process being created.<br>  
If you specify *explicit* for this parameter for a local capture process, then the local capture process cannot use the online redo log to find changes. In this case, the capture process must use the archived redo log.  
**See Also:** "Usage Notes" for information about adding redo log files manually. |
| negative_rule_set_name| The name of the negative rule set for the capture process. The negative rule set contains the rules that instruct the capture process to discard changes.<br>  
If you want to use a negative rule set for the capture process, then you must specify an existing rule set in the form [schema_name.].rule_set_name. For example, to specify a negative rule set in the hr schema named neg_capture_rules, enter hr.neg_capture_rules. If the schema is not specified, then the current user is the default.<br>  
If you specify NULL, and no positive rule set is specified, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas.<br>  
If you specify NULL, and a positive rule set exists for the capture process, then the capture process captures all changes that are not discarded by the positive rule set.<br>  
An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.<br>  
If you specify both a positive and a negative rule set for a capture process, then the negative rule set is always evaluated first. |
Table 32-9  (Cont.) CREATE_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_user</td>
<td>The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If NULL, then the user who runs the CREATE_CAPTURE procedure is used.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If the capture user for a capture process is dropped using DROP USER . . . CASCADE, then the capture process is also dropped automatically.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
<tr>
<td>checkpoint_retention_time</td>
<td>Either specify the number of days that a capture process retains checkpoints before purging them automatically, or specify DBMS_CAPTURE_ADM.INFINITE if checkpoints should not be purged automatically.</td>
</tr>
<tr>
<td></td>
<td>If a number is specified, then a capture process purges a checkpoint the specified number of days after the checkpoint was taken. Partial days can be specified using decimal values. For example, .25 specifies 6 hours.</td>
</tr>
<tr>
<td></td>
<td>When a checkpoint is purged, LogMiner data dictionary information for the archived redo log file that corresponds to the checkpoint is purged, and the first_scn of the capture process is reset to the SCN value corresponding to the first change in the next archived redo log file.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> Oracle Streams Concepts and Administration for more information about checkpoint retention time</td>
</tr>
<tr>
<td>start_time</td>
<td>A valid time from which the capture process starts capturing changes.</td>
</tr>
<tr>
<td></td>
<td>An error is returned if an invalid time is specified.</td>
</tr>
<tr>
<td></td>
<td>The start_scn and start_time parameters are mutually exclusive.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> &quot;Usage Notes&quot; for more information setting the start_time parameter</td>
</tr>
<tr>
<td>source_root_name</td>
<td>The global name of the root in the source CDB.</td>
</tr>
<tr>
<td></td>
<td>If you specify NULL, or if the specified name is the same as the global name of the current root, then local capture is assumed.</td>
</tr>
<tr>
<td></td>
<td>If not NULL, then remote capture is assumed and a condition is added to the generated rules to filter the LCRs based on the root in the source CDB.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> In a downstream capture configuration, if the capture database is CDB and the source database is a non-CDB, then specify the same value for source_root_name and source_database. The source_root_name parameter does not need to be specified for local capture for either a CDB or a non-CDB.</td>
</tr>
</tbody>
</table>
Table 32-9  (Cont.) CREATE_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_class</td>
<td>The valid values are 'Streams', 'XStream' or 'GoldenGate'.</td>
</tr>
<tr>
<td></td>
<td>If NULL is specified, then 'Streams' is assumed.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The capture_class parameter cannot be set to 'Streams' or NULL when the capture database is a CDB.</td>
</tr>
</tbody>
</table>

Usage Notes

Consider the following usage notes when you run this procedure:

- **DBA Role Requirement**
- **Capture User Requirements**
- **First SCN and Start SCN Settings**
- **Explicit Log File Assignment**

**DBA Role Requirement**

If the user who invokes this procedure is different from the user specified in the capture_user parameter, then the invoking user must be granted the **DBA** role. If the user who invokes this procedure is the same as the user specified in the capture_user parameter, then the **DBA** role is not required for the invoking user. Only the **SYS** user can set the capture_user to **SYS**.

**Capture User Requirements**

The capture_user parameter specifies the user who captures changes that satisfy the capture process rule sets. This user must have the necessary privileges to capture changes. This procedure grants the capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user of the queue.

In addition, ensure that the capture user has the following privileges:

- **EXECUTE** privilege on the rule sets used by the capture process
- **EXECUTE** privilege on all rule-based transformation functions used in the positive rule set

These privileges can be granted directly to the capture user, or they can be granted through roles.

In addition, the capture user must be granted **EXECUTE** privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the capture process. These privileges must be granted directly to the capture user. They cannot be granted through roles.
A capture user does not require privileges on a database object to capture changes to the database object. The capture process can pass these changes to a rule-based transformation function. Therefore, ensure that you consider security implications when you configure a capture process.

Creation of the first capture process in a database might take some time because the data dictionary is duplicated during this creation.

First SCN and Start SCN Settings

When you create a capture process using this procedure, you can specify the first SCN and start SCN for the capture process. A capture process scans the redo data from the first SCN or an existing capture process checkpoint forward, even if the start SCN is higher than the first SCN or the checkpoint SCN. In this case, the capture process does not capture any changes in the redo data before the start SCN. Oracle recommends that, at capture process creation time, the difference between the first SCN and start SCN be as small as possible to keep the amount of redo scanned by the capture process to a minimum.

When you specify the `start_time` parameter instead of the `start_scn` parameter, the `start_time` corresponds with a specific SCN. In this case, the information in this section also applies to the SCN that corresponds with the specified `start_time`.

In some cases, the behavior of the capture process is different depending on the settings of these SCN values and on whether the capture process is local or downstream.

The following table describes capture process behavior for SCN value settings:
<table>
<thead>
<tr>
<th>first_scn Setting</th>
<th>start_scn Setting</th>
<th>Capture Process Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-NULL</td>
<td>NULL</td>
<td>Local or Downstream</td>
<td>The new capture process is created at the local database with a new LogMiner session starting from the value specified for the first_scn parameter. The start SCN is set to the specified first SCN value automatically, and the new capture process does not capture changes that were made before this SCN. The BUILD procedure in the DBMS_CAPTURE_ADM package is not run automatically. This procedure must have been run at least once before on the source database, and the specified first SCN must correspond to the SCN value of a previous build that is still available in the redo log. When the new capture process is started for the first time, it creates a new LogMiner data dictionary using the data dictionary information in the redo log. If the BUILD procedure has not been run at least once on the source database, then an error is raised when the capture process is started. Capture process behavior is the same for a local capture process and a downstream capture process created with these SCN settings, except that a local capture process is created at the source database and a downstream capture process is created at the downstream database.</td>
</tr>
<tr>
<td>Non-NULL</td>
<td>Non-NULL</td>
<td>Local or Downstream</td>
<td>If the specified value for the start_scn parameter is greater than or equal to the specified value for the first_scn parameter, then the new capture process is created at the local database with a new LogMiner session starting from the specified first SCN. In this case, the new capture process does not capture changes that were made before the specified start SCN. If the specified value for the start_scn parameter is less than the specified value for the first_scn parameter, then an error is raised. The BUILD procedure in the DBMS_CAPTURE_ADM package is not run automatically. This procedure must have been called at least once before on the source database, and the specified first_scn must correspond to the SCN value of a previous build that is still available in the redo log. When the new capture process is started for the first time, it creates a new LogMiner data dictionary using the data dictionary information in the redo log. If the BUILD procedure has not been run at least once on the source database, then an error is raised. Capture process behavior is the same for a local capture process and a downstream capture process created with these SCN settings, except that a local capture process is created at the source database and a downstream capture process is created at the downstream database.</td>
</tr>
</tbody>
</table>
The new capture process creates a new LogMiner data dictionary if either one of the following conditions is true:

- There is no existing capture process for the local source database, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the database.
- There are existing capture processes, but none of the capture processes have taken a checkpoint yet, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the database.

In either of these cases, the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is run during capture process creation.

The new capture process uses the resulting build of the source data dictionary in the redo log to create a LogMiner data dictionary the first time it is started, and the first SCN corresponds to the SCN of the data dictionary build. If there are any in-flight transactions, then the `BUILD` procedure waits until these transactions commit before completing. An in-flight transaction is one that is active during capture process creation or a data dictionary build.

However, if there is at least one existing local capture process for the local source database that has taken a checkpoint, then the new capture process shares an existing LogMiner data dictionary with one or more of the existing capture processes. In this case, a capture process with a first SCN that is lower than or equal to the specified start SCN must have been started successfully at least once. Also, if there are any in-flight transactions, then the capture process is created after these transactions commit.

If there is no existing capture process for the local source database (or if no existing capture processes have taken a checkpoint yet), and the specified start SCN is less than the current SCN for the database, then an error is raised.
When the `CREATE_CAPTURE` procedure creates a downstream capture process, the `use_database_link` parameter must be set to `TRUE` when the `first_scn` parameter is set to `NULL`. Otherwise, an error is raised. The database link is used to obtain the current SCN of the source database.

The new capture process creates a new LogMiner data dictionary if either one of the following conditions is true:

- There is no existing capture process that captures changes to the source database at the downstream database, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the source database.
- There are existing capture processes that capture changes to the source database at the downstream database, but none of the capture processes have taken a checkpoint yet, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the source database.

In either of these cases, the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is run during capture process creation. The first time you start the new capture process, it uses the resulting build of the source data dictionary in the redo log files copied to the downstream database to create a LogMiner data dictionary. Here, the first SCN for the new capture process corresponds to the SCN of the data dictionary build. If there are any in-flight transactions, then the `BUILD` procedure waits until these transactions commit before completing.

However, if at least one existing capture process has taken a checkpoint and captures changes to the source database at the downstream database, then the new capture process shares an existing LogMiner data dictionary with one or more of these existing capture processes. In this case, one of these existing capture processes with a first SCN that is lower than or equal to the specified start SCN must have been started successfully at least once. Also, if there are any in-flight transactions, then the capture process is created after these transactions commit.

If there is no existing capture process that captures changes to the source database at the downstream database (or no existing capture process has taken a checkpoint), and the specified `start_scn` parameter value is less than the current SCN for the source database, then an error is raised.

The behavior is the same as setting the `first_scn` parameter to `NULL` and setting the `start_scn` parameter to the current SCN of the source database.

Note:

When you create a capture process using the `DBMS_STREAMS_ADM` package, both the first SCN and the start SCN are set to `NULL` during capture process creation.
Explicit Log File Assignment

If you specify `explicit` for the `logfile_assignment` parameter, then you add a redo log file manually to a downstream database using the following statement:

```
ALTER DATABASE REGISTER LOGICAL LOGFILE
  file_name FOR capture_process;
```

Here, `file_name` is the name of the redo log file being added and `capture_process` is the name of the capture process that will use the redo log file at the downstream database. The `capture_process` is equivalent to the `logminer_session_name` and must be specified. The redo log file must be present at the site running the downstream database. You must transfer this file manually to the site running the downstream database using the `DBMS_FILE_TRANSFER` package, FTP, or some other transfer method.

See Also:

Oracle Database SQL Language Reference for more information about the `ALTER DATABASE` statement and Oracle Data Guard Concepts and Administration for more information registering redo log files

32.3.9 CREATE_SYNC_CAPTURE Procedure

This procedure creates a synchronous capture.

See Also:

Oracle Streams Replication Administrator's Guide for more information about creating a synchronous capture

Syntax

```
DBMS_CAPTURE_ADM.CREATE_SYNC_CAPTURE(
  queue_name     IN  VARCHAR2,
  capture_name   IN  VARCHAR2,
  rule_set_name  IN  VARCHAR2,
  capture_user   IN  VARCHAR2  DEFAULT NULL);
```
Parameters

Table 32-10  CREATE_SYNC_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the queue into which the synchronous capture enqueues changes. You must specify an existing queue in the form [schema_name.]queue_name. For example, to specify a queue in the strmadmin schema named streams_queue, enter strmadmin.streams_queue. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>capture_name</td>
<td>The name of the synchronous capture being created. A NULL specification is not allowed. Do not specify an owner.</td>
</tr>
<tr>
<td>rule_set_name</td>
<td>The name of the positive rule set for the synchronous capture. The positive rule set contains the rules that instruct the synchronous capture to capture changes. Specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the strmadmin schema named sync_cap_rules, enter strmadmin.sync_cap_rules. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>capture_user</td>
<td>The user in whose security domain the synchronous capture captures changes that satisfy its rule set and runs custom rule-based transformations configured for synchronous capture rules. If NULL, then the user who runs the CREATE_SYNC_CAPTURE procedure is used. Only a user who is granted the DBA role can set a capture user. Only the SYS user can set the capture_user to SYS.</td>
</tr>
</tbody>
</table>

Usage Notes

When the CREATE_SYNC_CAPTURE procedure creates a synchronous capture, the procedure must obtain an exclusive lock on each table for which it will capture changes. The rules in the specified rule set for the synchronous capture determine these tables. If there are outstanding transactions on a table for which the synchronous capture will capture changes, then the procedure waits until it can obtain a lock.
The `capture_user` parameter specifies the user who captures changes that satisfy the synchronous capture rule set. This user must have the necessary privileges to capture changes.

In addition, ensure that the capture user has the following privileges:

- **ENQUEUE** privilege on the queue specified in the `queue_name` parameter
- **EXECUTE** privilege on the rule set used by the synchronous capture
- **EXECUTE** privilege on all rule-based transformation functions used in the rule set

These privileges can be granted directly to the capture user, or they can be granted through roles.

In addition, the capture user must be granted **EXECUTE** privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the synchronous capture. These privileges must be granted directly to the capture user. These privileges cannot be granted through roles.

### Note:

A capture user does not require privileges on a database object to capture changes to the database object. The synchronous capture can pass these changes to a rule-based transformation function. Therefore, ensure that you consider security implications when you configure a synchronous capture.

#### 32.3.10 DROP_CAPTURE Procedure

This procedure drops a capture process.

**Syntax**

```
DBMS_CAPTURE_ADM.DROP_CAPTURE(
    capture_name           IN  VARCHAR2,
    drop_unused_rule_sets  IN  BOOLEAN  DEFAULT FALSE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>capture_name</code></td>
<td>The name of the capture process being dropped. Specify an existing capture process name. Do not specify an owner.</td>
</tr>
<tr>
<td><code>drop_unused_rule_sets</code></td>
<td>If <strong>TRUE</strong>, then the procedure drops any rule sets, positive and negative, used by the specified capture process if these rule sets are not used by any other Oracle Streams client. Oracle Streams clients include capture processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set. If <strong>FALSE</strong>, then the procedure does not drop the rule sets used by the specified capture process, and the rule sets retain their rules.</td>
</tr>
</tbody>
</table>
Usage Notes

The following usage notes apply to this procedure:

- The Capture Process Must Be Stopped Before It Is Dropped
- The DROP_CAPTURE Procedure and Rules-related Information

The Capture Process Must Be Stopped Before It Is Dropped

A capture process must be stopped before it can be dropped.

See Also:

STOP_CAPTURE Procedure

The DROP_CAPTURE Procedure and Rules-related Information

When you use this procedure to drop a capture process, rules-related information for the capture process created by the DBMS_STREAMS_ADM package is removed from the data dictionary views for Oracle Streams rules. Information about such a rule is removed even if the rule is not in either rule set for the capture process.

The following are the data dictionary views for Oracle Streams rules:

- ALL_STREAMS_GLOBAL_RULES
- DBA_STREAMS_GLOBAL_RULES
- ALL_STREAMS_MESSAGE_RULES
- DBA_STREAMS_MESSAGE_RULES
- ALL_STREAMS_SCHEMA_RULES
- DBA_STREAMS_SCHEMA_RULES
- ALL_STREAMS_TABLE_RULES
- DBA_STREAMS_TABLE_RULES
- ALL_STREAMS_RULES
- DBA_STREAMS_RULES

See Also:

Oracle Streams Concepts and Administration for more information about Oracle Streams data dictionary views
32.3.11 INCLUDE_EXTRA_ATTRIBUTE Procedure

This procedure includes or excludes an extra attribute in logical change records (LCRs) captured by the specified capture process or synchronous capture.

Syntax

```sql
DBMS_CAPTURE_ADM.INCLUDE_EXTRA_ATTRIBUTE(
    capture_name    IN  VARCHAR2,
    attribute_name  IN  VARCHAR2,
    include         IN  BOOLEAN   DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the capture process or synchronous capture. Specify an existing capture process name or synchronous capture name. Do not specify an owner.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>The name of the attribute to be included in or excluded from LCRs captured by the capture process or synchronous capture. The following names are valid settings:</td>
</tr>
<tr>
<td></td>
<td>• row_id The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, or in row LCRs for index-organized tables. The type is VARCHAR2.</td>
</tr>
<tr>
<td></td>
<td>• serial# The serial number of the session that performed the change captured in the LCR. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>• session# The identifier of the session that performed the change captured in the LCR. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>• thread# The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>• tx_name The name of the transaction that includes the LCR. The type is VARCHAR2.</td>
</tr>
<tr>
<td></td>
<td>• username The name of the user who performed the change captured in the LCR. The type is VARCHAR2.</td>
</tr>
<tr>
<td>include</td>
<td>If TRUE, then the specified attribute is included in LCRs captured by the capture process or synchronous capture.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the specified attribute is excluded from LCRs captured by the capture process or synchronous capture.</td>
</tr>
</tbody>
</table>
Usage Notes

Some information is not captured by a capture process or synchronous capture unless you use this procedure to specify that the information should be captured. If you want to exclude an extra attribute that is being captured by a capture process or synchronous capture, then specify the attribute and specify FALSE for the include parameter.

32.3.12 PREPARE_GLOBAL_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating all the tables in the database at another database and can enable supplemental logging for key columns or all columns in these tables.

This procedure prepares the tables in the database for instantiation when a capture process will be used to capture changes to the tables in the database.

This procedure records the lowest SCN of each object in the database for instantiation. SCNs after the lowest SCN for an object can be used for instantiating the object. Running this procedure prepares all current and future objects in the database for instantiation.

See Also:

- Oracle Streams Replication Administrator’s Guide for more information about instantiation
- Oracle Streams Replication Administrator’s Guide for more information about supplemental logging

Syntax

```
DBMS_CAPTURE_ADM.PREPARE_GLOBAL_INSTANTIATION
  supplemental_logging IN VARCHAR2 DEFAULT 'KEYS',
  container IN VARCHAR2 DEFAULT 'CURRENT');
```
### Parameter

#### Table 32-13  PREPARE_GLOBAL_INSTANTIATION Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>supplemental_logging</td>
<td>Either NONE, KEYS, or ALL. If NONE is specified, then this procedure does not enable supplemental logging for any columns in the tables in the database. This procedure does not remove existing supplemental logging specifications for these tables. If KEYS is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the database and for any table added to the database in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally. Specifying KEYS does not enable supplemental logging of bitmap join index columns. If ALL is specified, then this procedure enables supplemental logging for all columns in the tables in the database and for any table added to the database in the future. The columns are logged unconditionally. Supplemental logging is not enabled for columns of the following types: LOB, LONG, LONG RAW, user-defined types, and Oracle-supplied types.</td>
</tr>
<tr>
<td>container</td>
<td>Either CURRENT, ALL, or pdb_name. If CURRENT is specified, then this procedure adds supplemental logging for the current container. If ALL is specified, then this procedure adds supplemental logging for all of the containers in the current CDB. If pdb_name is specified, then this procedure adds supplemental logging for the specified PDB. ALL and pdb_name are valid only when you invoke the procedure from the root.</td>
</tr>
</tbody>
</table>

### Usage Notes

Run this procedure at the source database.

If you use a capture process to capture all of the changes to a database, then use this procedure to prepare the tables in the database for instantiation after the capture process has been configured.

#### 32.3.13 PREPARE_SCHEMA_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating all tables in the schema at another database and can enable supplemental logging for key columns or all columns in these tables.

This procedure prepares the tables in the schema for instantiation when a capture process will be used to capture changes to the tables in the schema.

This procedure records the lowest SCN of each object in the schema for instantiation. SCNs after the lowest SCN for an object can be used for instantiating the object.
Running this procedure prepares all current and future objects in the schema for instantiation.

### See Also:
- *Oracle Streams Replication Administrator's Guide* for more information about instantiation
- *Oracle Streams Replication Administrator's Guide* for more information about supplemental logging

### Syntax

```sql
DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
    schema_name    IN  VARCHAR2,
    supplemental_logging  IN  VARCHAR2  DEFAULT 'KEYS',
    container    IN  VARCHAR2  DEFAULT 'CURRENT');
```

### Parameters

#### Table 32-14  PREPARE_SCHEMA_INSTANTIATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema. For example, hr.</td>
</tr>
<tr>
<td>supplemental_logging</td>
<td>Either NONE, KEYS, or ALL. If NONE is specified, then this procedure does not enable supplemental logging for any columns in the tables in the schema. This procedure does not remove existing supplemental logging specifications for these tables. If KEYS is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the schema and for any table added to this schema in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally. Specifying KEYS does not enable supplemental logging of bitmap join index columns. If ALL is specified, then this procedure enables supplemental logging for all columns in the tables in the schema and for any table added to this schema in the future. The columns are logged unconditionally. Supplemental logging is not enabled for columns of the following types: LOB, LONG, LONG RAW, user-defined types, and Oracle-supplied types.</td>
</tr>
<tr>
<td>container</td>
<td>Either CURRENT, ALL, or pdb_name. If CURRENT is specified, then this procedure adds supplemental logging for the current container. If ALL is specified, then this procedure adds supplemental logging for all of the containers in the current CDB. If pdb_name is specified, then this procedure adds supplemental logging for the specified PDB. ALL and pdb_name are valid only when you invoke the procedure from the root.</td>
</tr>
</tbody>
</table>
Usage Notes

Run this procedure at the source database. If you use a capture process to capture all of the changes to a schema, then use this procedure to prepare the tables in the schema for instantiation after the capture process has been configured.

32.3.14 PREPARE_SYNC_INSTANTIATION Function

This function performs the synchronization necessary for instantiating one or more tables at another database. This function returns the prepare system change number (SCN) for the table or tables being prepared for instantiation.

This function prepares one or more tables for instantiation when a synchronous capture will be used to capture changes to the tables.

This function records the lowest SCN of each table for instantiation (prepare SCN). SCNs after the lowest SCN for an object can be used for instantiating the object.

This function is overloaded. The table_names parameter is VARCHAR2 datatype in one version and DBMS_UTILITY.UNCL_ARRAY datatype in the other version.

See Also:

Oracle Streams Replication Administrator's Guide for more information about instantiation

Syntax

DBMS_CAPTURE_ADM.PREPARE_SYNC_INSTANTIATION(
   table_names  IN  VARCHAR2)
RETURN NUMBER;

DBMS_CAPTURE_ADM.PREPARE_SYNC_INSTANTIATION(
   table_names  IN  DBMS_UTILITY.UNCL_ARRAY)
RETURN NUMBER;

Parameters

Table 32-15 PREPARE_SYNC_INSTANTIATION Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| table_names | When the table_names parameter is VARCHAR2 datatype, a comma-delimited list of the tables to prepare for instantiation. There must be no spaces between entries.  
When the table_names parameter is DBMS_UTILITY.UNCL_ARRAY datatype, specify a PL/SQL associative array of this type that contains the names of the tables to prepare for instantiation. The first table name is at position 1, the second at position 2, and so on. The table does not need to be NULL terminated.  
In either version of the function, specify the name of each table in the form [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default. |
32.3.15 PREPARE_TABLE_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating the table at another database and can enable supplemental logging for key columns or all columns in the table.

This procedure prepares the table for instantiation when a capture process will be used to capture changes to the table.

This procedure records the lowest SCN of the table for instantiation. SCNs after the lowest SCN for an object can be used for instantiating the object.

See Also:

- Oracle Streams Replication Administrator's Guide for more information about instantiation
- Oracle Streams Replication Administrator's Guide for more information about supplemental logging

Syntax

```sql
DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name            IN  VARCHAR2,
    supplemental_logging  IN  VARCHAR2  DEFAULT 'KEYS',
    container             IN  VARCHAR2  DEFAULT 'CURRENT');
```

Parameters

Table 32-16  PREPARE_TABLE_INSTANTIATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as <code>[schema_name.]object_name</code>. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>supplemental_logging</td>
<td>Either <code>NONE</code>, <code>KEYS</code>, or <code>ALL</code>. If <code>NONE</code> is specified, then this procedure does not enable supplemental logging for any columns in the table. This procedure does not remove existing supplemental logging specifications for the table. If <code>KEYS</code> is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the table. The procedure places the key columns for the table in three separate log groups: the primary key columns in an unconditional log group, the unique key columns and bitmap index columns in a conditional log group, and the foreign key columns in a conditional log group. Specifying <code>KEYS</code> does not enable supplemental logging of bitmap join index columns. If <code>ALL</code> is specified, then this procedure enables supplemental logging for all columns in the table. The procedure places all of the columns for the table in an unconditional log group. Supplemental logging is not enabled for columns of the following types: LOB, LONG, LONG RAW, user-defined types, and Oracle-supplied types.</td>
</tr>
</tbody>
</table>
Table 32-16  (Cont.) PREPARE_TABLE_INSTANTIATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Either CURRENT, ALL, or pdb_name. If CURRENT is specified, then this procedure adds supplemental logging for the current container. If ALL is specified, then this procedure adds supplemental logging for all of the containers in the current CDB. If pdb_name is specified, then this procedure adds supplemental logging for the specified PDB. ALL and pdb_name are valid only when you invoke the procedure from the root.</td>
</tr>
</tbody>
</table>

Usage Notes
Run this procedure at the source database. If you use a capture process to capture all of the changes to a table, then use this procedure to prepare the table for instantiation after the capture process has been configured.

32.3.16 SET_PARAMETER Procedure

This procedure sets a capture process parameter to the specified value.

Syntax

```sql
DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name  IN  VARCHAR2,
    parameter     IN  VARCHAR2,
    value         IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 32-17  SET_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the capture process. Do not specify an owner.</td>
</tr>
<tr>
<td>parameter</td>
<td>The name of the parameter you are setting.</td>
</tr>
<tr>
<td>value</td>
<td>The value to which the parameter is set. If NULL, then the parameter is set to its default value.</td>
</tr>
</tbody>
</table>

Capture Process Parameters

The following table lists the parameters for the capture process.
### Table 32-18  Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotatedddlsize</td>
<td>0 or a positive integer up to 4294967296 (4 GB)</td>
<td>0</td>
<td>The maximum length of the DDL text, in bytes, up to which the DDL can be chosen for annotation, if applicable for the DDL. Any DDL text whose size is greater than the specified size is not annotated. The default, 0, indicates that no DDL text processed by this capture process is annotated. Annotating DDL can simplify filtering and transformation of the DDL within Oracle GoldenGate. <strong>Note:</strong> This parameter is intended for Oracle GoldenGate. Do not use this parameter in an Oracle Streams replication environment or in an XStream environment.</td>
</tr>
<tr>
<td>capture_idkey_objects</td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then the capture process captures ID key logical change records (LCRs). If N, then the capture process does not capture ID key LCRs. Capture processes do not fully support capturing changes to some data types from the redo log. ID key LCRs enable an XStream configuration to capture these changes and process them with an XStream client application. <strong>Note:</strong> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter and <em>Oracle Database XStream Guide</em> for more information about ID key LCRs.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Possible Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>capture_sequence_nextval</td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then the capture process captures sequence LCRs for all of the sequences in the database, except for sequences in Oracle-supplied, administrative schemas such as SYS and SYSTEM. The capture process's rule sets can filter sequence LCRs in the same way that they filter row LCRs and DDL LCRs. If N, then the capture process does not capture sequence LCRs. An apply process or XStream inbound server can use sequence LCRs to ensure that the sequence values at a destination database use the appropriate values. For increasing sequences, the sequence values at the destination are equal to or greater than the sequence values at the source database. For decreasing sequences, the sequence values at the destination are less than or equal to the sequence values at the source database. <strong>Note:</strong> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter and &quot;SET_PARAMETER Procedure&quot; for information about the apply_sequence_nextval apply process parameter.</td>
</tr>
<tr>
<td>disable_on_limit</td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then the capture process is disabled because it reached a value specified by the time_limit parameter or message_limit parameter. If N, then the capture process is restarted immediately after stopping because it reached a limit. When a capture process is restarted, it starts to capture changes at the point where it last stopped. A restarted capture process gets a new session identifier, and the processes associated with the capture process also get new session identifiers. However, the capture process number (CPnn) remains the same.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Possible Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>downstream_real_time_min</td>
<td>Y or N</td>
<td>Y</td>
<td>If Y, then the capture process is a real-time downstream capture process. After setting this parameter to Y, switch the redo log file at the source database using the SQL statement ALTER SYSTEM ARCHIVE LOG CURRENT to begin real-time downstream capture. If this parameter is set to Y, then redo data from the source database must be sent to the standby redo log at the downstream database. See Oracle Streams Replication Administrator's Guide for information about creating a real-time downstream capture process. If N, then the capture process is an archived-log downstream capture process. An error is raised if an attempt is made to set this parameter for a local capture process.</td>
</tr>
<tr>
<td>excludetag</td>
<td>Comma-delimited list of Oracle Streams tags</td>
<td>NULL</td>
<td>Controls whether the capture process captures DML changes that are tagged with one of the specified Oracle Streams tags. Whether the capture process captures these changes depends on the settings for the getapplops and getreplicates parameters. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter</td>
</tr>
</tbody>
</table>
Table 32-18  (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclusetrans</td>
<td>Comma-delimited list of transaction names</td>
<td>NULL</td>
<td>Controls whether the capture process captures DML changes in the specified transaction names. Whether the capture process captures these changes depends on the settings for the getapplops and getreplicates parameters. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter</td>
</tr>
<tr>
<td>excludeuser</td>
<td>Comma-delimited list of user names</td>
<td>NULL</td>
<td>Controls whether the capture process captures DML changes made by the specified users. Whether the capture process captures these changes depends on the settings for the getapplops and getreplicates parameters. Specify an exact pattern match for each user name. The pattern match is case sensitive. For example, specify HR for the hr user. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter</td>
</tr>
</tbody>
</table>
Table 32-18  (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>excludeuserid</td>
<td>Comma-delimited list of user ID values</td>
<td>NULL</td>
<td>Controls whether the capture process captures data manipulation language (DML) changes made by the specified users. Whether the capture process captures these changes depends on the settings for the <code>getapplops</code> and <code>getreplicates</code> parameters. To view the user ID for a user, query the <code>USER_ID</code> column in the <code>ALL_USERS</code> data dictionary view. If NULL, then the capture process ignores this parameter. <strong>Note:</strong> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter</td>
</tr>
<tr>
<td>getapplops</td>
<td>Y or N</td>
<td>Y</td>
<td>If Y, then the capture process captures DML changes if the original user is not specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is not specified in the <code>excludetrans</code> parameter. If N, then the capture process ignores DML changes if the original user is not specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is not specified in the <code>excludetrans</code> parameter. In either case, the capture process captures a DML change only if it satisfies the capture process's rule sets. <strong>Note:</strong> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Possible Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getreplicates</td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then the capture process captures DML changes if the original user is specified in the excludeuserid or excludeuser parameters and the transaction name is specified in the excludetrans parameter. If N, then the capture process ignores DML changes if the original user is specified in the excludeuserid or excludeuser parameters and the transaction name is specified in the excludetrans parameter. In either case, the capture process captures a DML change only if it satisfies the capture process's rule sets. Note: This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. See Also: &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
<tr>
<td>ignore_transaction</td>
<td>A valid transaction ID or NULL</td>
<td>NULL</td>
<td>Instructs the capture process to ignore the specified transaction from the source database, effective immediately. The capture process eliminates all subsequent LCRs for the transaction. If the specified transaction is committed successfully at the source database, the destination database will receive a ROLLBACK statement instead, and any LCRs from the transaction that were enqueued before the ignore transaction request are rolled backed at the destination database. If NULL, then the capture process ignores this parameter. Use caution when setting this parameter because ignoring a transaction might lead to data divergence between the source database and destination database. To ignore multiple transactions, specify each transaction in a separate call to the SET_PARAMETER procedure. The DBA_CAPTURE_PARAMETERS view displays a comma-delimited list of all transactions to be ignored. To clear the list of ignored transactions, run the SET_PARAMETER procedure and specify NULL for the ignore_transaction parameter.</td>
</tr>
</tbody>
</table>
Table 32-18  (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignore_unsupported_table</td>
<td>A fully qualified table name, *, or -</td>
<td>*</td>
<td>Controls the behavior of the capture process when it tries to capture changes to a specified table or to an unsupported table. A capture process tries to capture changes to an unsupported table when its rule sets instruct it to do so. If you do not want the capture process to try to capture changes to unsupported tables, then ensure that the capture process’s rule sets exclude unsupported tables. When a table name is specified, the capture process does not capture changes to the specified table. The table name must be entered in the form <code>table_owner. table_name</code>. For example, <code>hr.employees</code>. To specify multiple tables, specify each table in a separate call to the <code>SET_PARAMETER</code> procedure. When <code>*</code> is specified and the capture process tries to capture a change to an unsupported table, the capture process ignores the change and continues to run. The change to the unsupported table is not captured, and the capture process records the unsupported table in the alert log. When <code>-</code> is specified and the capture process tries to capture a change to an unsupported table, the capture process aborts.</td>
</tr>
<tr>
<td>include_objects</td>
<td>A list of tables or schema names separated by commas</td>
<td>none</td>
<td>Directs capture to include changes from the specified tables or schemas. An LCR that is selected by <code>include_objects</code> is passed to the outbound server regardless of any further filtering that is specified. <strong>Note:</strong> This parameter is intended for XStream. Do not use or attempt to set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure. See “ENABLE_GG_XSTREAM_FOR_STREAMS Procedure” for information about enabling XStream optimizations. <strong>See Also:</strong> “Usage Notes” for more information about this parameter</td>
</tr>
</tbody>
</table>
Table 32-18  (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inline_lob_optimization</td>
<td>Y or N</td>
<td>N</td>
<td>If Y, then LOBs that can be processed inline (such as small LOBs) are included in the LCR directly, rather than sending LOB chunk LCRs. If N, then each LOB column is sent as NULL followed by LOB chunk LCRs to update the LOB column. Note: This parameter is intended for XStream. Do not use or attempt to set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FORSTREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FORSTREAMS Procedure&quot; for information about enabling XStream optimizations. See Also: &quot;Usage Notes&quot; for more information about this parameter</td>
</tr>
<tr>
<td>maximum_scn</td>
<td>A valid SCN or INFINITE</td>
<td>INFINITE</td>
<td>The capture process is disabled before capturing a change record with an SCN greater than or equal to the value specified. If INFINITE, then the capture process runs regardless of the SCN value.</td>
</tr>
</tbody>
</table>
### Table 32-18 (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_sga_size</td>
<td>A positive integer</td>
<td>INFINITE</td>
<td>Controls the amount of system global area (SGA) memory allocated specifically to the capture process, in megabytes. The capture process attempts to allocate memory up to this limit. A capture process uses Oracle LogMiner to scan for changes in the redo log. The memory is allocated for the duration of the capture process session and is released when the capture process becomes disabled. <strong>Note:</strong> The sum of system global area (SGA) memory allocated for all components on a database must be less than the value set for the STREAMS_POOL_SIZE initialization parameter. If <strong>NULL</strong>, then the capture component uses the original default value. A <strong>NULL</strong> value has the same effect as resetting the parameter to its default value. <strong>Note:</strong> This parameter is intended for XStream. Do not use or attempt to set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
</tbody>
</table>
Table 32-18  (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge_threshold</td>
<td>A negative integer, 0, a positive integer, or INFINITE</td>
<td>60</td>
<td>The amount of time, in seconds, between the message creation time of the original capture process and the message creation time of the cloned capture process. Specifically, if the difference, in seconds, between the CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process and the original capture process is less than or equal to the value specified for this parameter, then automatic merge begins by running the MERGE_STREAMS procedure. If the difference is greater than the value specified by this parameter, then automatic merge does not begin, and the value is recorded in the LAG column of the DBA_STREAMS_SPLIT_MERGE view. The CAPTURE_MESSAGE_CREATE_TIME is recorded in the V$STREAMS_CAPTURE view. This parameter is relevant only when changes captured by the capture process are applied by two or more apply processes and the split_threshold parameter is set to a value other than INFINITE. If a negative value is specified, then automatic merge is disabled. If 0 (zero) is specified, then there must be no lag between the original capture process and the cloned capture process to begin the merge. If INFINITE is specified, then automatic merging starts immediately. See Also: Oracle Streams Replication Administrator's Guide</td>
</tr>
<tr>
<td>message_limit</td>
<td>A positive integer or INFINITE</td>
<td>INFINITE</td>
<td>The capture process stops after capturing the specified number of messages. If INFINITE, then the capture process continues to run regardless of the number of messages captured.</td>
</tr>
</tbody>
</table>
### Table 32-18  (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_tracking_frequency</td>
<td>0 or a positive integer</td>
<td>2000000</td>
<td>The frequency at which messages captured by the capture process are tracked automatically. For example, if this parameter is set to the default value of 2000000, then every two-millionth message is tracked automatically. The tracking label used for automatic message tracking is <code>capture_process_name:AUTOTRACK</code>, where <code>capture_process_name</code> is the name of the capture process. Only the first 20 bytes of the capture process name are used; the rest is truncated if it exceeds 20 bytes. If 0 (zero), then no messages are tracked automatically. See Oracle Streams Replication Administrator's Guide for more information about message tracking.</td>
</tr>
<tr>
<td>parallelism</td>
<td>A positive integer</td>
<td>1</td>
<td>The number of preparer servers that can concurrently mine the redo log for the capture process. A capture process consists of one reader server, one or more preparer servers, and one builder server. The preparer servers concurrently format changes found in the redo log into logical change records (LCRs). Each reader server, preparer server, and builder server is a process, and the number of preparer servers equals the number specified for the <code>parallelism</code> capture process parameter. So, if <code>parallelism</code> is set to 5, then a capture process uses a total of seven processes: one reader server, five preparer servers, and one builder server. Setting the <code>parallelism</code> parameter to a number higher than the number of available parallel execution servers might disable the capture process. Ensure that the PROCESSES initialization parameter is set appropriately when you set the <code>parallelism</code> capture process parameter. <strong>Note:</strong> When you change the value of this parameter, the capture process is stopped and restarted automatically. <strong>See Also:</strong> Oracle Streams Concepts and Administration for more information about capture process components</td>
</tr>
</tbody>
</table>
### Table 32-18  (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip_autofiltered_table_ddl</td>
<td>Y or N</td>
<td>Y</td>
<td>If Y, then the capture process does not capture data definition language (DDL) changes to tables that are automatically filtered by the capture process. If N, then the capture process can capture DDL changes to tables that are automatically filtered by the capture process if the DDL changes satisfy the capture process rule sets. The AUTO_FILTERED column in the DBA_STREAMS_UNSUPPORTED data dictionary view shows which tables are automatically filtered by capture processes.</td>
</tr>
<tr>
<td>split_threshold</td>
<td>0, a positive integer, or INFINITE</td>
<td>1800</td>
<td>The amount of time, in seconds, that a stream is broken before the stream is automatically split from other streams that flow from the capture process. When a stream is split, the capture process, queue, and propagation are cloned. In this case, a stream is a flow of logical change records (LCRs) that flows from a capture process to an apply. A stream is broken when LCRs captured by the capture process cannot reach the apply process. For example, a stream is broken when the relevant propagation or apply process is disabled. This parameter is relevant only when changes captured by the capture process are applied by two or more apply processes. If 0 (zero), then automatic splitting starts immediately. If INFINITE, then automatic splitting is disabled. The stream is not split regardless of the amount of time that it is broken. This parameters is designed to be used with the merge_threshold parameter. You can monitor an automatic split and merge operation by querying the DBA_STREAMS_SPLIT_MERGE view. <strong>See Also:</strong> Oracle Streams Replication Administrator’s Guide</td>
</tr>
<tr>
<td>startup_seconds</td>
<td>0, a positive integer, or INFINITE</td>
<td>0</td>
<td>The maximum number of seconds to wait for another instantiation of the same capture process to finish. If the other instantiation of the same capture process does not finish within this time, then the capture process does not start. This parameter is useful only if you are starting the capture process manually. If INFINITE, then the capture process does not start until another instantiation of the same capture process finishes.</td>
</tr>
</tbody>
</table>
### Table 32-18 (Cont.) Capture Process Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Possible Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_limit</td>
<td>A positive integer or INFINITE</td>
<td>INFINITE</td>
<td>The capture process stops as soon as possible after the specified number of seconds since it started. If INFINITE, then the capture process continues to run until it is stopped explicitly.</td>
</tr>
<tr>
<td>trace_level</td>
<td>0 or a positive integer</td>
<td>0</td>
<td>Set this parameter only under the guidance of Oracle Support Services.</td>
</tr>
<tr>
<td>use_rac_service</td>
<td>Y or N</td>
<td>Y if Streams</td>
<td>N if XStream</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If Y, then the capture process is run in the owning instance of its queue. If N, then the client specifies where the capture process is to run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> This parameter is intended for XStream. Do not use or attempt to set this parameter in an Oracle Streams replication environment unless XStream optimizations are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See &quot;ENABLE_GG_XSTREAM_FOR_STREAMS Procedure&quot; for information about enabling XStream optimizations. <strong>See Also:</strong> &quot;Usage Notes&quot; for more information about this parameter</td>
</tr>
<tr>
<td>write_alert_log</td>
<td>Y or N</td>
<td>Y</td>
<td>If Y, then the capture process writes a message to the alert log on exit. If N, then the capture process does not write a message to the alert log on exit. The message specifies the reason the capture process stopped.</td>
</tr>
<tr>
<td>xout_client_exists</td>
<td>Y or N</td>
<td>Y if the capture process sends LCRs to XStream outbound servers N if the capture process sends LCRs to Oracle Streams apply processes</td>
<td>Y indicates that the capture process sends LCRs to one or more XStream outbound servers. N indicates that the capture process sends LCRs to one or more Oracle Streams apply processes. A single capture process cannot send LCRs to both outbound servers and apply processes. In an XStream configuration where an outbound server runs on a different database than its capture process, set this parameter to Y to enable the capture process to send LCRs to the outbound server. <strong>Note:</strong> Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.</td>
</tr>
</tbody>
</table>

### Usage Notes

The following usage notes apply to the SET_PARAMETER procedure:

- **Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.
• Delays Are Possible Before New Parameter Settings Take Effect
• Parameters Interpreted as Positive Integers
• Parameters with a System Change Number (SCN) Setting
• Parameters that Require XStream Optimizations
• XStream or Oracle GoldenGate Integrated Capture Configurations

Delays Are Possible Before New Parameter Settings Take Effect

When you alter a parameter value, a short amount of time might pass before the new value for the parameter takes effect.

Parameters Interpreted as Positive Integers

For all parameters that are interpreted as positive integers, the maximum possible value is 4,294,967,295. Where applicable, specify INFINITE for larger values.

Parameters with a System Change Number (SCN) Setting

For parameters that require an SCN setting, any valid SCN value can be specified.

Parameters that Require XStream Optimizations

A capture process uses the following parameters only when the capture process is sending logical change records (LCRs) to an XStream outbound server or when XStream optimizations are enabled for Oracle Streams components:

• capture_idkey_objects
• capture_sequence_nextval
• excludetag
• excludetrans
• excludeuser
• excludeuserid
• getapplops
• getreplicates
• include_objects
• inline_lob_optimization
• max_sga_size
• use_rac_services

The DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure enables XStream optimizations for Oracle Streams. When XStream optimizations are not enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure, a capture process raises an error if one of these parameters is set to any value other than its default value.

When XStream optimizations are enabled for Oracle Streams and the capture_idkey_objects parameter is set to Y, a capture process can capture ID key LCRs. ID key LCRs do not contain all of the columns for a row change. Instead, they contain the rowid of the changed row, a group of key columns to identify the row in the table, and the data for the scalar columns of the table that are supported by capture
processes. An apply process can apply these changes using the information available the ID key LCRs.

To determine the database objects for which a capture process will capture ID key LCRs, run the following query on the source database:

```sql
SELECT OWNER, OBJECT_NAME
FROM DBA_XSTREAM_OUT_SUPPORT_MODE
WHERE SUPPORT_MODE='ID KEY';
```

**Note:**
Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.

**See Also:**
- Oracle Database XStream Guide for more information about ID key LCRs
- Oracle Streams Concepts and Administration for information about the data types supported by capture processes

XStream or Oracle GoldenGate Integrated Capture Configurations

In an XStream or Oracle GoldenGate integrated capture configuration, the following parameters control which changes are captured by a capture process:

- `capture_idkey_objects`
- `capture_sequence_nextval`
- `excludetag`
- `excludetrans`
- `excludeuser`
- `excludeuserid`
- `getapplops`
- `getreplicates`
- `include_objects`
- `inline_lob_optimization`
- `max_sga_size`
- `use_rac_services`

You can set these parameters to avoid change cycling. Change cycling sends a change back to the database where it originated. Typically, change cycling should be avoided in a replication environment so that the same change is not made to a database more than once.

In an XStream or Oracle GoldenGate integrated capture configuration that performs bi-directional replication, a GoldenGate Replicat process runs on the source database for a
capture process. Therefore, the changes made by the GoldenGate Replicat are recorded in the redo log.

If an integrated configuration performs bi-directional replication, then, to avoid change cycling, the capture process should not capture the changes made by the Oracle GoldenGate Replicat process. To accomplish this goal, use the default settings for the `getapplops` and `getreplicates` parameters and exclude changes made by the user running the Replicat process. To exclude these changes, specify this user in the `excludeuserid` or `excludeuser` parameter. Typically, the user running the Oracle GoldenGate Replicat process is the XStream administrator.

In some configurations, the goal might be to capture or exclude changes made by applications or by the Replicat process. For example, an intermediate database in a replication environment might capture all of the changes made to the database, including both application changes and Replicat process changes, and send these changes to a different destination database.

*Table 32-19* describes the capture process behavior when at least one of the exclude parameters is non-NULL.

**Table 32-19  Behavior When at Least One exclude Parameter Is Non-NULL**

<table>
<thead>
<tr>
<th>getapplops Setting</th>
<th>getreplicates Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>The capture process captures all DML changes.</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>The capture process captures the DML changes made by the users that are not in the <code>excludeuserid</code> or <code>excludeuser</code> parameters. The capture process captures the DML changes that are not in the transactions in the <code>excludetrans</code> parameter. The capture process captures only the DML changes that do not have a tag that is in the <code>excludetags</code> parameter.</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>The capture process captures only the DML changes made by the users that are in the <code>excludeuserid</code> or <code>excludeuser</code> parameters. The capture process captures only the DML changes that are in the transactions in the <code>excludetrans</code> parameter. The capture process captures only the DML changes that have a tag that is in the <code>excludetags</code> parameter.</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>The capture process does not capture any DML changes.</td>
</tr>
</tbody>
</table>

*Table 32-19* describes the capture process behavior when all of the exclude parameters are set to NULL.
Table 32-20  Behavior When All exclude Parameters Are Set to NULL

<table>
<thead>
<tr>
<th>getapplops Setting</th>
<th>getreplicates Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>The capture process captures all DML changes.</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>The capture process captures all DML changes.</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>The capture process does not capture any DML changes.</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>The capture process does not capture any DML changes.</td>
</tr>
</tbody>
</table>

See the documentation for the Oracle GoldenGate product for more information:

http://docs.oracle.com/cd/E15881_01/index.htm

**Note:**

A capture process evaluates a change using these parameters before it evaluates a change using its rule sets. Therefore, a capture process can discard a change before the change is evaluated against the capture process’s rule sets. Also, regardless of the settings for these parameters, a capture process captures a change only if the change satisfies the capture process’s rule sets.

**See Also:**

*Oracle Database XStream Guide*

### 32.3.17 START_CAPTURE Procedure

This procedure starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue.

The start status is persistently recorded. Hence, if the status is enabled, then the capture process is started upon database instance startup.

The capture process is a background Oracle process and is prefixed by c.

The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the start status of a capture process.

**See Also:**

*DBMSSTREAMSADM*
Syntax

```sql
DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  IN  VARCHAR2);
```

Parameters

**Table 32-21 START_CAPTURE Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the capture process. Do not specify an owner. The capture process uses LogMiner to capture changes in the redo information. A NULL setting is not allowed.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The capture process status is persistently recorded. Hence, if the status is `ENABLED`, then the capture process is started upon database instance startup. A capture process (``cppn``) is an Oracle background process.

### 32.3.18 STOP_CAPTURE Procedure

This procedure stops the capture process from mining redo logs.

**Syntax**

```sql
DBMS_CAPTURE_ADM.STOP_CAPTURE(
    capture_name  IN  VARCHAR2,
    force         IN  BOOLEAN  DEFAULT FALSE);
```

**Parameters**

**Table 32-22 STOP_CAPTURE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the capture process. A NULL setting is not allowed. Do not specify an owner.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, then the procedure stops the capture process as soon as possible. If the capture process cannot stop normally, then it aborts. If FALSE, then the procedure stops the capture process as soon as possible. If the capture process cannot stop normally, then an ORA-26672 error is returned, and the capture process might continue to run.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The following usage notes apply to this procedure:

- The capture process status is persistently recorded. Hence, if the status is `DISABLED` or `ABORTED`, then the capture process is not started upon database instance startup.
- A capture process is an Oracle background process with a name in the form `CPnn`, where `nn` can include letters and numbers.
• The enqueue and dequeue state of DBMS_AQADM.START_QUEUE and DBMS_AQADM.STOP_QUEUE have no effect on the stop status of a capture process.
The **DBMS_COMPARISON** package provides interfaces to compare and converge database objects at different databases.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Views
- Operational Notes
- Data Structures
- Summary of DBMS_COMPARISON Subprograms

### See Also:

- *Oracle Streams Replication Administrator's Guide* for information about using the basic features of this package

## 33.1 DBMS_COMPARISON Overview

The **DBMS_COMPARISON** package is an Oracle-supplied package that you can use to compare database objects at two databases. This package also enables you converge the database objects so that they are consistent at different databases. Typically, this package is used in environments that share a database object at multiple databases. When copies of the same database object exist at multiple databases, the database object is a **shared database object**. Several data dictionary views contain information about comparisons made with the **DBMS_COMPARISON** package.

Shared database objects might be maintained by data replication. For example, materialized views or Oracle Streams components might replicate the database objects and maintain them at multiple databases. A custom application might also maintain shared database objects. When a database object is shared, it can diverge at the databases that share it. You can use this package to identify differences in the shared database objects. After identifying the differences, you can optionally use this package to synchronize the shared database objects.

To compare a database object that is shared at two different databases, complete the following general steps:

1. Run the **CREATE_COMPARISON Procedure** in this package to create a **comparison**. The comparison identifies the database objects to compare and specifies parameters for the comparison.
2. Run the **COMPARE Function** in this package to compare the database object at the two databases and identify differences. This function returns **TRUE** when no differences are found and **FALSE** when differences are found. This function also populates data dictionary views with comparison results. Separate comparison results are generated for each execution of the **COMPARE function**.

3. If you want to examine the comparison results, query the following data dictionary views:
   - `DBA_COMPARISON_SCAN`
   - `USER_COMPARISON_SCAN`
   - `DBA_COMPARISON_SCAN_VALUES`
   - `USER_COMPARISON_SCAN_VALUES`
   - `DBA_COMPARISON_ROW_DIF`
   - `USER_COMPARISON_ROW_DIF`

4. If there are differences, and you want to synchronize the database objects at the two databases, then run the **CONVERGE** procedure in this package.

After you create a comparison with the **CREATE_COMPARISON** procedure in the **DBMS_COMPARISON** package, you can run the comparison at any time using the **COMPARE** function. Each time you run the **COMPARE** function, it records comparison results in the appropriate data dictionary views. Comparison results might be modified when subprograms in this package are invoked and the scans in the comparison results are specified. For example, comparison results might be modified when you run the **RECHECK** function.

The comparison results for a single execution of the **COMPARE** function can include one or more **scans**. A scan checks for differences in some or all of the rows in a shared database object at a single point in time. You can compare database objects multiple times, and a unique scan ID identifies each scan in the comparison results.

A **bucket** is a range of rows in a database object that is being compared. Buckets improve performance by splitting the database object into ranges and comparing the ranges independently. Every comparison divides the rows being compared into an appropriate number of buckets, and each bucket is compared by a scan.

Each time the **COMPARE** function splits a bucket into smaller buckets, it performs new scans of the smaller buckets. The scan that analyzes a larger bucket is the **parent scan** of each scan that analyzes the smaller buckets into which the larger bucket was split. The **root scan** in the comparison results is the highest level parent scan. The root scan does not have a parent.

You can recheck a scan using the **RECHECK** function, and you can converge a scan using the **CONVERGE** procedure. When you want to recheck or converge all of the rows comparison results, specify the root scan ID for the comparison results in the appropriate subprogram. When you want to recheck or converge a portion of the rows in comparison results, specify the scan ID of the scan that contains the differences.
33.2 DBMS_COMPARISON Security Model

Security on this package can be controlled in one of two ways.

- Granting `EXECUTE` on this package to selected users or roles.
- Granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

Each subprogram in the `DBMS_COMPARISON` package has a `comparison_name` parameter. The current user must be the owner of the specified comparison to run a subprogram in the `DBMS_COMPARISON` package.

To run the `COMPARE` function, `RECHECK` function, or `CONVERGE` procedure, the following users must have the `SELECT` or `READ` privilege on each copy of the shared database object:

- The comparison owner at the local database
- When a database link is used, the user at the remote database to which the comparison owner connects through a database link

The `CONVERGE` procedure also requires additional privileges for one of these users at the database where it makes changes to the shared database object. The user must have `INSERT`, `UPDATE`, and `DELETE` privileges on the shared database object at this database.

In addition, when the `CONVERGE` procedure is run with either the `local_converge_tag` or `remote_converge_tag` parameter set to a non-NULL value, then the following additional requirements must be met:

- If the local table "wins," then the user at the remote database to which the invoker of the `CONVERGE` procedure connects through a database link must be granted either `EXECUTE_CATALOG_ROLE` or `EXECUTE` privilege on the `DBMS_STREAMS_ADM` package.
- If the remote table "wins," then the invoker of the `CONVERGE` procedure at the local database must be granted either `EXECUTE_CATALOG_ROLE` or `EXECUTE` privilege on the `DBMS_STREAMS_ADM` package.

Note:

The database administrator (DBA) can assume control over some of the `DBMS_COMPARISON` functions and procedures owned by other users. This control applies to `DROP_COMPARISON` and `PURGE_COMPARISON`. This DBA override can be particularly useful in cleanup operations when comparisons created by another user need to be dropped.
33.3 DBMS_COMPARISON Constants

The DBMS_COMPARISON package defines several enumerated constants to use specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS_COMPARISON.CMP_SCAN_MODE_FULL.

Table 33-1 lists the parameters and enumerated constants.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_mode</td>
<td>CMP_COMPARE_MODE_OBJECT</td>
<td>VARCHAR2(30)</td>
<td>CMP_COMPARE_MODE_OBJECT is a database object. This constant can be specified as 'OBJECT'.</td>
</tr>
<tr>
<td>scan_mode</td>
<td>CMP_SCAN_MODE_FULL</td>
<td>VARCHAR2(30)</td>
<td>CMP_SCAN_MODE_FULL indicates that the entire database object is compared. This constant can be specified as 'FULL'.</td>
</tr>
<tr>
<td></td>
<td>CMP_SCAN_MODE_RANDOM</td>
<td></td>
<td>CMP_SCAN_MODE_RANDOM indicates that a random portion of the database object is compared. This constant can be specified as 'RANDOM'.</td>
</tr>
<tr>
<td></td>
<td>CMP_SCAN_MODE_CYCLIC</td>
<td></td>
<td>CMP_SCAN_MODE_CYCLIC indicates that a portion of the database object is compared when you perform a single comparison. When you compare the database object again, another portion of the database object is compared, starting where the last comparison ended. This constant can be specified as 'CYCLIC'.</td>
</tr>
<tr>
<td></td>
<td>CMP_SCAN_MODE_CUSTOM</td>
<td></td>
<td>CMP_SCAN_MODE_CUSTOM indicates that the user who runs the subprogram specifies the range to compare in the database object. This constant can be specified as 'CUSTOM'.</td>
</tr>
<tr>
<td>converge_options</td>
<td>CMP_CONVERGE_LOCAL_WINS</td>
<td>VARCHAR2(30)</td>
<td>CMP_CONVERGE_LOCAL_WINS indicates that the column values at the local database replace the column values at the remote database when these column values are different. This constant can be specified as 'LOCAL'.</td>
</tr>
<tr>
<td></td>
<td>CMP_CONVERGE_REMOTE_WINS</td>
<td></td>
<td>CMP_CONVERGE_REMOTE_WINS indicates that the column values at the remote database replace the column values at the local database when these column values are different. This constant can be specified as 'REMOTE'.</td>
</tr>
</tbody>
</table>
Table 33-1  (Cont.) DBMS_COMPARISON Parameters with Enumerated Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>null_value</td>
<td>CMP_NULL_VALUE_DEF</td>
<td>VARCHAR2(100)</td>
<td>CMP_NULL_VALUE_DEF indicates that ORA$STREAMS$NV is substituted for NULL values in database objects during comparison. This constant can be specified as 'ORA$STREAMS$NV'.</td>
</tr>
<tr>
<td>max_num_buckets</td>
<td>CMP_MAX_NUM_BUCKETS</td>
<td>INTEGER</td>
<td>CMP_MAX_NUM_BUCKETS indicates that the maximum number of buckets is 1,000. This constant can be specified as 1000.</td>
</tr>
<tr>
<td>min_rows_in_bucket</td>
<td>CMP_MIN_ROWS_IN_BUCKET</td>
<td>INTEGER</td>
<td>CMP_MIN_ROWS_IN_BUCKET indicates that the minimum number of rows in a bucket is 10,000. This constant can be specified as 10000.</td>
</tr>
</tbody>
</table>

33.4 DBMS_COMPARISON Views

The DBMS_COMPARISON package uses several views.

These views are listed below:

- DBA_COMPARISON
- USER_COMPARISON
- DBA_COMPARISON_COLUMNS
- USER_COMPARISON_COLUMNS
- DBA_COMPARISON_SCAN
- USER_COMPARISON_SCAN
- DBA_COMPARISON_SCAN_VALUES
- USER_COMPARISON_SCAN_VALUES
- DBA_COMPARISON_ROW_DIF
- USER_COMPARISON_ROW_DIF

See Also:

Oracle Database Reference
33.5 DBMS_COMPARISON Operational Notes

The DBMS_COMPARISON package has certain requirements and operational notes. These include the following:

- Oracle Database Release Requirements for the DBMS_COMPARISON Package
- Database Character Set Requirements for the DBMS_COMPARISON Package
- Database Object Requirements for the DBMS_COMPARISON Package
- Index Column Requirements for the DBMS_COMPARISON Package
- Datatype Requirements for the DBMS_COMPARISON Package
- Only Converge Rows That Are Not Being Updated

Oracle Database Release Requirements for the DBMS_COMPARISON Package

Meet the following Oracle Database release requirements when running the subprograms in the DBMS_COMPARISON package:

- The local database that runs the subprograms in the DBMS_COMPARISON package must be an Oracle Database 11g Release 1 (11.1) database.
- The remote database must be an Oracle Database 10g Release 1 (10.1) or later database. Oracle databases before this release and non-Oracle databases are not supported.

Database Character Set Requirements for the DBMS_COMPARISON Package

The database character sets must be the same for the databases that contain the database objects being compared.

See Also:

Oracle Database Globalization Support Guide for information about database character sets

Database Object Requirements for the DBMS_COMPARISON Package

The DBMS_COMPARISON package can compare the following types of database objects:

- Tables
- Single-table views
- Materialized views
- Synonyms for tables, single-table views, and materialized views

Database objects of different types can be compared and converged at different databases. For example, a table at one database and a materialized view at another database can be compared and converged with this package.

To run the subprograms in the DBMS_COMPARISON package, the specified database objects must have the same shape at each database. Specifically, the database
objects must have the same number of columns at each database, and the datatypes of corresponding columns must match.

If a database object being compared contains columns that do not exist in the other database object, then you can compare the database objects by excluding the extra columns during comparison creation. Use the column_list parameter in the CREATE_COMPARISON procedure to list only the columns that exist in both database objects.

See Also:
CREATE_COMPARISON Procedure

Index Column Requirements for the DBMS_COMPARISON Package

This section discusses number, timestamp, and interval columns. These include the following datatypes:

- **Number columns are of the following datatypes:** NUMBER, FLOAT, BINARY_FLOAT, and BINARY_DOUBLE.
- **Timestamp columns are of the following datatypes:** TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE
- **Interval columns are of the following datatypes:** INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND.

For all scan modes to be supported by the DBMS_COMPARISON package, the database objects must have one of the following types of indexes:

- A single-column index on a number, timestamp, interval, or DATE datatype column
- A composite index that only includes number, timestamp, interval, or DATE datatype columns. Each column in the composite index must either have a NOT NULL constraint or be part of the primary key.

For the scan modes CMP_SCAN_MODE_FULL and CMP_SCAN_MODE_CUSTOM to be supported, the database objects must have one of the following types of indexes:

- A single-column index on a number, timestamp, interval, DATE, VARCHAR2, or CHAR datatype column
- A composite index that only includes number, timestamp, interval, DATE, VARCHAR2, or CHAR columns. Each column in the composite index must either have a NOT NULL constraint or be part of the primary key.

If the database objects do not have one of these types of indexes, then the DBMS_COMPARISON package does not support the database objects. For example, if the database objects only have a single index on an NVARCHAR2 column, then the DBMS_COMPARISON package does not support them. Or, if the database objects have only one index, and it is a composite index that includes a NUMBER column and an NCHAR column, then the DBMS_COMPARISON package does not support them.

You can specify an index when you create a comparison using the index_schema_name and index_name parameters in the CREATE_COMPARISON procedure. If you specify an index, then make sure the columns in the index meet the requirements of the scan mode used for the comparison.
The index columns in a comparison must uniquely identify every row involved in a comparison. The following constraints satisfy this requirement:

- A primary key constraint
- A unique constraint on one or more non-NULL columns

If these constraints are not present on a table, then use the `index_schema_name` and `index_name` parameters in the `CREATE_COMPARISON` procedure to specify an index whose columns satisfy this requirement.

When a single index value identifies both a local row and a remote row, the two rows must be copies of the same row in the replicated tables. In addition, each pair of copies of the same row must always have the same index value.

The `DBMS_COMPARISON` package can use an index only if all of the columns in the index are included in the `column_list` parameter when the comparison is created with the `CREATE_COMPARISON` procedure.

After a comparison is created, you can determine the index column or columns for it by running the following query:

```
SELECT COLUMN_NAME, COLUMN_POSITION FROM DBA_COMPARISON_COLUMNS
WHERE COMPARISON_NAME = 'COMPARE_CUSTOM' AND INDEX_COLUMN = 'Y';
```

If there are multiple index columns, then the index column with 1 for the `COLUMN_POSITION` is the lead index column in the composite index.

**See Also:**
- "Constants" for information about scan modes
- `CREATE_COMPARISON Procedure` for information about specifying an index for a comparison

**Datatype Requirements for the DBMS_COMPARISON Package**

The `DBMS_COMPARISON package` can compare data in columns of the following datatypes:

- VARCHAR2
- NVARCHAR2
- NUMBER
- FLOAT
- DATE
- BINARY_FLOAT
- BINARY_DOUBLE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
• INTERVAL YEAR TO MONTH
• INTERVAL DAY TO SECOND
• RAW
• CHAR
• NCHAR

If a column with datatype TIMESTAMP WITH LOCAL TIME ZONE is compared, then the two databases must use the same time zone. Also, if a column with datatype NVARCHAR2 or NCHAR is compared, then the two databases must use the same national character set.

The DBMS_COMPARISON package cannot compare data in columns of the following datatypes:

• LONG
• LONG RAW
• ROWID
• UROWID
• CLOB
• NCLOB
• BLOB
• BFILE
• User-defined types (including object types, REFS, varrays, and nested tables)
• Oracle-supplied types (including any types, XML types, spatial types, and media types)

You can compare database objects that contain unsupported columns by excluding the unsupported columns during comparison creation. Use the column_list parameter in the CREATE_COMPARISON procedure to list only the supported columns in a shared database object.

See Also:
• CREATE_COMPARISON Procedure
• Oracle Database SQL Language Reference for more information about datatypes
• Oracle Database Globalization Support Guide for information about national character sets

Only Converge Rows That Are Not Being Updated

You should only converge rows that are not being updated on either database. For example, if the shared database object is updated by replication components, then only converge rows for which replication changes have been applied and make sure no new changes are in the process of being replicated for these rows. If you compare replicated database objects, then it is typically best to compare them during a time of little or no replication activity to identify persistent differences.
Note:
If a scan identifies that a row is different in the shared database object at two databases, and the row is modified after the scan, then it can result in unexpected data in the row after the CONVERGE procedure is run.

See Also:
Oracle Streams Replication Administrator's Guide for information about the DBMS_COMPARISON package in an Oracle Streams replication environment

33.6 DBMS_COMPARISON Data Structures

The DBMS_COMPARISON package defines a RECORD type.

Contains information returned by the COMPARE function or CONVERGE procedure in the DBMS_COMPARISON package.

Note:
The COMPARE function only returns a value for the scan_id field.

33.6.1 COMPARISON_TYPE Record Type

This record type contains information returned by the COMPARE function or CONVERGE procedure in the DBMS_COMPARISON package.

Note:
The COMPARE function only returns a value for the scan_id field.

Syntax

```sql
TYPE COMPARISON_TYPE IS RECORD(
    scan_id            NUMBER,
    loc_rows_merged    NUMBER,
    rmt_rows_merged    NUMBER,
    loc_rows_deleted   NUMBER,
    rmt_rows_deleted   NUMBER);
```
Table 33-2  COMPARISON_TYPE Attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan_id</td>
<td>The scan ID of the scan</td>
</tr>
<tr>
<td>loc_rows_merged</td>
<td>The number of rows in the local database object updated with information</td>
</tr>
<tr>
<td></td>
<td>from the database object at the remote site</td>
</tr>
<tr>
<td>rmt_rows_merged</td>
<td>The number of rows in the database object updated at the remote site with</td>
</tr>
<tr>
<td></td>
<td>information from the database object at the local site</td>
</tr>
<tr>
<td>loc_rows_deleted</td>
<td>The number of rows deleted from the local database object</td>
</tr>
<tr>
<td>rmt_rows_deleted</td>
<td>The number of rows deleted from the remote database object</td>
</tr>
</tbody>
</table>

33.7 Summary of DBMS_COMPARISON Subprograms

This table lists the DBMS_COMPARISON subprograms and briefly describes them.

Table 33-3  DBMS_COMPARISON Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPARE Function</td>
<td>Performs the specified comparison</td>
</tr>
<tr>
<td>CONVERGE Procedure</td>
<td>Executes data manipulation language (DML) changes to synchronize the portion</td>
</tr>
<tr>
<td></td>
<td>of the database object that was compared in the specified scan</td>
</tr>
<tr>
<td>CREATE_COMPARISON Procedure</td>
<td>Creates a comparison</td>
</tr>
<tr>
<td>DROP_COMPARISON Procedure</td>
<td>Drops a comparison</td>
</tr>
<tr>
<td>PURGE_COMPARISON Procedure</td>
<td>Purges the comparison results, or a subset of the comparison results, for a</td>
</tr>
<tr>
<td>RECHECK Function</td>
<td>Rechecks the differences in a specified scan for a comparison</td>
</tr>
</tbody>
</table>

33.7.1 COMPARE Function

This function performs the specified comparison.

Each time a comparison is performed, it results in at least one new scan, and each scan has a unique scan ID. You can define and name a comparison using the CREATE_COMPARISON procedure.

See Also:

- “Overview”
- CREATE_COMPARISON Procedure
Syntax

```sql
DBMS_COMPARISON.COMPARE(
    comparison_name  IN   VARCHAR2,
    scan_info        OUT  COMPARISON_TYPE,
    min_value        IN   VARCHAR2   DEFAULT NULL,
    max_value        IN   VARCHAR2   DEFAULT NULL,
    perform_row_dif  IN   BOOLEAN    DEFAULT FALSE)
RETURN BOOLEAN;
```

Parameters

### Table 33-4  COMPARE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_name</td>
<td>The name of the comparison.</td>
</tr>
<tr>
<td>scan_info</td>
<td>Information about the compare operation returned in the COMPARISON_TYPE datatype. See COMPARISON_TYPE Record Type.</td>
</tr>
<tr>
<td>min_value</td>
<td>When the scan mode for the comparison is set to CMP_SCAN_MODE_CUSTOM, specify the minimum index column value for the range of rows that are being compared. To determine the index column for a comparison, query the DBA_COMPARISON_COLUMNS data dictionary view. For a composite index, specify a value for the column with column_position equal to 1 in the DBA_COMPARISON_COLUMNS view. See the index column requirements under DBMS_COMPARISON Operational Notes. If the scan mode is set to a value other than CMP_SCAN_MODE_CUSTOM, then this parameter must be set to NULL. If NULL and the scan_mode parameter is set to CMP_SCAN_MODE_CUSTOM, then an error is raised. To determine the scan mode for the comparison, query the DBA_COMPARISON data dictionary view. See DBMS_COMPARISON Constants for information about scan modes.</td>
</tr>
<tr>
<td>max_value</td>
<td>When the scan mode for the comparison is set to CMP_SCAN_MODE_CUSTOM, specify the maximum index column value for the range of rows that are being compared. To determine the index column for a comparison, query the DBA_COMPARISON_COLUMNS data dictionary view. For a composite index, specify a value for the column with column_position equal to 1 in the DBA_COMPARISON_COLUMNS view. See the index column requirements under DBMS_COMPARISON Operational Notes. If the scan mode is set to a value other than CMP_SCAN_MODE_CUSTOM, then this parameter must be set to NULL. If NULL and the scan_mode parameter is set to CMP_SCAN_MODE_CUSTOM, then an error is raised. To determine the scan mode for the comparison, query the DBA_COMPARISON data dictionary view. See DBMS_COMPARISON Constants for information about scan modes.</td>
</tr>
</tbody>
</table>
Table 33-4  (Cont.) COMPARE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>perform_row_dif</td>
<td>If TRUE, then compares each row individually in the database object being compared after reaching the smallest possible bucket for the comparison.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then compares buckets for differences but does not compare each row individually when differences are found in the smallest possible bucket.</td>
</tr>
<tr>
<td></td>
<td>See DBMS_COMPARISON Overview for information about buckets.</td>
</tr>
</tbody>
</table>

Return Values

This function returns TRUE when no differences are found in the database objects being compared. This function returns FALSE when differences are found in the database objects being compared.

33.7.2 CONVERGE Procedure

This procedure executes data manipulation language (DML) changes to synchronize the portion of the database objects that was compared in the specified scan.

Syntax

```
DBMS_COMPARISON.CONVERGE(
    comparison_name      IN   VARCHAR2,
    scan_id              IN   NUMBER,
    scan_info            OUT  COMPARISON_TYPE,
    converge_options     IN   VARCHAR2  DEFAULT CMP_CONVERGE_LOCAL_WINS,
    perform_commit       IN   BOOLEAN   DEFAULT TRUE,
    local_converge_tag   IN   RAW       DEFAULT NULL,
    remote_converge_tag  IN   RAW       DEFAULT NULL);
```

Parameters

Table 33-5  CONVERGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_name</td>
<td>The name of the comparison.</td>
</tr>
<tr>
<td>scan_id</td>
<td>The identifier for the scan that contains the differences between the database objects being converged.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Overview&quot; for more information about specifying a scan ID in this parameter.</td>
</tr>
<tr>
<td>scan_info</td>
<td>Information about the converge operation returned in the COMPARISON_TYPE datatype.</td>
</tr>
<tr>
<td></td>
<td>See COMPARISON_TYPE Record Type.</td>
</tr>
<tr>
<td>converge_options</td>
<td>Either the CMP_CONVERGE_LOCAL_WINS constant or the CMP_CONVERGE_REMOTE_WINS constant.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Constants&quot; for information about these constants.</td>
</tr>
</tbody>
</table>
Table 33-5  (Cont.) CONVERGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>perform_commit</td>
<td>If TRUE, then performs a COMMIT periodically while making the DML changes. The CONVERGE procedure might perform more than one COMMIT when this parameter is set to TRUE. If FALSE, then does not perform a COMMIT after making DML changes.</td>
</tr>
<tr>
<td>local_converge_tag</td>
<td>The Oracle Streams tag to set in the session on the local database before performing any changes to converge the data in the database objects being converged. If non-NULL, then this parameter setting takes precedence over the local_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison. If NULL, then this parameter is ignored, and the local_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison is used. See Security Model for information about security requirement related to this parameter, and see the Oracle Streams Replication Administrator's Guide for more information about tags.</td>
</tr>
<tr>
<td>remote_converge_tag</td>
<td>The Oracle Streams tag to set in the session on the remote database before performing any changes to converge the data in the database objects being converged. If non-NULL, then this parameter setting takes precedence over the remote_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison. If NULL, then this parameter is ignored, and the remote_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison is used. See Security Model for information about security requirement related to this parameter, and see the Oracle Streams Replication Administrator's Guide for more information about tags.</td>
</tr>
</tbody>
</table>

Usage Notes

If one of the database objects being converged is a read-only materialized view, then the converge_options parameter must be set to ensure that the read-only materialized view “wins” in the converge operation. The CONVERGE procedure raises an error if it tries to make changes to a read-only materialized view.

33.7.3 CREATE_COMPARISON Procedure

This procedure creates a comparison.

Syntax

```sql
DBMS_COMPARISON.CREATE_COMPARISON (  
    comparison_name   IN  VARCHAR2,  
    schema_name       IN  VARCHAR2,  
    object_name       IN  VARCHAR2,  
    dblink_name       IN  VARCHAR2,  
    index_schema_name IN  VARCHAR2  DEFAULT NULL,  
    index_name        IN  VARCHAR2  DEFAULT NULL,  
    remote_schema_name IN  VARCHAR2  DEFAULT NULL,
)```

Chapter 33

Summary of DBMS_COMPARISON Subprograms
Parameters

Table 33-6  CREATE_COMPARISON Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_name</td>
<td>The name of the comparison.</td>
</tr>
<tr>
<td>schema_name</td>
<td>The name of the schema that contains the local database object to compare.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the local database object to compare.</td>
</tr>
<tr>
<td>dblink_name</td>
<td>Database link to the remote database. The specified database object in the</td>
</tr>
<tr>
<td></td>
<td>remote database is compared with the database object in the local database.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the comparison is configured to compare two database objects</td>
</tr>
<tr>
<td></td>
<td>in the local database. In this case, parameters that specify the remote</td>
</tr>
<tr>
<td></td>
<td>database object apply to the second database object in the comparison and</td>
</tr>
<tr>
<td></td>
<td>to operations on the second database object. For example, specify the</td>
</tr>
<tr>
<td></td>
<td>second database object in this procedure by using the remote_schema_name</td>
</tr>
<tr>
<td></td>
<td>and remote_object_name parameters.</td>
</tr>
<tr>
<td>index_schema_name</td>
<td>The name of the schema that contains the index.</td>
</tr>
<tr>
<td>index_name</td>
<td>The name of the index.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the system determines the index columns for the</td>
</tr>
<tr>
<td></td>
<td>comparison automatically.</td>
</tr>
<tr>
<td></td>
<td>If the index_schema_name parameter is non-NULL, then the index_name</td>
</tr>
<tr>
<td></td>
<td>parameter must also be non-NULL. Otherwise, an error is raised.</td>
</tr>
<tr>
<td>remote_schema_name</td>
<td>The name of the schema that contains the database object at the remote</td>
</tr>
<tr>
<td></td>
<td>database. Specify a non-NULL value if the schema names are different at</td>
</tr>
<tr>
<td></td>
<td>the two databases.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the schema specified in the schema_name parameter is used.</td>
</tr>
<tr>
<td>remote_object_name</td>
<td>The name of the database object at the remote database. Specify a non-NULL</td>
</tr>
<tr>
<td></td>
<td>value if the database object names are different at the two databases.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the database object specified in the object_name parameter</td>
</tr>
<tr>
<td></td>
<td>is used.</td>
</tr>
</tbody>
</table>
### Table 33-6  (Cont.) CREATE_COMPARISON Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_mode</td>
<td>Specify the default value CMP_COMPARE_MODE_OBJECT. Additional modes might be added in future releases.</td>
</tr>
<tr>
<td>column_list</td>
<td>Specify '*' to include all of the columns in the database objects being compared.</td>
</tr>
<tr>
<td></td>
<td>To compare a subset of columns in the database objects, specify a comma-delimited list of the columns to check. Any columns that are not in the list are ignored during a comparison and convergence.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Usage Notes&quot; for information about columns that are required in the column_list parameter.</td>
</tr>
<tr>
<td>scan_mode</td>
<td>Either CMP_SCAN_MODE_FULL, CMP_SCAN_MODE_RANDOM, CMP_SCAN_MODE_CYCLIC, or CMP_SCAN_MODE CUSTOM.</td>
</tr>
<tr>
<td></td>
<td>If you specify CMP_SCAN_MODE_CUSTOM, then make sure you specify an index using the index_schema_name and index_name parameters. Specifying an index ensures that you can specify the correct min_value and max_value for the lead index column when you run the COMPARE or RECHECK function.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Constants&quot; for information about these constants.</td>
</tr>
<tr>
<td>scan_percent</td>
<td>The percentage of the database object to scan for comparison when the scan_mode parameter is set to either CMP_SCAN_MODE_RANDOM or CMP_SCAN_MODE_CYCLIC. For these scan_mode settings, a non-NULL value that is greater than 0 (zero) and less than 100 is required.</td>
</tr>
<tr>
<td></td>
<td>If NULL and the scan_mode parameter is set to CMP_SCAN_MODE_FULL, then the entire database object is scanned for comparison.</td>
</tr>
<tr>
<td></td>
<td>If NULL and the scan_mode parameter is set to CMP_SCAN_MODE_CUSTOM, then the portion of the database object scanned for comparison is specified when the COMPARE function is run.</td>
</tr>
<tr>
<td></td>
<td>If non-NULL and the scan_mode parameter is set to either CMP_SCAN_MODE_FULL or CMP_SCAN_MODE_CUSTOM, then the scan_percent parameter is ignored.</td>
</tr>
<tr>
<td></td>
<td>Note: When the scan_percent parameter is non-NULL, and the lead index column for the comparison does not distribute the rows in the database object evenly, the portion of the database object that is compared might be smaller or larger than the specified scan_percent value. See DBMS_COMPARISON Operational Notes for more information about the DBMS_COMPARISON package index requirements for the lead index column.</td>
</tr>
<tr>
<td>null_value</td>
<td>The value to substitute for each NULL in the database objects being compared. Specify a value or use the CMP_NULL_VALUE_DEF constant.</td>
</tr>
<tr>
<td></td>
<td>If a column being compared can contain NULLs, then the value specified for this parameter must be different than any non-NULL value in the column. Otherwise, if the value specified for this parameter can appear in the column, some row differences might not be found.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Constants&quot; for information about this constant.</td>
</tr>
</tbody>
</table>
Table 33-6  (Cont.) CREATE_COMPARISON Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_converge_tag</td>
<td>The Oracle Streams tag to set in the session on the local database before performing any changes to converge the data in the database objects being compared. If the local_converge_tag parameter is non-NULL in the CONVERGE procedure when comparison results for this comparison are converged, then the setting in the CONVERGE procedure takes precedence. See CONVERGE Procedure for more information. See the Oracle Streams Replication Administrator's Guide for more information about tags.</td>
</tr>
<tr>
<td>remote_converge_tag</td>
<td>The Oracle Streams tag to set in the session on the remote database before performing any changes to converge the data in the database objects being compared. If the remote_converge_tag parameter is non-NULL in the CONVERGE procedure when comparison results for this comparison are converged, then the setting in the CONVERGE procedure takes precedence. See CONVERGE Procedure for more information. See the Oracle Streams Replication Administrator's Guide for more information about tags.</td>
</tr>
<tr>
<td>max_num_buckets</td>
<td>Specify the maximum number of buckets to use. Specify a value or use the CMP_MAX_NUM_BUCKETS constant. See &quot;Constants&quot; for information about this constant. See &quot;Overview&quot; for information about buckets. Note: If an index column for a comparison is a VARCHAR2 or CHAR column, then the number of buckets might exceed the value specified for the max_num_buckets parameter.</td>
</tr>
<tr>
<td>min_rows_in_bucket</td>
<td>Specify the minimum number of rows in each bucket. Specify a value or use the CMP_MIN_ROWS_IN_BUCKET constant. See &quot;Constants&quot; for information about this constant. See &quot;Overview&quot; for information about buckets.</td>
</tr>
</tbody>
</table>

Usage Notes

This section contains usage notes for the CREATE_COMPARISON procedure.

Usage Notes for the index_schema_name and index_name Parameters

When you specify an index for a comparison with the index_schema_name and index_name parameters, the specified index determines the comparison's index columns and their ordering. The order of the columns in the index determines the index column ordering for the comparison. Therefore, the column in column position 1 in the index is the lead column for the comparison.

The index columns and their ordering affect the details of each SQL statement generated and executed for a comparison. For each SQL statement, the optimizer decides whether to use indexes. If the optimizer decides to use indexes, then the optimizer decides which particular indexes to use. An index specified in column_list parameter might or might not be used.

The columns in the specified index must meet the requirements described in DBMS_COMPARISON Operational Notes. If the index columns do not meet these requirements, then an error is raised.
Note:
If you do not specify an index when you create a comparison, then the CREATE_COMPARISON procedure selects either the primary key, if it exists, or an existing unique index. The procedure never selects a non-unique index. However, if you specify an index, then the CREATE_COMPARISON procedure does not check its uniqueness. Therefore, if you specify a non-unique index, and duplicate index keys exist, then the results might be incorrect when the CONVERGE procedure synchronizes data.

Usage Notes for the column_list Parameter
When the column_list parameter is set to a value other than '*', the following columns are required in the column_list parameter:

- Any columns that are required to meet the index column requirements for the DBMS_COMPARISON package. If the index_name parameter is non-NULL, then the columns in the specified index must be in the column list. If the index_name parameter is NULL, then see DBMS_COMPARISON Operational Notes for information about the DBMS_COMPARISON index requirements.

- If you plan to use the CONVERGE procedure to make changes to a database object based on the comparison, then any columns in this database object that have a NOT NULL constraint but no default value must be included in the column list. If these columns are not included, then the CONVERGE procedure returns an error. See CONVERGE Procedure.

33.7.4 DROP_COMPARISON Procedure

This procedure drops a comparison.

Syntax

```sql
DBMS_COMPARISON.DROP_COMPARISON(
    comparison_name  IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_name</td>
<td>The name of the comparison.</td>
</tr>
</tbody>
</table>
33.7.5 PURGE_COMPARISON Procedure

This procedure purges the comparison results, or a subset of the comparison results, for a comparison.

![Note:]
At least one of the following parameters must be set to `NULL`: `scan_id` or `purge_time`. If both the `scan_id` and `purge_time` parameters are `NULL`, then this procedure purges all comparison results for the comparison.

Syntax

```sql
DBMS_COMPARISON.PURGE_COMPARISON(
    comparison_name  IN  VARCHAR2,
    scan_id          IN  NUMBER     DEFAULT NULL,
    purge_time       IN  TIMESTAMP  DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_name</td>
<td>The name of the comparison.</td>
</tr>
<tr>
<td>scan_id</td>
<td>The scan ID of the scan for which results are purged. The scan ID must identify a root scan. If the scan ID does not identify a root scan, then an error is raised. When a root scan ID is specified, it is purged, and all direct and indirect child scans of the specified root scan are purged. If <code>NULL</code>, then no scan ID is considered when purging comparison results for the comparison. See &quot;Overview&quot; for information about scans.</td>
</tr>
<tr>
<td>purge_time</td>
<td>The date before which results are purged. If <code>NULL</code>, then no date is considered when purging comparison results for the comparison.</td>
</tr>
</tbody>
</table>

33.7.6 RECHECK Function

This function rechecks the differences in a specified scan for a comparison.

This function performs one of the following actions:

- If the specified scan completed successfully the last time it ran, then this function checks the previously identified differences in the scan.
- If the specified scan completed partially, then this function continues to check the database object from the point where the previous scan ended.
Note:

This function does not compare the shared database object for differences that were not recorded in the specified comparison scan. To check for those differences, run the COMPARE function.

See Also:

- Oracle Streams Replication Administrator's Guide
- COMPARE Function

Syntax

```sql
DBMS_COMPARISON.RECHECK(
  comparison_name  IN  VARCHAR2,
  scan_id          IN  NUMBER,
  perform_row_dif  IN  BOOLEAN  DEFAULT FALSE
) RETURN BOOLEAN;
```

Parameters

Table 33-9  RECHECK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison_name</td>
<td>The name of the comparison.</td>
</tr>
<tr>
<td>scan_id</td>
<td>The scan ID of the scan to recheck.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Overview&quot; for more information about specifying a scan ID in this parameter.</td>
</tr>
<tr>
<td>perform_row_dif</td>
<td>If TRUE, then compares each row individually in the database objects being compared after reaching the smallest possible bucket for the comparison.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then compares buckets for differences but does not compare each row individually when differences are found in the smallest possible bucket.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Overview&quot; for information about buckets.</td>
</tr>
</tbody>
</table>

Return Values

This function returns TRUE when no differences are found in the database objects being compared. This function returns FALSE when differences are found in the database objects being compared.
The **DBMS_COMPRESSION** package provides an interface to facilitate choosing the correct compression level for an application.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Data Structures
- Summary of DBMS_COMPRESSION Subprograms

[See Also:](#)

- *Oracle Database Administrator’s Guide*
- *Oracle Database Concepts*
- *Oracle Database SQL Language Reference*
- *Oracle Database Data Warehousing Guide*
- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database Reference*

### 34.1 DBMS_COMPRESSION Overview

The **DBMS_COMPRESSION** package gathers compression-related information within a database environment. This includes tools for estimating compressibility of a table for both partitioned and non-partitioned tables, and gathering row-level compression information on previously compressed tables. This gives the user with adequate information to make compression-related decision.

### 34.2 DBMS_COMPRESSION Security Model

The **DBMS_COMPRESSION** package is defined with AUTHID CURRENT USER, so it executes with the privileges of the current user.
34.3 DBMS_COMPRESSION Constants

The `DBMS_COMPRESSION` package uses constants that can be used for specifying parameter values.

These constants are shown in the following table:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP_NOCOMPRESS</td>
<td>NUMBER</td>
<td>1</td>
<td>No compression</td>
</tr>
<tr>
<td>COMP_ADVANCED</td>
<td>NUMBER</td>
<td>2</td>
<td>Advanced row compression</td>
</tr>
<tr>
<td>COMP_QUERY_HIGH</td>
<td>NUMBER</td>
<td>4</td>
<td>High for query warehouse compression (Hybrid Columnar Compression)</td>
</tr>
<tr>
<td>COMP_QUERY_LOW</td>
<td>NUMBER</td>
<td>8</td>
<td>Low for query warehouse compression (Hybrid Columnar Compression)</td>
</tr>
<tr>
<td>COMP_ARCHIVE_HIGH</td>
<td>NUMBER</td>
<td>16</td>
<td>High archive compression (Hybrid Columnar Compression)</td>
</tr>
<tr>
<td>COMP_ARCHIVE_LOW</td>
<td>NUMBER</td>
<td>32</td>
<td>Low archive compression (Hybrid Columnar Compression)</td>
</tr>
<tr>
<td>COMP_BLOCK</td>
<td>NUMBER</td>
<td>64</td>
<td>Compressed block</td>
</tr>
<tr>
<td>COMP_LOB_HIGH</td>
<td>NUMBER</td>
<td>128</td>
<td>High compression level for LOB operations</td>
</tr>
<tr>
<td>COMP_LOB_MEDIUM</td>
<td>NUMBER</td>
<td>256</td>
<td>Medium compression level for LOB operations</td>
</tr>
<tr>
<td>COMP_LOB_LOW</td>
<td>NUMBER</td>
<td>512</td>
<td>Low compression level for LOB operations</td>
</tr>
<tr>
<td>COMP_INDEX_ADVANCED_HIGH</td>
<td>NUMBER</td>
<td>1024</td>
<td>High compression level for indexes</td>
</tr>
<tr>
<td>COMP_INDEX_ADVANCED_LOW</td>
<td>NUMBER</td>
<td>2048</td>
<td>Low compression level for indexes</td>
</tr>
<tr>
<td>COMP_RATIO_LOB_MINROWS</td>
<td>NUMBER</td>
<td>1000</td>
<td>Minimum required number of LOBs in the object for which LOB compression ratio is to be estimated</td>
</tr>
<tr>
<td>COMP_BASIC</td>
<td>NUMBER</td>
<td>4096</td>
<td>Basic table compression</td>
</tr>
<tr>
<td>COMP_RATIO_LOB_MAXROWS</td>
<td>NUMBER</td>
<td>5000</td>
<td>Maximum number of LOBs used to compute the LOB compression ratio</td>
</tr>
<tr>
<td>COMP_INMEMORY_NOCOMPRESS</td>
<td>NUMBER</td>
<td>8192</td>
<td>In-Memory with no compression</td>
</tr>
<tr>
<td>COMP_INMEMORY_DML</td>
<td>NUMBER</td>
<td>16384</td>
<td>In-Memory compression level for DML</td>
</tr>
<tr>
<td>Constant</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COMP_INMEMORY_QUERY_LOW</td>
<td>NUMBER</td>
<td>32768</td>
<td>In-Memory compression level optimized for query performance</td>
</tr>
<tr>
<td>COMP_INMEMORY_QUERY_HIGH</td>
<td>NUMBER</td>
<td>65536</td>
<td>In-Memory compression level optimized on query performance as well as space saving</td>
</tr>
<tr>
<td>COMP_INMEMORY_CAPACITY_LOW</td>
<td>NUMBER</td>
<td>131072</td>
<td>In-Memory low compression level optimizing for capacity</td>
</tr>
<tr>
<td>COMP_INMEMORY_CAPACITY_HIGH</td>
<td>NUMBER</td>
<td>262144</td>
<td>In-Memory high compression level optimizing for capacity</td>
</tr>
<tr>
<td>COMP_RATIO_MINROWS</td>
<td>NUMBER</td>
<td>1000000</td>
<td>Minimum required number of rows in the object for which HCC ratio is to be estimated</td>
</tr>
<tr>
<td>COMP_RATIO_ALLROWS</td>
<td>NUMBER</td>
<td>-1</td>
<td>To indicate the use of all the rows in the object to estimate HCC ratio</td>
</tr>
<tr>
<td>OBJTYPE_TABLE</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Identifies the object whose compression ratio is estimated as of type table</td>
</tr>
<tr>
<td>OBJTYPE_INDEX</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>Identifies the object whose compression ratio is estimated as of type index</td>
</tr>
</tbody>
</table>

**Note:**
Hybrid columnar compression is a feature of certain Oracle storage systems. See Oracle Database Concepts for more information.

### 34.4 DBMS_COMPRESSION Data Structures

The `DBMS_COMPRESSION` package defines a `RECORD` type and a `TABLE` type.

**RECORD TYPES**

- `COMPREC` Record Type

**TABLE TYPES**

- `COMPRECLIST` Table Type
34.4.1 COMPREC Record Type

The COMPREC record type is a record for calculating an individual index compression ratio on a table.

Syntax

```
TYPE COMPREC IS RECORD(
    ownname           varchar2(255),
    objname           varchar2(255),
    blkcnt_cmp        PLS_INTEGER,
    blkcnt_uncmp      PLS_INTEGER,
    row_cmp           PLS_INTEGER,
    row_uncmp         PLS_INTEGER,
    cmp_ratio         NUMBER,
    objtype           PLS_INTEGER);
```

Fields

Table 34-2 COMPREC Attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of the object owner</td>
</tr>
<tr>
<td>objname</td>
<td>Name of the object</td>
</tr>
<tr>
<td>blkcnt_cmp</td>
<td>Number of blocks used by the compressed sample of the object</td>
</tr>
<tr>
<td>blkcnt_uncmp</td>
<td>Number of blocks used by the uncompressed sample of the object</td>
</tr>
<tr>
<td>row_cmp</td>
<td>Number of rows in a block in compressed sample of the object</td>
</tr>
<tr>
<td>row_uncmp</td>
<td>Number of rows in a block in uncompressed sample of the object</td>
</tr>
<tr>
<td>cmp_ratio</td>
<td>Compression ratio, blkcnt_uncmp divided by blkcnt_cmp</td>
</tr>
<tr>
<td>objtype</td>
<td>Type of the object</td>
</tr>
</tbody>
</table>

34.4.2 COMPRECLIST Table Type

COMPRECLIST is a table type of the COMPREC Record Type.

Syntax

```
TYPE compreclist IS TABLE OF comprec;
```

Related Topics

- **COMPREC Record Type**
  The COMPREC record type is a record for calculating an individual index compression ratio on a table.

34.5 Summary of DBMS_COMPRESSION Subprograms

The DBMS_COMPRESSION package uses the GET_COMPRESSION_RATIO Procedure and GET_COMPRESSION_TYPE Function subprograms.
Table 34-3  DBMS_COMPRESSION Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_COMPRESSION_RATIO Procedure</td>
<td>Analyzes the compression ratio of a table, and gives information about compressibility of a table</td>
</tr>
<tr>
<td>GET_COMPRESSION_TYPE Function</td>
<td>Returns the compression type for a specified row</td>
</tr>
</tbody>
</table>

34.5.1 GET_COMPRESSION_RATIO Procedure

This procedure analyzes the compression ratio of a table or an index, and gives information about compressibility of the object. Various parameters can be provided by the user to selectively analyze different compression types.

Syntax

Get compression ratio for an object (table or index, default is table):

```sql
DBMS_COMPRESSION.GET_COMPRESSION_RATIO (  
  scratchtbsname    IN     VARCHAR2,  
  ownname           IN     VARCHAR2,  
  objname           IN     VARCHAR2,  
  subobjname        IN     VARCHAR2,  
  comptype          IN     NUMBER,  
  blkcnt_cmp        OUT    PLS_INTEGER,  
  blkcnt_uncmp      OUT    PLS_INTEGER,  
  row_cmp           OUT    PLS_INTEGER,  
  row_uncmp         OUT    PLS_INTEGER,  
  cmp_ratio         OUT    NUMBER,  
  comptype_str      OUT    VARCHAR2,  
  subset_numrows    IN     NUMBER DEFAULT COMP_RATIO_MINROWS,  
  objtype           IN     PLS_INTEGER DEFAULT OBJTYPE_TABLE);  
```

Get compression ratio for LOBs:

```sql
DBMS_COMPRESSION.GET_COMPRESSION_RATIO (  
  scratchtbsname    IN     VARCHAR2,  
  tabowner          IN     VARCHAR2,  
  tabname           IN     VARCHAR2,  
  lobname           IN     VARCHAR2,  
  partname          IN     VARCHAR2,  
  comptype          IN     NUMBER,  
  blkcnt_cmp        OUT    PLS_INTEGER,  
  blkcnt_uncmp      OUT    PLS_INTEGER,  
  lobcnt            OUT    PLS_INTEGER,  
  cmp_ratio         OUT    NUMBER,  
  comptype_str      OUT    VARCHAR2,  
  subset_numrows    IN     number DEFAULT COMP_RATIO_LOB_MAXROWS);  
```

Get compression ratio for all indexes on a table. The compression ratios are returned as a collection.

```sql
DBMS_COMPRESSION.GET_COMPRESSION_RATIO (  
  scratchtbsname    IN     VARCHAR2,  
  ownname           IN     VARCHAR2,  
  tabname           IN     VARCHAR2,  
  comptype          IN     NUMBER,  
```
Table 34-4  GET_COMPRESSION_RATIO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scratchbsname</td>
<td>Temporary scratch tablespace that can be used for analysis</td>
</tr>
<tr>
<td>ownname/tabowner</td>
<td>Schema of the table to analyze</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to analyze</td>
</tr>
<tr>
<td>objname</td>
<td>Name of the object</td>
</tr>
<tr>
<td>subobjname</td>
<td>Name of the partition or sub-partition of the object</td>
</tr>
<tr>
<td>comtype</td>
<td>Compression types for which analysis should be performed</td>
</tr>
<tr>
<td>blkcnt_cmp</td>
<td>Number of blocks used by compressed sample of the table</td>
</tr>
<tr>
<td>blkcnt_uncmp</td>
<td>Number of blocks used by uncompressed sample of the table</td>
</tr>
<tr>
<td>row_cmp</td>
<td>Number of rows in a block in compressed sample of the table</td>
</tr>
<tr>
<td>row_uncmp</td>
<td>Number of rows in a block in uncompressed sample of the table</td>
</tr>
<tr>
<td>cmp_ratio</td>
<td>Compression ratio, ( \frac{\text{blkcnt}<em>\text{uncmp}}{\text{blkcnt}</em>\text{cmp}} )</td>
</tr>
<tr>
<td>comtype_str</td>
<td>String describing the compression type</td>
</tr>
<tr>
<td>subset_numrows</td>
<td>Number of rows sampled to estimate compression ratio.</td>
</tr>
<tr>
<td>objtype</td>
<td>Type of the object, either OBJTYPE_TABLE or OBJTYPE_INDEX</td>
</tr>
<tr>
<td>lobname</td>
<td>Name of the LOB column</td>
</tr>
<tr>
<td>partname</td>
<td>In case of partitioned tables, the related partition name</td>
</tr>
<tr>
<td>lobcnt</td>
<td>Number of lobs actually sampled to estimate compression ratio</td>
</tr>
<tr>
<td>index_cr</td>
<td>List of indexes and their estimated compression ratios</td>
</tr>
</tbody>
</table>

Usage Notes

The procedure creates different tables in the scratch tablespace and runs analysis on these objects. It does not modify anything in the user-specified tables.

34.5.2 GET_COMPRESSION_TYPE Function

This function returns the compression type for a specified row. If the row is chained, the function returns the compression type of the head piece only, and does not
examine the intermediate or the tail piece since head pieces can be differently compressed.

Syntax

```sql
DBMS_COMPRESSION.GET_COMPRESSION_TYPE ( 
    ownname    IN    VARCHAR2,
    tabname    IN    VARCHAR2,
    row_id     IN    ROWID,
    subobjname IN    VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Parameters

Table 34-5  GET_COMPRESSION_TYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema name of the table</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of table</td>
</tr>
<tr>
<td>rowid</td>
<td>Rowid of the row</td>
</tr>
<tr>
<td>subobjname</td>
<td>Name of the table partition or subpartition</td>
</tr>
</tbody>
</table>

Return Values

Flag to indicate the compression type (see Table 34-1).
The DBMS_CONNECTION_POOL package provides an interface to manage Database Resident Connection Pool.

See Also:

Oracle Database Concepts for more information on “Database Resident Connection Pooling”

This chapter contains the following topic:

• Summary of DBMS_CONNECTION_POOL Subprograms

35.1 Summary of DBMS_CONNECTION_POOL Subprograms

This table lists the DBMS_CONNECTION_POOL subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_PARAM Procedure</td>
<td>Alters a specific configuration parameter as a standalone unit and does not affect other parameters</td>
</tr>
<tr>
<td>CONFIGURE_POOL Procedure</td>
<td>Configures the pool with advanced options</td>
</tr>
<tr>
<td>START_POOL Procedure</td>
<td>Starts the pool for operations. It is only after this call that the pool could be used by connection clients for creating sessions</td>
</tr>
<tr>
<td>STOP_POOL Procedure</td>
<td>Stops the pool and makes it unavailable for the registered connection clients</td>
</tr>
<tr>
<td>RESTORE_DEFAULTS Procedure</td>
<td>Restores the pool to default settings</td>
</tr>
</tbody>
</table>

35.1.1 ALTER_PARAM Procedure

This procedure alters a specific configuration parameter as a standalone unit and does not affect other parameters.

Syntax

```sql
DBMS_CONNECTION_POOL.ALTER_PARAM (  
  pool_name  IN  VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL',  
  param_name IN  VARCHAR2,  
  param_value IN  VARCHAR2);
```
Parameters

Table 35-2 ALTER_PARAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Pool to be configured. Currently only the default pool name is supported.</td>
</tr>
<tr>
<td>param_name</td>
<td>Any parameter name from CONFIGURE_POOL.</td>
</tr>
<tr>
<td>param_value</td>
<td>Parameter value for param_name.</td>
</tr>
</tbody>
</table>

See Also:
For the list and description of all the database resident connection pooling parameters that can be configured using this procedure, see the Oracle Database Administrator’s Guide.

Exceptions

Table 35-3 ALTER_PARAM Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-56500</td>
<td>Connection pool not found</td>
</tr>
<tr>
<td>ORA-56504</td>
<td>Invalid connection pool configuration parameter name</td>
</tr>
<tr>
<td>ORA-56505</td>
<td>Invalid connection pool configuration parameter value</td>
</tr>
<tr>
<td>ORA-56507</td>
<td>Connection pool alter configuration failed</td>
</tr>
</tbody>
</table>

Examples

```sql
DBMS_CONNECTION_POOL.ALTER_PARAM(
    'SYS_DEFAULT_CONNECTION_POOL', 'MAX_LIFETIME_SESSION', '120');
```

35.1.2 CONFIGURE_POOL Procedure

This procedure configures the pool with advanced options.

Syntax

```sql
DBMS_CONNECTION_POOL.CONFIGURE_POOL (  
    pool_name                IN VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL',
    minsize                  IN NUMBER   DEFAULT 4,
    maxsize                  IN NUMBER   DEFAULT 40,
    incrsize                 IN NUMBER   DEFAULT 2,
    session_cached_cursors   IN NUMBER   DEFAULT 20,
    inactivity_timeout       IN NUMBER   DEFAULT 300,
    max_think_time           IN NUMBER   DEFAULT 120,
    max_use_session          IN NUMBER   DEFAULT 500000,
    max_lifetime_session     IN NUMBER   DEFAULT 86400,
    max_txn_think_time       IN NUMBER);  
```
Parameters

Table 35-4  CONFIGURE_POOL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Pool to be configured. Currently only the default pool name is supported.</td>
</tr>
<tr>
<td>minsize</td>
<td>Minimum number of pooled servers in the pool</td>
</tr>
<tr>
<td>maxsize</td>
<td>Maximum allowed pooled servers in the pool</td>
</tr>
<tr>
<td>incrsize</td>
<td>Pool would increment by this number of pooled server when pooled server are unavailable at application request time</td>
</tr>
<tr>
<td>session_cached_cursors</td>
<td>Turn on SESSION_CACHED_CURSOR for all connections in the pool. This is an existing init.ora parameter</td>
</tr>
<tr>
<td>inactivity_timeout</td>
<td>TTL (Time to live) for an idle session in the pool. This parameter helps to shrink the pool when it is not used to its maximum capacity. If a connection remains in the pool idle for this time, it is killed.</td>
</tr>
<tr>
<td>max_think_time</td>
<td>The maximum time of inactivity, in seconds, for a client after it obtains a pooled server from the pool with no open transactions in it. After obtaining a pooled server from the pool, if the client application does not issue a database call for the time specified by MAX_THINK_TIME, the pooled server is freed and the client connection is terminated.</td>
</tr>
<tr>
<td>max_use_session</td>
<td>Maximum number of times a connection can be taken and released to the pool.</td>
</tr>
<tr>
<td>max_lifetime_session</td>
<td>TTL (Time to live) for a pooled session</td>
</tr>
<tr>
<td>max_txn_think_time</td>
<td>The maximum time of inactivity, in seconds, for a client after it obtains a pooled server from the pool with an open transaction. After obtaining the pooled server from the pool, if the client application does not issue a database call for the time specified by MAX_TXN_THINK_TIME, then the pooled server is freed, and the client connection is terminated. The default value of this parameter is the value of the MAX_THINK_TIME parameter. Applications can set the value of the MAX_TXN_THINK_TIME parameter to a value higher than the MAX_THINK_TIME value to allow more time for the connections with open transactions.</td>
</tr>
</tbody>
</table>

Exceptions

Table 35-5  CONFIGURE_POOL Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-56500</td>
<td>Connection pool not found</td>
</tr>
<tr>
<td>ORA-56507</td>
<td>Connection pool alter configuration failed</td>
</tr>
</tbody>
</table>

Usage Notes

- All expressions of time are in seconds
- All of the parameters should be set based on statistical request patterns.
minsize should be set keeping in mind that it puts a lower bound on server resource consumption. This is to prevent the timeout from dragging the pool too low, because of a brief period of inactivity.

maxsize should be set keeping in mind that it puts an upper bound on concurrency and response-times and also server resource consumption.

session_cached_cursors is typically set to the number of most frequently used statements. It occupies cursor resource on the server.

In doubt, do not set the increment and inactivity_timeout. The pool will have reasonable defaults.

max_use_session and max_lifetime_session allow for software rejuvenation or defensive approaches to potential bugs, leaks, accumulations, and like problems, by getting brand new sessions once in a while.

The connection pool reserves 5% of the pooled servers for authentication, and at least one pooled server is always reserved for authentication. When setting the maxsize parameter, ensure that there are enough pooled servers for both authentication and connections.

35.1.3 START_POOL Procedure

This procedure starts the pool for operations. It is only after this call that the pool could be used by connection classes for creating sessions.

Syntax

DBMS_CONNECTION_POOL.START_POOL (pool_name  IN  VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL');

Parameters

Table 35-6 START_POOL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Pool to be started. Currently only the default pool name is supported.</td>
</tr>
</tbody>
</table>

Exceptions

Table 35-7 START_POOL Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-56500</td>
<td>Connection pool not found</td>
</tr>
<tr>
<td>ORA-56501</td>
<td>Connection pool startup failed</td>
</tr>
</tbody>
</table>

Usage Notes

If the instance is restarted (shutdown followed by startup), the pool is automatically started.
### 35.1.4 STOP_POOL Procedure

This procedure stops the pool and makes it unavailable for the registered connection classes.

**Syntax**

```sql
DBMS_CONNECTION_POOL.STOP_POOL ( 
    pool_name IN VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Pool to be stopped. Currently only the default pool name is supported.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-56500</td>
<td>Connection pool not found</td>
</tr>
<tr>
<td>ORA-56506</td>
<td>Connection pool shutdown failed</td>
</tr>
</tbody>
</table>

**Usage Notes**

This stops the pool and takes it offline. This does not destroy the persistent data (such as, the pool name and configuration parameters) associated with the pool.

### 35.1.5 RESTORE_DEFAULTS Procedure

This procedure restores the pool to default settings.

**Syntax**

```sql
DBMS_CONNECTION_POOL.RESTORE_DEFAULTS ( 
    pool_name IN VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Pool to be restored. Currently only the default pool name is supported.</td>
</tr>
</tbody>
</table>
Exceptions

Table 35-11  RESTORE_DEFAULTS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-56500</td>
<td>Connection pool not found</td>
</tr>
<tr>
<td>ORA-56507</td>
<td>Connection pool alter configuration failed</td>
</tr>
</tbody>
</table>

Usage Notes

If the instance is restarted (shutdown followed by startup), the pool is automatically started.
The DBMS_CQ_NOTIFICATION package is part of the database change notification feature that provides the functionality to create registration on queries designated by a client application and so to receive notifications in response to DML or DDL changes on the objects associated with the queries. The notifications are published by the database when the DML or DDL transaction commits.

See Also:
Oracle Database Development Guide regarding implementing database change notification.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Operational Notes
- Examples
- Data Structures
- Summary of DBMS_CQ_NOTIFICATION Subprograms

36.1 DBMS_CQ_NOTIFICATION Overview

The DBMS_CQ_NOTIFICATION package provides PL/SQL based registration interfaces. A client can use this interface to create registrations on queries based on objects of interest and specify a PL/SQL callback handler to receive notifications. In case of object level registration, when a transaction changes any of the objects associated with the registered queries and COMMITs, the PL/SQL callback, specified during registration for those objects, is invoked. The application can define client-specific processing inside the implementation of its PL/SQL callback handler.

The interface lets you define a registration block (using a mechanism similar to a BEGIN-END block). The recipient of notifications namely the name of the PL/SQL callback handler and a few other registration properties like time-outs can be specified during the BEGIN phase. Any queries executed subsequently (inside the registration block) are considered "interesting queries" and objects referenced by those queries during query execution are registered. The registration is completed by ENDing the registration block. The registration block lets you create new registrations or add objects to existing registrations.

When a registration is created through the PL/SQL interface, a unique registration ID is assigned to the registration by the RDBMS. The client application can use the registration ID
to keep track of registrations created by it. When a notification is published by the 
RDBMS, the registration ID will be part of the notification.

**Typical Applications**

This functionality is useful for example to applications that cache query result sets on 
mostly read-only objects in the mid-tier to avoid network round trips to the database. 
Such an application can create a registration on the queries it is interested in caching. 
On changes to objects referenced inside those queries, the database publishes a 
notification when the underlying transaction commits. In response to the notification, 
the mid-tier application can refresh its cache by re-executing the query/queries.

### 36.2 DBMS_CQ_NOTIFICATION Security Model

The `DBMS_CQ_NOTIFICATION` package requires that the user have the `CHANGE` 
`NOTIFICATION` system privilege in order to receive notifications, and be granted 
`EXECUTE` privilege on the `DBMS_CQ_NOTIFICATION` package.

In addition the user is required to have `SELECT` or `READ` privileges on all objects to be 
registered. Note that if the `SELECT` or `READ` privilege on an object was granted at the 
time of registration creation but lost subsequently (due to a revoke), then the 
registration will be purged and a notification to that effect will be published.

### 36.3 DBMS_CQ_NOTIFICATION Constants

`DBMS_CQ_NOTIFICATION` constants are used as flag parameters either during 
registration or when received during the notification.

The `DBMS_CQ_NOTIFICATION` package has sets of constants:

- `EVENT_STARTUP`, `EVENT_SHUTDOWN`, `EVENT_SHUTDOWN_ANY`, `EVENT_DEREG` describe the 
type of the notification published by the database.
- `INSERTOP`, `DELETEOP`, `UPDATEOP`, `ALTEROP`, `DROP` and `UNKNOWNOP` describe the type 
of operation on a table (during a notification published by the database).
- `QOS_RELIABLE`, `QOS_DEREG_NFY`, `QOS_ROWIDs` describe registration Quality of Service 
properties that the client requires. These are specified during registration.

The constants are shown in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL_OPERATIONS</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>Interested in being notified on all operations, specified as a parameter during registration</td>
</tr>
<tr>
<td>ALL_ROWS</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>All rows within the table may have been potentially modified</td>
</tr>
<tr>
<td>EVENT_STARTUP</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Instance startup notification</td>
</tr>
<tr>
<td>EVENT_SHUTDOWN</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Instance shutdown notification</td>
</tr>
<tr>
<td>EVENT_SHUTDOWN_ANY</td>
<td>BINARY_INTEGER</td>
<td>3</td>
<td>Any instance shutdown when running Oracle Real Application Clusters (Oracle RAC)</td>
</tr>
</tbody>
</table>
Table 36-1  (Cont.) DBMS_CQ_NOTIFICATION Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_DEREG</td>
<td>BINARY_INTEGER</td>
<td>5</td>
<td>Registration has been removed</td>
</tr>
<tr>
<td>EVENT_OBJCHANGE</td>
<td>BINARY_INTEGER</td>
<td>6</td>
<td>Notification for object change</td>
</tr>
<tr>
<td>EVENT_QUERYCHANGE</td>
<td>BINARY_INTEGER</td>
<td>7</td>
<td>Notification for query result set change</td>
</tr>
<tr>
<td>INSERTOP</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Insert operation</td>
</tr>
<tr>
<td>UPDATEOP</td>
<td>BINARY_INTEGER</td>
<td>4</td>
<td>Update operation</td>
</tr>
<tr>
<td>DELETETOP</td>
<td>BINARY_INTEGER</td>
<td>8</td>
<td>Delete operation</td>
</tr>
<tr>
<td>ALTEROP</td>
<td>BINARY_INTEGER</td>
<td>16</td>
<td>Table altered</td>
</tr>
<tr>
<td>DROPPOP</td>
<td>BINARY_INTEGER</td>
<td>32</td>
<td>Table dropped</td>
</tr>
<tr>
<td>UNKNOWNOP</td>
<td>BINARY_INTEGER</td>
<td>64</td>
<td>Unknown operation</td>
</tr>
<tr>
<td>QOS_RELIABLE</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Reliable or persistent notification. Also implies that the notifications will be inserted into the persistent storage atomically with the committing transaction that results in an object change.</td>
</tr>
<tr>
<td>QOS_DEREG_NFY</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Purge registration on first notification</td>
</tr>
<tr>
<td>QOS_ROWIDS</td>
<td>BINARY_INTEGER</td>
<td>4</td>
<td>Require rowids of modified rows</td>
</tr>
<tr>
<td>QOS_QUERY</td>
<td>BINARY_INTEGER</td>
<td>8</td>
<td>Register at query granularity</td>
</tr>
<tr>
<td>QOS_BEST_EFFORT</td>
<td>BINARY_INTEGER</td>
<td>16</td>
<td>Best effort evaluation</td>
</tr>
<tr>
<td>NTFN_GROUPING_CLASS</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Group notifications by time</td>
</tr>
<tr>
<td>NTFN_GROUPING_TYPE_</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Summary grouping of notifications</td>
</tr>
<tr>
<td>NTFN_GROUPING_TYPE_</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Last notification in the group</td>
</tr>
<tr>
<td>NTFN_GROUPING_FOREVER</td>
<td>BINARY_INTEGER</td>
<td>-1</td>
<td>Repeat notifications forever</td>
</tr>
</tbody>
</table>

36.4 DBMS_CQ_NOTIFICATION Operational Notes

The following are DBMS_CQ_NOTIFICATION operational notes.

Object Level Registration

- The notifications are published by the database when a transaction changes the registered objects and COMMITs.
- All objects referenced in the queries executed inside the registration block starting from the previous NEW_REG_START or ENABLE_REG to REG_END are considered interesting objects and added to the registration.
Query Result Change Registration

- The notifications are published by the database when a transaction changes the result set of the registered query and COMMITs.

Troubleshooting

If you have created a registration and seem to not receive notifications when the underlying tables are changed, please check the following.

- Is the job_queue_processes parameter set to a nonzero value? This parameter needs to be configured to a nonzero value in order to receive PL/SQL notifications through the handler.
- Are the registrations being created as a non-SYS user?
- If you are attempting DML changes on the registered object, are you COMMITting the transaction? Please note that the notifications are transactional and will be generated when the transaction COMMITs.
- It maybe possible that there are run-time errors during the execution of the PL/SQL callback due to implementation errors. If so, they would be logged to the trace file of the JOBQ process that attempts to execute the procedure. The trace file would be usually named <ORACLE_SID>_j*_<PID>.trc.

For example, if the ORACLE_SID is 'dbs1' and the process is 12483, the trace file might be named 'dbs1_j000_12483.trc.'

Suppose a registration is created with 'chnf_callback as the notification handler and with registration_id 100. Let us suppose the user forgets to define the chnf_callback procedure. Then the JOBQ trace file might contain a message of the following form.

Runtime error during execution of PL/SQL cbk chnf_callback for reg CHNF100
  Error in PLSQL notification of msgid:
  Queue :
  Consumer Name :
  PLSQL function :chnf_callback
  Exception Occured, Error msg:
  ORA-00604: error occurred at recursive SQL level 2
  ORA-06550: line 1, column 7:
  PLS-00201: identifier 'CHNF_CALLBACK' must be declared
  ORA-06550: line 1, column 7:
  PL/SQL: Statement ignored

See Also:

For more information about troubleshooting Database Change Notification, see Oracle Database Development Guide.

36.5 DBMS_CQ_NOTIFICATION Examples

Suppose that a mid-tier application has a lot of queries on the HR.EMPLOYEES table. If the EMPLOYEES table is infrequently updated, it can obtain better performance by caching rows from the table because that would avoid a round-trip to the backend.
database server and server side execution latency. Let us assume that the application has implemented a mid-tier HTTP listener that listens for notifications and updates the mid-tier cache in response to a notification.

Object Change Registration Example

The DBMS_CQ_NOTIFICATION package can be utilized in this scenario to send notifications about changes to the table by means of the following steps:

1. Implement a mid-tier listener component of the cache management system (for example, using HTTP) that listens to notification messages sent from the database and refreshes the mid-tier cache in response to the notification.

2. Create a server side stored procedure to process notifications

```sql
CONNECT system;
Enter password: password
GRANT CHANGE NOTIFICATION TO hr;
GRANT EXECUTE ON DBMS_CQ_NOTIFICATION TO hr;

Rem Enable job queue processes to receive notifications.
ALTER SYSTEM SET "job_queue_processes"=2;

CONNECT hr;
Enter password: password
Rem Create a table to record notification events
CREATE TABLE nfevents(regid number, event_type number);

Rem create a table to record changes to registered tables
CREATE TABLE nftablechanges(regid number, table_name varchar2(100),
                            table_operation number);

Rem create a table to record rowids of changed rows.
CREATE TABLE nfrowchanges(regid number, table_name varchar2(100),
                          row_id varchar2(30));

Rem Create a PL/SQL callback handler to process notifications.
CREATE OR REPLACE PROCEDURE chnf_callback(ntfnds IN SYS.CHNF$_DESC) IS
  regid           NUMBER;
  tbname          VARCHAR2(60);
  event_type      NUMBER;
  numtables       NUMBER;
  operation_type  NUMBER;
  numrows         NUMBER;
  row_id          VARCHAR2(20);
BEGIN
  regid      := ntfnds.registration_id;
  numtables  := ntfnds.numtables;
  event_type := ntfnds.event_type;
  INSERT INTO nfevents VALUES(regid, event_type);
  IF (event_type = DBMS_CQ_NOTIFICATION.EVENT_OBJCHANGE) THEN
    FOR i IN 1..numtables LOOP
      tbname          := ntfnds.table_desc_array(i).table_name;
      operation_type  := ntfnds.table_desc_array(I).Opflags;
      INSERT INTO nftablechanges VALUES(regid, tbname, operation_type);
      /* Send the table name and operation_type to client side listener using */
      /* UTL_HTTP */
      /* If interested in the rowids, obtain them as follows */
      IF (bitand(operation_type, DBMS_CQ_NOTIFICATION.ALL_ROWS) = 0) THEN
        numrows := ntfnds.table_desc_array(i).numrows;
    END LOOP;
  END IF;
END;
```

Chapter 36

DBMS_CQ_NOTIFICATION Examples

36-5
ELSE
  numrows := 0;  /* ROWID INFO NOT AVAILABLE */
END IF;
/* The body of the loop is not executed when numrows is ZERO */
FOR j IN 1..numrows LOOP
  Row_id := ntfnds.table_desc_array(i).row_desc_array(j).row_id;
  INSERT INTO nfrowchanges VALUES(regid, tbname, Row_id);
  /* optionally Send out row_ids to client side listener using
  UTL_HTTP; */
END LOOP;
END LOOP;
END IF;
COMMIT;
END;
/

In Step 2 we can send as much information about the invalidation as the mid-tier application needs based on the information obtained from the notification descriptor.

Notes
a. In the above example, a registration was created on the EMPLOYEES table with 'chnf_callback' as the PL/SQL handler for notifications. During registration, the client specified reliable notifications (QOS_RELIABLE) and rowid notifications (QOS_ROWIDS)
b. The handler accesses the table descriptor array from the notification descriptor only if the notification type is of EVENT_OBJCHANGE. In all other cases (e.g. EVENT_DEREG, EVENT_SHUTDOWN), the table descriptor array should not be accessed.
c. The handler accesses the row descriptor array from the table notification descriptor only if the ALL_ROWS bit is not set in the table operation flag. If the ALL_ROWS bit is set in the table operation flag, then it means that all rows within the table may have been potentially modified. In addition to operations like TRUNCATE that affect all rows in the tables, this bit may also be set if individual rowids have been rolled up into a FULL table invalidation.

This can occur if too many rows were modified on a given table in a single transaction (more than 80) or the total shared memory consumption due to rowids on the RDBMS is determined too large (exceeds 1% of the dynamic shared pool size). In this case, the recipient must conservatively assume that the entire table has been invalidated and the callback/application must be able to handle this condition.

Also note that the implementation of the user defined callback is up to the developer. In the above example, the callback was used to record event details into database tables. The application can additionally send the notification details to a mid-tier HTTP listener of its cache management system (as in the example) using UTL_HTTP. The listener could then refresh its cache by querying from the back-end database.

3. Create a registrations on the tables that we wish to be notified about. We pass in the previously defined procedure name (chnf_callback) as the name of the server side PL/SQL procedure to be executed when a notification is generated.
Rem Create a REGISTRATION on the EMPLOYEES TABLE
DECLARE
  REGDS SYS.CHNF$ _REG_INFO;
  regid NUMBER;
  mgr_id NUMBER;
  dept_id NUMBER;
  qosflags NUMBER;
BEGIN
  qosflags := DBMS_CQ_NOTIFICATION.QOS_RELIABLE +
              DBMS_CQ_NOTIFICATION.QOS_ROWIDS;
  REGDS := SYS.CHNF$ _REG_INFO ("chnf_callback", qosflags, 0,0,0);
  regid := DBMS_CQ_NOTIFICATION.NEW_REG_START (REGDS);
  SELECT manager_id INTO mgr_id FROM EMPLOYEES WHERE employee_id = 200;
  DBMS_CQ_NOTIFICATION.REG_END;
END;
/

Once the registration is created in Step 3 above, the server side PL/SQL procedure
defined in Step 2 is executed in response to any COMMITed changes to the HR.EMPLOYEES
table. As an example, let us assume that the following update is performed on the
employees table.
UPDATE employees SET salary=salary*1.05 WHERE employee_id=203;COMMIT;

Once the notification is processed, you will find rows which might look like the following in
the nfevents, nftablechanges and nfrowchanges tables.

SQL> SELECT * FROM nfevents;
REGID EVENT_TYPE
---------------------------
   20045       6

SQL> SELECT * FROM nftablechanges;
REGID TABLE_NAME TABLE_OPERATION
-------------------------------------------
   20045    HR.EMPLOYEES          4

SQL> select * from nfrowchanges;
REGID TABLE_NAME ROW_ID
-------------------------------------------
   20045    HR.EMPLOYEES AAAKB/AABAAAJ8zAAF

Query Result Change Registration Example

1. Creating a Callback

CONNECT system;
Enter password: password
GRANT CHANGE NOTIFICATION TO hr;
GRANT EXECUTE ON DBMS_CQ_NOTIFICATION TO hr;
CONNECT hr;
Enter password: password
Rem Create a table to record notification events
CREATE TABLE nfevents(regid NUMBER, event_type NUMBER);
Rem Create a table to record notification queries
CREATE TABLE nfqueries (qid NUMBER, qop NUMBER);

Rem Create a table to record changes to registered tables
CREATE TABLE nftablechanges(
  qid NUMBER,
  table_name VARCHAR2(100),
  table_operation NUMBER);

Rem Create a table to record rowids of changed rows.
CREATE TABLE nfrowchanges(
  qid NUMBER,
  table_name VARCHAR2(100),
  row_id VARCHAR2(2000));

CREATE OR REPLACE PROCEDURE chnf_callback
  (ntfnds IN CQ_NOTIFICATION DESCRIPTOR)
IS
  regid           NUMBER;
  tbname          VARCHAR2(60);
  event_type      NUMBER;
  numtables       NUMBER;
  operation_type  NUMBER;
  numrows         NUMBER;
  numqueries      NUMBER;
  qid NUMBER;
  qop NUMBER;
BEGIN
  regid := ntfnds.registration_id;
  event_type := ntfnds.event_type;
  INSERT INTO nfevents VALUES(regid, event_type);
  numqueries :=0;
  IF (event_type = DBMS_CQ_NOTIFICATION.EVENT_QUERYCHANGE) THEN
    numqueries := ntfnds.query_desc_array.count;
    FOR i in 1..numqueries LOOP
      qid := ntfnds.QUERY_DESC_ARRAY(i).queryid;
      qop := ntfnds.QUERY_DESC_ARRAY(i).queryop;
      INSERT INTO nfqueries VALUES(qid, qop);
      numtables := 0;
      numtables := ntfnds.QUERY_DESC_ARRAY(i).table_desc_array.count;
      FOR j IN 1..numtables LOOP
        tbname :=
          ntfnds.QUERY_DESC_ARRAY(i).table_desc_array(j).table_name;
        operation_type :=
          ntfnds.QUERY_DESC_ARRAY(i).table_desc_array(j).Opflags;
        INSERT INTO nftablechanges VALUES(qid, tbname, operation_type);
        IF (bitand(operation_type, DBMS_CQ_NOTIFICATION.ALL_ROWS) = 0)
          THEN
            numrows :=
              ntfnds.query_desc_array(i).table_desc_array(j).numrows;
          ELSE
            numrows :=0;   /* ROWID INFO NOT AVAILABLE */
          END IF;
        FOR k IN 1..numrows LOOP
          Row_id :=
            ntfnds.query_desc_array(i).table_desc_array(j).row_desc_array(k).row_id;
        END LOOP;
      END LOOP;
    END LOOP;
  END IF;
END;
2. Creates a query registration

DECLARE
    reginfo    cq_notification$_reg_info;
    mgr_id     NUMBER;
    dept_id    NUMBER;
    v_cursor   SYS_REFCURSOR;
    regid      NUMBER;
    qosflags   NUMBER;
BEGIN
    /* Register two queries for result-set-change notifications: */

    /* 1. Construct registration information. */
    'chnf_callback' is name of notification handler.
    QOS_QUERY specifies result-set-change notifications. */

    qosflags := DBMS_CQ_NOTIFICATION.QOS_QUERY +
                DBMS_CQ_NOTIFICATION.QOS_ROWIDS;

    reginfo := cq_notification$_reg_info('chnf_callback', qosflags, 0, 0, 0);

    /* 2. Create registration */

    regid := DBMS_CQ_NOTIFICATION.NEW_REG_START(reginfo);

    OPEN v_cursor FOR
        SELECT DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID, manager_id
            FROM HR.employees
            WHERE employee_id = 7902;
    CLOSE v_cursor;

    OPEN v_cursor for
        SELECT DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID, department_id
            FROM HR.departments
            WHERE department_name = 'IT';
    CLOSE v_cursor;

    DBMS_CQ_NOTIFICATION.REG_END;
END;
/

3. After creating the query registrations, the output from USER_CQ_NOTIFICATION QUERIES would appear as follows.

SQL> SELECT queryid, regid, to_char(querytext)
            FROM user_cq_notification_queries;

    QUERYID REGID
    --------- -------
    TO_CHAR(QUERYTEXT)
Now, let us perform an UPDATE that changes the result of the query with queryid 22
by renaming the department with name 'IT' to FINANCE.

SQL> update departments set department_name = 'FINANCE' where department_name = 'IT';
1 row updated.
SQL> commit;
Commit complete.

Now we can query the notifications that we recorded in the callback.

SQL> select * from nfevents;
REGID EVENT_TYPE
---------- ----------
  61         7

Event type 7 corresponds to EVENT_QUERYCHANGE

SQL> select * from nfqueries;
QID QOP
---------- ----------
  42        7

Event type 7 corresponds to EVENT_QUERYCHANGE

SQL> select * from nftablechanges;
SQL> select * from nftablechanges;

REGID
----------
TABLE_NAME
-------------------------------------------------------------------------------------------------------------------------------------
---
TABLE_OPERATION
----------
  42
HR.DEPARTMENTS
  4

TABLE_OPERATION 4 corresponds to UPDATEOP
36.6 DBMS_CQ_NOTIFICATION Data Structures

The DBMS_CQ_NOTIFICATION package defines several OBJECT types.

OBJECT Types

- CQ_NOTIFICATION$_DESCRIPTOR Object Type
- CQ_NOTIFICATION$_QUERY Object Type
- CQ_NOTIFICATION$_QUERY_ARRAY Object (Array) Type
- CQ_NOTIFICATION$_TABLE Object Type
- CQ_NOTIFICATION$_TABLE_ARRAY Object (Array) Type
- CQ_NOTIFICATION$_ROW Object Type
- CQ_NOTIFICATION$_ROW_ARRAY Object (Array) Type
- CQ_NOTIFICATION$_REG_INFO Object Type

36.6.1 CQ_NOTIFICATION$_DESCRIPTOR Object Type

This is the top level change notification descriptor type. It is a synonym for the SYS.CHNF$_DESC type.

Syntax

```
TYPE SYS.CHNF$_DESC IS OBJECT(
   registration_id    NUMBER,
   transaction_id     RAW(8),
   dbname             VARCHAR2(30),
   event_type         NUMBER,
   numtables          NUMBER,
   table_desc_array   CQ_NOTIFICATION$_TABLE_ARRAY,
   query_desc_array   CQ_NOTIFICATION$_QUERY_ARRAY);
```

Attributes

**Table 36-2 CQ_NOTIFICATION$_DESCRIPTOR Object Type**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registration_id</td>
<td>Registration ID returned during registration</td>
</tr>
</tbody>
</table>

Chapter 36

DBMS_CQ_NOTIFICATION Data Structures
Table 36-2 (Cont.) CQ_NOTIFICATION$_DESCRIPTOR Object Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td>Transaction ID. transaction_id of the transaction that made the change. Will be NULL unless the event_type is EVENT_OBJCHANGE or EVENT_QUERYCHANGE.</td>
</tr>
<tr>
<td>dbname</td>
<td>Name of database</td>
</tr>
<tr>
<td>event_type</td>
<td>Database event associated with the notification. Can be one of EVENT_OBJCHANGE (change to a registered object), EVENT_STARTUP, or EVENT_QUERYCHANGE, EVENT_SHUTDOWN or EVENT_DEREG (registration has been removed due to a timeout or other reason)</td>
</tr>
<tr>
<td>numtables</td>
<td>Number of modified tables. Will be NULL unless the event_type is EVENT_OBJCHANGE.</td>
</tr>
<tr>
<td>table_desc_array</td>
<td>Array of table descriptors. Will be NULL unless the event_type is EVENT_OBJCHANGE.</td>
</tr>
<tr>
<td>query_desc_array</td>
<td>Array of queries changed. This will be NULL unless event_type is EVENT_QUERYCHANGE</td>
</tr>
</tbody>
</table>

36.6.2 CQ_NOTIFICATION$_QUERY Object Type

The object type describes the changes to a query result caused by an event such as a transaction commit.

An array of CQ_NOTIFICATION$_QUERY descriptors is embedded inside the top level notification descriptor (CQ_NOTIFICATION$_DESCRIPTOR) for events of type EVENT_QUERYCHANGE. The array corresponds to the SET of queryids which were invalidated as a result of the event.

This is a synonym for the base type SYS.CNF$_QDESC.

Syntax

```
TYPE SYS.CNF$_QDESC IS OBJECT (
    queryid           NUMBER,
    queryop           NUMBER,
    table_desc_array  CQ_NOTIFICATION$_TABLE_ARRAY);
```

Attributes

Table 36-3 TYPE SYS.CQ_NOTIFICATION$_QUERY Object Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryid</td>
<td>QueryId of the changed query</td>
</tr>
<tr>
<td>queryop</td>
<td>Operation describing change to the query</td>
</tr>
<tr>
<td>table_desc_array</td>
<td>Array of table changes which contributed to the query Result Set change</td>
</tr>
</tbody>
</table>
36.6.3 CQ_NOTIFICATION$_QUERY_ARRAY Object (Array) Type

This type corresponds to an array of CQ_NOTIFICATION$_QUERY objects. It is a synonym for the SYS.CHNF$_QUERY_ARRAY type.

Syntax

TYPE CQ_NOTIFICATION$_QUERY_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_QUERY;

36.6.4 CQ_NOTIFICATION$_TABLE Object Type

This descriptor type describes a change to a table and is embedded inside the top level change notification descriptor type for EVENT_OBJCHANGE event types. For query result set changes (event type will be set to EVENT_QUERYCHANGE), the array of table descriptors is embedded inside each query change descriptor.

Note that this is a synonym for the type previously named SYS.CHNF$_TDESC.

Syntax

TYPE SYS.CHNF$_TDESC IS OBJECT (  
  opflags NUMBER,  
  table_name VARCHAR2(2*M_IDEN+1),  
  numrows NUMBER,  
  row_desc_array CQ_NOTIFICATION$_ROW_ARRAY)

Attributes

Table 36-4  TYPE SYS.CQ_NOTIFICATION$_TABLE Object Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opflags</td>
<td>Table level operation flags. This is a flag field (bit-vector) that describes the operations that occurred on the table. It can be an OR of the following bit fields - INSERTOP, UPDATEOP, DELETEOP, DROPPOP, ALTEROP, ALL_ROWS. If the ALL_ROWS (0x1) bit is set it means that either the entire table is modified (for example, DELETE * FROM t) or row level granularity of information is not requested or not available in the notification and the receiver has to conservatively assume that the entire table has been invalidated.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of modified table</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of modified rows within the table. numrows will be NULL and hence should not be accessed if the ALL_ROWS bit is set in the table change descriptor.</td>
</tr>
<tr>
<td>row_desc_array</td>
<td>Array of row descriptors. This field will be NULL if the ALL_ROWS bit is set in opflags.</td>
</tr>
</tbody>
</table>
36.6.5 CQ_NOTIFICATION$_TABLE_ARRAY Object (Array) Type

This type corresponds to an array of CQ_NOTIFICATION$_TABLE objects. It is a synonym for the SYS.CHNF$_TDESC_ARRAY type.

Syntax

```
TYPE CQ_NOTIFICATION$_TABLE_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_TABLE;
```

36.6.6 CQ_NOTIFICATION$_ROW Object Type

An array of CQ_NOTIFICATION$_ROW is embedded inside a CQ_NOTIFICATION$_TABLE (table change descriptor), if the QOS_ROWIDS option was chosen at registration and the ALL_ROWS bit is not set in the opflags field of the table change descriptor.

Note that this is a synonym for the type previously named SYS.CHNF$_RDESC.

Syntax

```
TYPE SYS.CHNF$_RDESC IS OBJECT (  
  opflags            NUMBER,  
  row_id             VARCHAR2 (2000));
```

Attributes

Table 36-5  TYPE SYS.CQ_NOTIFICATION$_ROW Object Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opflags</td>
<td>Row level operation flags. The flag field (bit vector) describes the operations in the row (could be INSERTOP, UPDATEOP or DELETEOP).</td>
</tr>
<tr>
<td>row_id</td>
<td>The rowid of the modified row</td>
</tr>
</tbody>
</table>

36.6.7 CQ_NOTIFICATION$_ROW_ARRAY Object (Array) Type

This object type corresponds to an array of CQ_NOTIFICATION$_ROW objects. It is embedded inside the CQ_NOTIFICATION$_TABLE if QOS_ROWIDS was specified during registration and the ALL_ROWS bit is not set in the opflags field of the table change descriptor.

This type is a synonym for the SYS.CHNF$_RDESC_ARRAY type.

Syntax

```
TYPE CQ_NOTIFICATION$_ROW_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_ROW;
```
36.6.8 CQ_NOTIFICATION$$_REG_INFO Object Type

The object type describes the attributes associated with creating a new registration. It is a synonym for the type previously named SYS.CHNF$$_REG_INFO.

Syntax

```sql
TYPE SYS.CHNF$$_REG_INFO IS OBJECT (
    callback                   VARCHAR2(20),
    qosflags                   NUMBER,
    timeout                    NUMBER,
    operations_filter          NUMBER,
    transaction_lag            NUMBER,
    ntfn_grouping_class        NUMBER,
    ntfn_grouping_value        NUMBER,
    ntfn_grouping_type         NUMBER,
    ntfn_grouping_start_time   TIMESTAMP WITH TIME ZONE,
    ntfn_grouping_repeat_count NUMBER);
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>Name of the server side PL/SQL procedure to be executed on a notification. Prototype is &lt;call_backname&gt;(ntfnds IN SYS.chnf$$_desc)</td>
</tr>
<tr>
<td>qosflags</td>
<td>Quality of service flags. Can be set to an OR of the following values:</td>
</tr>
<tr>
<td></td>
<td>• QOS_RELIABLE (0x1): Notifications are reliable (persistent) and survive instance death. This means that on an instance death in an Oracle RAC cluster, surviving instances will be able to deliver any queued invalidations. Similarly, pending invalidations can be delivered on instance restart, in a single instance configuration. The disadvantage is that there is a CPU cost/ latency involved in inserting the invalidation message to a persistent store. If this parameter is false, then server side CPU and latency are minimized, because invalidations are buffered into an in memory queue but the client could lose invalidation messages on an instance shutdown.</td>
</tr>
<tr>
<td></td>
<td>• QOS_DEREG_NFY (0x2): The registration will be expunged on the first notification</td>
</tr>
<tr>
<td></td>
<td>• QOS_ROWIDS (0x4): The notification needs to include information about the rowids that were modified</td>
</tr>
<tr>
<td></td>
<td>• QOS_QUERY (0x8): specifies query result change notification as opposed to object change notification</td>
</tr>
<tr>
<td></td>
<td>• QOS_BEST_EFFORT (0x10) or QOS_BEST_EFFORT (0x16): can register simplified versions of queries and minimizes evaluation with some false positives.</td>
</tr>
</tbody>
</table>
Table 36-6  (Cont.) TYPE CQ_NOTIFICATION$$_REG_INFO Object Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>If set to a nonzero value, specifies the time in seconds after which the registration is automatically expunged by the database. If zero / NULL, the registration lives until explicitly deregistered. Note that the timeout option can be combined with the purge on notification (QOS_DEREG_NFY) option as well.</td>
</tr>
<tr>
<td>operations_filter</td>
<td>if nonzero, specifies a filter to be selectively notified on certain operations. These flags can be used to filter based on specific operation types:</td>
</tr>
<tr>
<td></td>
<td>- 0: Notify on all operations (DBMS_CQ_NOTIFICATION.ALL_OPERATIONS)</td>
</tr>
<tr>
<td></td>
<td>- 0x2: Notify on every INSERT (DBMS_CQ_NOTIFICATION.INSERTOP)</td>
</tr>
<tr>
<td></td>
<td>- 0x4: Notify on every UPDATE (DBMS_CQ_NOTIFICATION.UPDATEOP)</td>
</tr>
<tr>
<td></td>
<td>- 0x8: Notify on every DELETE (DBMS_CQ_NOTIFICATION.DELETEOP)</td>
</tr>
<tr>
<td></td>
<td>A combination of operations can be specified by using a bitwise OR.</td>
</tr>
<tr>
<td>Caution:</td>
<td>This parameter will be honored for object level registrations but ignored for query result change registrations. To implement notification flow control in 11g, the applications can use the &quot;GROUPING notification&quot; option.</td>
</tr>
<tr>
<td>transaction_lag</td>
<td>Lag between consecutive notifications in units of transactions. Can be used to specify the number of transactions/database changes, by which the client is willing to lag behind the database. If 0, it means that the client needs to receive an invalidation message as soon as it is generated.</td>
</tr>
<tr>
<td>Caution:</td>
<td>This parameter will be honored for object level registrations but ignored for query result change notification registrations.</td>
</tr>
<tr>
<td>ntnf_grouping_class</td>
<td>When grouping notifications, the class based on which the group is derived. Currently, the only allowed value is DBMS_CQ_NOTIFICATION.NTFN_GROUPING_CLASS_TIME by which notifications are grouped by time.</td>
</tr>
<tr>
<td>ntnf_grouping_value</td>
<td>The grouping value. This describes the time interval that defines the group in seconds. For example, if this were set to 900, it would mean that notifications that were generated in each 15 minute interval would be grouped together.</td>
</tr>
<tr>
<td>ntnf_grouping_type</td>
<td>The type of grouping desired. It can be one of two allowed values</td>
</tr>
<tr>
<td></td>
<td>- DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_SUMMARY - all notifications in the group are summarized into a single notification</td>
</tr>
<tr>
<td></td>
<td>- DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_LAST - only the last notification in the group is published and the earlier ones discarded</td>
</tr>
</tbody>
</table>
Table 36-6  (Cont.) TYPE CQ_NOTIFICATION$_REG_INFO Object Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntfn_grouping_start_time</td>
<td>When to start generating notifications. If specified as NULL, it defaults to the current system generated time.</td>
</tr>
<tr>
<td>ntfn_grouping_repeat_count</td>
<td>How many times the notification should be repeated. Set this to DBMS_CQ_NOTIFICATION.NTFN_GROUPING_FOREVER to receive notifications for the life time of the registration. Set to a nonzero value if only a certain number of notifications are desired for the life time of the registration.</td>
</tr>
</tbody>
</table>

Usage Notes

- The type declaration incorporates three other alternative constructors. In the first case all other parameters default to their default values.

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
    callback                        VARCHAR2(20),
    qosflags                        NUMBER,
    timeout                         NUMBER);
```

The second option applies to the type constructor defined in a previous release, and which is retained for backward compatibility:

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
    callback                        VARCHAR2(20),
    qosflags                        NUMBER,
    timeout                         NUMBER,
    operations_filter               NUMBER,
    transaction_lag                 NUMBER);
```

The third definition contains all the members of the type except `transaction_lag` which is being deprecated:

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
    callback                        VARCHAR2(20),
    qosflags                        NUMBER,
    timeout                         NUMBER,
    operations_filter               NUMBER,
    ntfn_grouping_class             NUMBER,
    ntfn_grouping_value             NUMBER,
    ntfn_grouping_type              NUMBER,
    ntfn_grouping_start_time        TIMESTAMP WITH TIME ZONE,
    ntfn_grouping_repeat_count      NUMBER);
```

- In response to a database change, the server side PL/SQL procedure specified by "callback" is executed. The PL/SQL procedure name has to be specified in the format `schema_name.procedure_name`. The procedure must have the following signature:

```
PROCEDURE <procedure_name>(ntfnds IN SYS.chnf$_desc)
```

`CHNF$_DESC` describes the change notification descriptor.

- The `init.ora` parameter `job_queue_processes` must be set to a nonzero value to receive PL/SQL notifications, because the specified procedure is executed inside a job queue process when a notification is generated.
36.7 Summary of DBMS_CQ_NOTIFICATION Subprograms

This table lists the DBMS_CQ_NOTIFICATION subprograms and briefly describes them.

Table 36-7 DBMS_CQ_NOTIFICATION Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ_NOTIFICATION_QUERYID</td>
<td>Returns the queryid of the most recent query that was attempted to be registered in a registration block</td>
</tr>
<tr>
<td>DEREGISTER Procedure</td>
<td>De-subscribes the client with the supplied registration identifier (ID)</td>
</tr>
<tr>
<td>ENABLE_REG Procedure</td>
<td>Begins a registration block using an existing registration identifier (ID)</td>
</tr>
<tr>
<td>NEW_REG_START Function</td>
<td>Begins a new registration block</td>
</tr>
<tr>
<td>REG_END Procedure</td>
<td>Ends the registration boundary</td>
</tr>
<tr>
<td>SET_ROWID_THRESHOLD Procedure</td>
<td>Configures the maximum number of rows of a table published in a change notification if the rows of the table are modified in a transaction</td>
</tr>
</tbody>
</table>

36.7.1 CQ_NOTIFICATION_QUERYID Function

This function returns the queryid of the most recent query that was attempted to be registered in a registration block.

Syntax

```
DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID
RETURN NUMBER;
```

Return Values

Returns the queryid of the most recently registered query.

36.7.2 DEREGISTER Procedure

This procedure describes the client with the specified registration identifier (ID).

Syntax

```
DBMS_CQ_NOTIFICATION.DEREGISTER (regid IN NUMBER);
```

Parameters

Table 36-8 DEREGISTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regid</td>
<td>Client registration ID</td>
</tr>
</tbody>
</table>
Usage Notes
Only the user that created the registration (or the SYS user) will be able to desubscribe the registration.

36.7.3 ENABLE_REG Procedure

This procedure adds objects to an existing registration identifier (ID).

It is similar to the interface for creating a new registration, except that it takes an existing regid to which to add objects. Subsequent execution of queries causes the objects referenced in the queries to be added to the specified regid, and the registration is completed on invoking the REG_END Procedure.

Syntax

```
DBMS_CQ_NOTIFICATION.ENABLE_REG (
    regid IN NUMBER);
```

Parameters

Table 36-9  ENABLE_REG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regid</td>
<td>Client registration ID</td>
</tr>
</tbody>
</table>

Usage Notes
Only the user that created the registration will be able to add further objects to the registration.

36.7.4 NEW_REG_START Function

This procedure begins a new registration block.

Any objects referenced by queries executed within the registration block are considered interesting objects and added to the registration. The registration block ends upon calling the REG_END procedure.

Syntax

```
DBMS_CQ_NOTIFICATION.NEW_REG_START (
    regds IN sys.chnf$_reg_info)
RETURN NUMBER;
```

Parameters

Table 36-10  NEW_REG_START Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys.chnf$_reg_info</td>
<td>Registration descriptor describing the notification handler and other properties of the registration</td>
</tr>
</tbody>
</table>
Return Values

The procedure returns a registration-id which is a unique integer assigned by the database to this registration. The registration-id will be echoed back in every notification received for this registration.

Usage Notes

- The only operations permitted inside a registration block are queries (the ones the user wishes to register). DML and DDL operations are not permitted.
- The registration block is a session property and implicitly terminates upon exiting the session. While the registration block is a session property, the registration itself is a persistent database entity. Once created, the registration survives until explicitly deregistered by the client application or timed-out or removed by the database for some other reason (such as loss of privileges).
- The user must have the CHANGE NOTIFICATION system privilege and SELECT or READ privileges on any objects to be registered.
- The SYS user will not be permitted to create new registrations.
- Nesting of registration block is not permitted.

36.7.5 REG_END Procedure

This procedure marks the end of the registration block. No newly executed queries are tracked.

Syntax

```
DBMS_CQ_NOTIFICATION.REG_END;
```

36.7.6 SET_ROWID_THRESHOLD Procedure

This procedure configures the maximum number of rows of a table published in a change notification if the rows of the table are modified in a transaction.

Syntax

```
DBMS_CQ_NOTIFICATION.SET_ROWID_THRESHOLD (    
    tname       IN  VARCHAR2,    
    threshold   IN  NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tname</td>
<td>Table name qualified by the schema name in the form schemaname.tablename</td>
</tr>
<tr>
<td>threshold</td>
<td>Maximum number of modified rows of the table to be published in the change notification</td>
</tr>
</tbody>
</table>
Usage Notes

- The table needs to be registered for change notification either at object change granularity or at query result set granularity.
- The threshold set by means of this subprogram applies to that instance only and does not persist across instance startup/shutdown.
The DBMS_CREDENTIAL package provides an interface for authenticating and impersonating EXTPROC callout functions, as well as external jobs, remote jobs and file watchers from the SCHEDULER.

See Also:

- Oracle Database Administrator's Guide regarding Specifying Job Credentials

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Summary of DBMS_CREDENTIAL Subprograms

37.1 DBMS_CREDENTIAL Overview

Credentials are database objects that hold a username/password pair for authenticating and impersonating EXTPROC callout functions, as well as remote jobs, external jobs and file watchers from the SCHEDULER.

They are created using the CREATE_CREDENTIAL Procedure. The procedure also allows you to specify the Windows domain for remote external jobs executed against a Windows server.

37.2 DBMS_CREDENTIAL Security Model

Every Oracle credential has a unique credential name and you can associate a credential through its unique credential name with EXTPROC by means of a PL/SQL alias library.

In order to associate a credential with a PL/SQL alias library and external procedure, you must have the CREATE AND/OR REPLACE LIBRARY privilege or CREATE AND/OR REPLACE FUNCTION / PROCEDURE privilege and read permission of the DLL or shared object that the alias library to be associated with so that you can create and/or replace function or procedure to make use of the alias library.

Once authenticated, EXTPROC must act on behalf of the client based on client's identity defined in the supplied user credential. If not authenticated, EXTPROC must return an error message.
In order to create or alter a credential, you must have the CREATE CREDENTIAL privilege. If you are attempting to create or alter a credential in a schema other than your own, you must have the CREATE ANY CREDENTIAL privilege.

### 37.3 DBMS_CREDENTIAL Operational Notes

As the existing CREATE OR REPLACE LIBRARY statement and CREATE OR REPLACE FUNCTION/PROCEDURE do not support a CREDENTIAL clause, this model requires syntax and semantic changes in CREATE OR REPLACE LIBRARY and CREATE OR REPLACE FUNCTION/PROCEDURE statement.

For example:

```sql
CREATE OR REPLACE LIBRARY test
    AS '$ORACLE_HOME/bin/test.so' CREDENTIAL ricky_cred;
CREATE OR REPLACE FUNCTION ftest1
    (x VARCHAR2, y BINARY_INTEGER)
RETURN BINARY_INTEGER
AS LANGUAGE C
LIBRARY test
NAME "negative"
PARAMETERS(x STRING, y INT);
```

The credential name defined in the CREDENTIAL clause is a name of a database object. Therefore, do not enclose the credential name with single or double quotes.

An example of a credential being used on an external job:

```sql
BEGIN
    DBMS_SCHEDULER.CREATE_JOB(
        job_name => 'example_job',
        job_type => 'EXECUTABLE',
        job_action => '/bin/ls',
        credential_name => 'ricky_cred');
END;
```

### 37.4 Summary of DBMS_CREDENTIAL Subprograms

This table lists the DBMS_CREDENTIAL subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_CREDENTIAL</td>
<td>Creates a stored username/password pair in a database object called an Oracle credential</td>
</tr>
<tr>
<td>DISABLE_CREDENTIAL</td>
<td>Disables an Oracle credential</td>
</tr>
<tr>
<td>DROP_CREDENTIAL</td>
<td>Drops an Oracle credential</td>
</tr>
<tr>
<td>ENABLE_CREDENTIAL</td>
<td>Enables an Oracle credential</td>
</tr>
<tr>
<td>UPDATE_CREDENTIAL</td>
<td>Updates an existing Oracle credential</td>
</tr>
</tbody>
</table>
37.4.1 CREATE_CREDENTIAL Procedure

This procedure creates a stored username/password pair in a database object called an Oracle credential.

Syntax

```sql
DBMS_CREDENTIAL.CREATE_CREDENTIAL (  
    credential_name   IN  VARCHAR2,  
    username          IN  VARCHAR2,  
    password          IN  VARCHAR2,  
    database_role     IN  VARCHAR2 DEFAULT NULL,  
    windows_domain    IN  VARCHAR2 DEFAULT NULL,  
    comments          IN  VARCHAR2 DEFAULT NULL,  
    enabled           IN  BOOLEAN DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential_name</td>
<td>Name of the credential. It can optionally be prefixed with a schema. This cannot be set to NULL. It is converted to upper case unless enclosed in double quotes.</td>
</tr>
<tr>
<td>username</td>
<td>User name to login to the operating system or remote database to run a job if this credential is chosen. This cannot be set to NULL.</td>
</tr>
<tr>
<td>password</td>
<td>Password to login to the remote operating system to run a job if this credential is chosen. It is case sensitive.</td>
</tr>
<tr>
<td>database_role</td>
<td>Whether a database job using this credential should attempt to log in with administrative privileges. Values: SYSDBA, SYSDG, SYSADMIN or SYSBACKUP.</td>
</tr>
<tr>
<td>windows_domain</td>
<td>For a Windows remote executable target, this is the domain that the specified user belongs to. The domain will be converted to uppercase automatically.</td>
</tr>
<tr>
<td>comments</td>
<td>A text string that can be used to describe the credential to the user. The Scheduler does not use this field.</td>
</tr>
<tr>
<td>enabled</td>
<td>Determines whether the credential is enabled or not</td>
</tr>
</tbody>
</table>

Usage Notes

- Credentials reside in a particular schema and can be created by any user with the CREATE_CREDENTIAL or CREATE ANY CREDENTIAL system privilege. To create a credential in a schema other than your own, you must have the CREATE CREDENTIAL or CREATE ANY CREDENTIAL privilege.
- The user name is case sensitive. It cannot contain double quotes or spaces.
- Attempting to create a credential with an existing credential name returns an error. To alter an existing credential, users must drop the existing credential first using the DROP_CREDENTIAL Procedure.
- Attempting to drop an existing credential, which is already referenced by alias libraries, returns an error. To drop an existing credential without any checking, users must set the force parameter of DROP_CREDENTIAL Procedure to TRUE.
• You may also alter a credential, by means of the `UPDATE_CREDENTIAL` Procedure.

Examples

Create a Basic Credential

CONN scott
Enter password: password

BEGIN
  -- Basic credential.
  DBMS_CREDENTIAL.CREATE_CREDENTIAL(
    credential_name  =>  'JAMES_SMITH_CREDENTIAL',
    username         =>  'james_smith',
    password         =>  'password');
END

Create a Windows Credential

CONN scott
Enter password: password

  -- Credential including Windows domain
BEGIN
  DBMS_CREDENTIAL.CREATE_CREDENTIAL(
    credential_name   =>  'JAMES_SMITH_WIN_CREDENTIAL',
    username          =>  'james_smith',
    password          =>  'password',
    windows_domain    =>  'localdomain');
END

Display Information about Credentials

Information about credentials is displayed using the `[DBA|ALL|USER]_CREDENTIALS` views.

COLUMN credential_name FORMAT A25
COLUMN username FORMAT A20
COLUMN windows_domain FORMAT A20
SELECT credential_name, username, windows_domain
FROM   user_credentials
ORDER BY credential_name;

<table>
<thead>
<tr>
<th>CREDENTIAL_NAME</th>
<th>USERNAME</th>
<th>WINDOWS_DOMAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAMES_SMITH_CREDENTIAL</td>
<td>james_smith</td>
<td></td>
</tr>
<tr>
<td>JAMES_SMITH_WIN_CREDENTIAL</td>
<td>james_smith</td>
<td>LOCALDOMAIN</td>
</tr>
</tbody>
</table>

2 rows selected.

SQL>
37.4.2 DISABLE_CREDENTIAL Procedure

This procedure disables an Oracle credential.

Syntax

DBMS_CREDENTIAL.DISABLE_CREDENTIAL (  
    credential_name   IN  VARCHAR2,  
    force             IN  BOOLEAN DEFAULT FALSE);  

Parameters

Table 37-3  DISABLE_CREDENTIAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential_name</td>
<td>Name of the credential. It can optionally be prefixed with a schema. This cannot be set to NULL. It is converted to upper case unless enclosed in double quotes.</td>
</tr>
<tr>
<td>force</td>
<td>If FALSE, the credential is not disabled provided it has no dependency on any existing scheduler job or PL/SQL library. An error is returned if the dependency is observed. If TRUE, the credential is disabled whether or not there is any scheduler job or PL/SQL library referencing it.</td>
</tr>
</tbody>
</table>

Usage Notes

- Credentials reside in a particular schema and can be disabled by any user with the CREATE CREDENTIAL or CREATE ANY CREDENTIAL system privilege. To disable a credential in a schema other than your own, you must have the CREATE ANY CREDENTIAL privilege.

- A credential for an OS user can be viewed as an entry point into an operating system as a particular user. Allowing a credential to be disabled lets an administrator (or credential owner) to quickly, easily and reversibly disallow all logins from the database to the OS as a particular user of external jobs, database jobs, file transfers, external procedures, and file watching. To enable an existing disabled credential, you need to use the ENABLE_CREDENTIAL Procedure.

- A library can become invalid if the properties of the credential – windows domain, username, password, its enable/disable bit – are changed.

37.4.3 DROP_CREDENTIAL Procedure

This procedure drops an Oracle credential.

Syntax

DBMS_CREDENTIAL.DROP_CREDENTIAL (  
    credential_name   IN  VARCHAR2,  
    force             IN  BOOLEAN DEFAULT FALSE);
Parameters

**Table 37-4 DROP_CREDENTIAL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential_name</td>
<td>Name of the credential. It can optionally be prefixed with a schema. This cannot be set to NULL.</td>
</tr>
<tr>
<td>force</td>
<td>If set to FALSE, the credential must not be referenced by any EXTPROC alias library or an error is raised. If set to TRUE, the credential is dropped whether or not there are extproc alias libraries referencing it. EXTPROC alias libraries that reference the dropped credential become invalid.</td>
</tr>
</tbody>
</table>

**Usage Notes**

Only the owner of a credential or a user with the CREATE ANY CREDENTIAL system privilege may drop the credential.

**Examples**

EXEC DBMS_CREDENTIAL.DROP_CREDENTIAL('JAMES_SMITH_CREDENTIAL', FALSE);
EXEC DBMS_CREDENTIAL.DROP_CREDENTIAL('JAMES_SMITH_WIN_CREDENTIAL', FALSE);

37.4.4 ENABLE_CREDENTIAL Procedure

This procedure enables an Oracle credential.

**Syntax**

```sql
DBMS_CREDENTIAL.ENABLE_CREDENTIAL ( credential_name IN VARCHAR2 );
```

**Parameters**

**Table 37-5 ENABLE_CREDENTIAL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential_name</td>
<td>Name of the credential. It can optionally be prefixed with a schema. This cannot be set to NULL. It is converted to upper case unless enclosed in double quotes.</td>
</tr>
</tbody>
</table>

**Usage Notes**

- Credentials reside in a particular schema and can be disabled by any user with the CREATE CREDENTIAL OR CREATE ANY CREDENTIAL system privilege. To disable a credential in a schema other than your own, you must have the CREATE CREDENTIAL OR CREATE ANY CREDENTIAL privilege.

- A credential for an OS user can be viewed as an entry point into an operating system as a particular user. Allowing a credential to be disabled would allow an administrator (or credential owner) to quickly, easily and reversibly disallow all logins from the database to the OS as a particular user (external jobs, file
transfers, external procedures, file watching). To disable an existing credential, you need to use the DISABLE_CREDENTIAL Procedure.

- A library can become invalid if the properties of the credential – windows domain, username, password, its enable/disable bit – are changed.

### 37.4.5 UPDATE_CREDENTIAL Procedure

This procedure updates an existing Oracle credential.

**Syntax**

```sql
DBMS_CREDENTIAL.UPDATE_CREDENTIAL (  
    credential_name   IN   VARCHAR2,  
    attribute         IN   VARCHAR2,  
    value             IN   VARCHAR2);
```

**Parameters**

**Table 37-6 UPDATE_CREDENTIAL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential_name</td>
<td>Name of the credential. It can optionally be prefixed with a schema. This cannot be set to NULL. It is converted to upper case unless enclosed in double quotation marks.</td>
</tr>
<tr>
<td>attribute</td>
<td>Name of attribute to update: USERNAME, PASSWORD, WINDOWS_DOMAIN, DATABASE_ROLE or COMMENTS</td>
</tr>
<tr>
<td>value</td>
<td>New value for the selected attribute</td>
</tr>
</tbody>
</table>

**Usage Notes**

- Credentials reside in a particular schema and can be created by any user with the CREATE CREDENTIAL or CREATE ANY CREDENTIAL system privilege. To create a credential in a schema other than your own, you must have the CREATE ANY CREDENTIAL privilege.
- The user name is case sensitive. It cannot contain double quotes or spaces.
- EXTPROC alias libraries that reference the updated credential will become invalid. A library becomes invalid if the properties of the credential – windows domain, username, password, its enable/disable bit – are changed.

**Examples**

**Update a Basic Credential**

```
CONN scott
Enter password: password

BEGIN
  -- Basic credential.
  DBMS_CREDENTIAL.UPDATE_CREDENTIAL (  
    credential_name => 'JAMES_SMITH_CREDENTIAL',  
    attribute        => 'password',  
    value            => 'password2');

  DBMS_CREDENTIAL.UPDATE_CREDENTIAL (  
    credential_name => 'JAMES_SMITH_CREDENTIAL',  
    attribute        => 'password',  
    value            => 'password2');

END;
```
attribute => 'username',
value => 'james_smith');
END;

Update a Windows Credential

CONN scott
Enter password: password

-- Credential including Windows domain
BEGIN
DBMS_CREDENTIAL.UPDATE_CREDENTIAL(
    credential_name  =>  'JAMES_SMITH_WIN_CREDENTIAL',
    username          =>  'james_smith',
    password          =>  'password',
    windows_domain    =>  'localdomain');
END

Display Information about Credentials

Information about credentials is displayed using the \[DBA|ALL|USER\] _CREDENTIALS views.

COLUMN credential_name FORMAT A25
COLUMN username FORMAT A20
COLUMN windows_domain FORMAT A20
SELECT credential_name,
    username,
    windows_domain
FROM   all_credentials
ORDER BY credential_name;

<table>
<thead>
<tr>
<th>CREDENTIAL_NAME</th>
<th>USERNAME</th>
<th>WINDOWS_DOMAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAMES_SMITH_CREDENTIAL</td>
<td>james_smith</td>
<td></td>
</tr>
<tr>
<td>JAMES_SMITH_WIN_CREDENTIAL</td>
<td>james_smith</td>
<td>localdomain</td>
</tr>
</tbody>
</table>

2 rows selected.

SQL>
DBMS_CRYPTO provides an interface to encrypt and decrypt stored data, and can be used in conjunction with PL/SQL programs running network communications. It provides support for several industry-standard encryption and hashing algorithms, including the Advanced Encryption Standard (AES) encryption algorithm. AES has been approved by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES).

See Also:
Oracle Database Security Guide for further information about using this package and about encrypting data in general.

This chapter contains the following topics:

- Overview
- Security Model
- Datatypes
- Algorithms
- Restrictions
- Exceptions
- Operational Notes
- Summary of DBMS_CRYPTO Subprograms

38.1 DBMS_CRYPTO Overview

DBMS_CRYPTO contains basic cryptographic functions and procedures. To use this package correctly and securely, a general level of security expertise is assumed.

The DBMS_CRYPTO package enables encryption and decryption for common Oracle datatypes, including RAW and large objects (LOBs), such as images and sound. Specifically, it supports BLOBs and CLOBs. In addition, it provides Globalization Support for encrypting data across different database character sets.

The following cryptographic algorithms are supported:

- Data Encryption Standard (DES), Triple DES (3DES, 2-key and 3-key)
- Advanced Encryption Standard (AES)
- MD5, MD4, SHA-1, and SHA-2 cryptographic hashes
- MD5, SHA-1, and SHA-2 Message Authentication Code (MAC)
Block cipher modifiers are also provided with DBMS_CRYPTO. You can choose from several padding options, including PKCS (Public Key Cryptographic Standard) #5, and from four block cipher chaining modes, including Cipher Block Chaining (CBC).

Table 38-1 summarizes the DBMS_CRYPTO package features.

Table 38-1  DBMS_CRYPTO Features

<table>
<thead>
<tr>
<th>Package Feature</th>
<th>DBMS_CRYPTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic algorithms</td>
<td>DES, 3DES, AES, RC4, 3DES_2KEY</td>
</tr>
<tr>
<td>Padding forms</td>
<td>PKCSS, zeroes</td>
</tr>
<tr>
<td>Block cipher chaining modes</td>
<td>CBC, CFB, ECB, OFB</td>
</tr>
<tr>
<td>Cryptographic hash algorithms</td>
<td>MD5, SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512), MD4</td>
</tr>
<tr>
<td>Keyed hash (MAC) algorithms</td>
<td>HMAC_MD5, HMAC_SH1, HMAC_SH256, HMAC_SH384, HMAC_SH512</td>
</tr>
<tr>
<td>Cryptographic pseudo-random number generator</td>
<td>RAW, NUMBER, BINARY_INTEGER</td>
</tr>
<tr>
<td>Database types</td>
<td>RAW, CLOB, BLOB</td>
</tr>
</tbody>
</table>

The DBMS_CRYPTO package replaces DBMS_OBFUSCATION_TOOLKIT, providing greater ease of use and support for a range of algorithms to accommodate new and existing systems. Specifically, 3DES_2KEY and MD4 are provided for backward compatibility. It is not recommended that you use these algorithms because they do not provide the same level of security as provided by 3DES, AES, MD5, SHA-1, or SHA-2.

38.2 DBMS_CRYPTO Security Model

Oracle Database installs this package in the SYS schema. You can then grant package access to existing users and roles as needed.

38.3 DBMS_CRYPTO Datatypes

Parameters for the DBMS_CRYPTO subprograms use these datatypes.

Table 38-2  DBMS_CRYPTO Datatypes

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>A source or destination binary LOB</td>
</tr>
<tr>
<td>CLOB</td>
<td>A source or destination character LOB (excluding NCLOB)</td>
</tr>
<tr>
<td>PLS_INTEGER</td>
<td>Specifies a cryptographic algorithm type (used with BLOB, CLOB, and RAW datatypes)</td>
</tr>
<tr>
<td>RAW</td>
<td>A source or destination RAW buffer</td>
</tr>
</tbody>
</table>
38.4 DBMS_CRYPTO Algorithms

The DBMS_CRYPTO package contains predefined cryptographic algorithms, modifiers, and cipher suites.

These are shown in the following tables.

Table 38-3  DBMS_CRYPTO Cryptographic Hash Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH_MD4</td>
<td>Produces a 128-bit hash, or message digest of the input message</td>
</tr>
<tr>
<td>HASH_MD5</td>
<td>Also produces a 128-bit hash, but is more complex than MD4</td>
</tr>
<tr>
<td>HASH_SH1</td>
<td>Secure Hash Algorithm (SHA-1). Produces a 160-bit hash.</td>
</tr>
<tr>
<td>HASH_SH256</td>
<td>SHA-2, produces a 256-bit hash.</td>
</tr>
<tr>
<td>HASH_SH384</td>
<td>SHA-2, produces a 384-bit hash.</td>
</tr>
<tr>
<td>HASH_SH512</td>
<td>SHA-2, produces a 512-bit hash.</td>
</tr>
</tbody>
</table>

Table 38-4  DBMS_CRYPTO MAC (Message Authentication Code) Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMAC_MD5⁹</td>
<td>Same as MD5 hash function, except it requires a secret key to verify the hash value.</td>
</tr>
<tr>
<td>HMAC_SH1⁹</td>
<td>Same as SHA hash function, except it requires a secret key to verify the hash value.</td>
</tr>
<tr>
<td>HMAC_SH256</td>
<td>Same as SHA-2 256-bit hash function, except it requires a secret key to verify the hash value.</td>
</tr>
<tr>
<td>HMAC_SH384</td>
<td>Same as SHA-2 384-bit hash function, except it requires a secret key to verify the hash value.</td>
</tr>
<tr>
<td>HMAC_SH512</td>
<td>Same as SHA-2 512-bit hash function, except it requires a secret key to verify the hash value.</td>
</tr>
</tbody>
</table>

¹ Complies with IETF RFC 2104 standard

Table 38-5  DBMS_CRYPTO Encryption Algorithms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPT_DES</td>
<td>Data Encryption Standard. Block cipher. Uses key length of 56 bits.</td>
</tr>
<tr>
<td>ENCRYPT_3DES</td>
<td>Data Encryption Standard. Block cipher. Operates on a block 3 times with 2 keys. Effective key length of 112 bits.</td>
</tr>
<tr>
<td>ENCRYPT_3DES_2KEY</td>
<td>Data Encryption Standard. Block cipher. Operates on a block 3 times.</td>
</tr>
<tr>
<td>ENCRYPT_RC4</td>
<td>Stream cipher. Uses a secret, randomly generated key unique to each session.</td>
</tr>
</tbody>
</table>
### Table 38-6  DBMS_CRYPTO Block Cipher Suites

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES_CBC_PKCS5</td>
<td>ENCRYPT_DES&lt;sup&gt;1&lt;/sup&gt; + CHAIN_CBC&lt;sup&gt;2&lt;/sup&gt; + PAD_PKCS5&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td>DES3_CBC_PKCS5</td>
<td>ENCRYPT_3DES&lt;sup&gt;1&lt;/sup&gt; + CHAIN_CBC&lt;sup&gt;2&lt;/sup&gt; + PAD_PKCS5&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

<sup>1</sup> See Table 38-5  
<sup>2</sup> See Table 38-7  
<sup>3</sup> See Table 38-8

### Table 38-7  DBMS_CRYPTO Block Cipher Chaining Modifiers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAIN_ECB</td>
<td>Electronic Codebook. Encrypts each plaintext block independently.</td>
</tr>
<tr>
<td>CHAIN_CBC</td>
<td>Cipher Block Chaining. Plaintext is XORed with the previous ciphertext block before it is encrypted.</td>
</tr>
<tr>
<td>CHAIN_CFB</td>
<td>Cipher-Feedback. Enables encrypting units of data smaller than the block size.</td>
</tr>
<tr>
<td>CHAIN_OFB</td>
<td>Output-Feedback. Enables running a block cipher as a synchronous stream cipher. Similar to CFB, except that n bits of the previous output block are moved into the right-most positions of the data queue waiting to be encrypted.</td>
</tr>
</tbody>
</table>

### Table 38-8  DBMS_CRYPTO Block Cipher Padding Modifiers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAD_PKCS5</td>
<td>Provides padding which complies with the PKCS #5: Password-Based Cryptography Standard</td>
</tr>
<tr>
<td>PAD_NONE</td>
<td>Provides option to specify no padding. Caller must ensure that blocksize is correct, else the package returns an error.</td>
</tr>
<tr>
<td>PAD_ZERO</td>
<td>Provides padding consisting of zeroes</td>
</tr>
</tbody>
</table>

### 38.5 DBMS_CRYPTO Restrictions

The `VARCHAR2` datatype is not directly supported by `DBMS_CRYPTO`. Before you can perform cryptographic operations on data of the type `VARCHAR2`, you must convert it to the uniform database character set `AL32UTF8`, and then convert it to the `RAW` datatype. After performing these conversions, you can then encrypt it with the `DBMS_CRYPTO` package.

**See Also:**

DBMS_CRYPTO Operational Notes for information about the conversion rules for converting datatypes.
### 38.6 DBMS_CRYPTO Exceptions

The following table lists exceptions that have been defined for DBMS_CRYPTO.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CipherSuiteInvalid</td>
<td>28827</td>
<td>The specified cipher suite is not defined.</td>
</tr>
<tr>
<td>CipherSuiteNull</td>
<td>28829</td>
<td>No value has been specified for the cipher suite to be used.</td>
</tr>
<tr>
<td>KeyNull</td>
<td>28239</td>
<td>The encryption key has not been specified or contains a NULL value.</td>
</tr>
<tr>
<td>KeyBadSize</td>
<td>28234</td>
<td>DES keys: Specified key size is too short. DES keys must be at least 8 bytes (64 bits). AES keys: Specified key size is not supported. AES keys must be 128, 192, or 256 bits in length.</td>
</tr>
<tr>
<td>DoubleEncryption</td>
<td>28233</td>
<td>Source data was previously encrypted.</td>
</tr>
</tbody>
</table>

### 38.7 DBMS_CRYPTO Operational Notes

This section describes several DBMS_CRYPTO operational notes.

- When to Use Encrypt and Decrypt Procedures or Functions
- When to Use Hash or Message Authentication Code (MAC) Functions
- About Generating and Storing Encryption Keys
- Conversion Rules

#### When to Use Encrypt and Decrypt Procedures or Functions

This package includes both `ENCRYPT` and `DECRYPT` procedures and functions. The procedures are used to encrypt or decrypt LOB datatypes (overloaded for CLOB and BLOB datatypes). In contrast, the `ENCRYPT` and `DECRYPT` functions are used to encrypt and decrypt RAW datatypes. Data of type `VARCHAR2` must be converted to RAW before you can use DBMS_CRYPTO functions to encrypt it.

#### When to Use Hash or Message Authentication Code (MAC) Functions

This package includes two different types of one-way hash functions: the `HASH` function and the `MAC` function. Hash functions operate on an arbitrary-length input message, and return a fixed-length hash value. One-way hash functions work in one direction only. It is easy to compute a hash value from an input message, but it is extremely difficult to generate an input message that hashes to a particular value. Note that hash values should be at least 128 bits in length to be considered secure.

You can use hash values to verify whether data has been altered. For example, before storing data, the user runs `DBMS_CRYPTO_HASH` against the stored data to create a hash value. On returning the stored data, the user can again run the hash function against it, using the same algorithm. If the second hash value is identical to the first one, then the data has not been altered. Hash values are similar to “file fingerprints” and are used to ensure data integrity.
The **HASH** function included with **DBMS_CRYPTO**, is a one-way hash function that you can use to generate a hash value from either **RAW** or **LOB** data. The **MAC** function is also a one-way hash function, but with the addition of a secret key. It works the same way as the **DBMS_CRYPTO.HASH** function, except only someone with the key can verify the hash value.

MACs can be used to authenticate files between users. They can also be used by a single user to determine if her files have been altered, perhaps by a virus. A user could compute the MAC of his files and store that value in a table. If the user did not use a MAC function, then the virus could compute the new hash value after infection and replace the table entry. A virus cannot do that with a MAC because the virus does not know the key.

**About Generating and Storing Encryption Keys**

The **DBMS_CRYPTO** package can generate random material for encryption keys, but it does not provide a mechanism for maintaining them. Application developers must take care to ensure that the encryption keys used with this package are securely generated and stored. Also note that the encryption and decryption operations performed by **DBMS_CRYPTO** occur on the server, not on the client. Consequently, if the key is sent over the connection between the client and the server, the connection must be protected by using network encryption. Otherwise, the key is vulnerable to capture over the wire.

Although **DBMS_CRYPTO** cannot generate keys on its own, it does provide tools you can use to aid in key generation. For example, you can use the **RANDOMBYTES** function to generate random material for keys.

When generating encryption keys for DES, it is important to remember that some numbers are considered weak and semiweak keys. Keys are considered weak or semiweak when the pattern of the algorithm combines with the pattern of the initial key value to produce ciphertext that is more susceptible to cryptanalysis. To avoid this, filter out the known weak DES keys. Lists of the known weak and semiweak DES keys are available on several public Internet sites.

---

**See Also:**

- *Oracle Database Security Guide* for information about configuring network encryption and SSL
- **RANDOMBYTES** Function

---

**Conversion Rules**

- To convert **VARCHAR2** to **RAW**, use the **UTL_I18N.STRING_TO_RAW** function to perform the following steps:
  
  1. Convert **VARCHAR2** in the current database character set to **VARCHAR2** in the **AL32UTF8** database character.
  2. Convert **VARCHAR2** in the **AL32UTF8** database character set to **RAW**.

  Syntax example:

  ```sql
  UTL_I18N.STRING_TO_RAW (string, 'AL32UTF8');
  ```
To convert RAW to VARCHAR2, use the UTL_I18N.RAW_TO_CHAR function to perform the following steps:

1. Convert RAW to VARCHAR2 in the AL32UTF8 database character set.
2. Convert VARCHAR2 in the AL32UTF8 database character set to VARCHAR2 in the database character set you wish to use.

Syntax example:

```
UTL_I18N.RAW_TO_CHAR (data, 'AL32UTF8');
```

See Also:

UTL_I18N for information about using the UTL_I18N PL/SQL package.

If you want to store encrypted data of the RAW datatype in a VARCHAR2 database column, then use RAWTOHEX or UTL_ENCODE.BASE64_ENCODE to make it suitable for VARCHAR2 storage. These functions expand data size by 2 and 4/3, respectively.

### 38.8 DBMS_CRYPTO Examples

This listing shows PL/SQL block encrypting and decrypting pre-defined 'input_string' using 256-bit AES algorithm with Cipher Block Chaining and PKCS#5 compliant padding.

```
DECLARE
    input_string       VARCHAR2 (200) := 'Secret Message';
    output_string      VARCHAR2 (200);
    encrypted_raw      RAW (2000);             -- stores encrypted binary text
    decrypted_raw      RAW (2000);             -- stores decrypted binary text
    num_key_bytes      NUMBER := 256/8;        -- key length 256 bits (32 bytes)
    key_bytes_raw      RAW (32);               -- stores 256-bit encryption key
    encryption_type    PLS_INTEGER := 0;
    iv_raw             RAW (16);
BEGIN
    DBMS_OUTPUT.PUT_LINE ( 'Original string: ' || input_string);
    key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
    iv_raw        := DBMS_CRYPTO.RANDOMBYTES (16);
    encrypted_raw := DBMS_CRYPTO.ENCRYPT
    (src => UTL_I18N.STRING_TO_RAW (input_string, 'AL32UTF8'),
     typ => encryption_type,
     key => key_bytes_raw,
     iv  => iv_raw
    );

    -- The encrypted value "encrypted_raw" can be used here

    decrypted_raw := DBMS_CRYPTO.DECRYPT
    (src => encrypted_raw,
     typ => encryption_type,
    );
```

Chapter 38
This table lists the DBMS_CRYPTO subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECRYPT Function</td>
<td>Decrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector)</td>
</tr>
<tr>
<td>DECRYPT Procedures</td>
<td>Decrypts LOB data using a stream or block cipher with a user supplied key and optional IV</td>
</tr>
<tr>
<td>ENCRYPT Function</td>
<td>Encrypts RAW data using a stream or block cipher with a user supplied key and optional IV</td>
</tr>
<tr>
<td>ENCRYPT Procedures</td>
<td>Encrypts LOB data using a stream or block cipher with a user supplied key and optional IV</td>
</tr>
<tr>
<td>HASH Function</td>
<td>Applies one of the supported cryptographic hash algorithms (MD4, MD5, SHA-1, or SHA-2) to data</td>
</tr>
<tr>
<td>MAC Function</td>
<td>Applies Message Authentication Code algorithms (MD5, SHA-1, or SHA-2) to data to provide keyed message protection</td>
</tr>
<tr>
<td>RANDOMBYTES Function</td>
<td>Returns a RAW value containing a cryptographically secure pseudo-random sequence of bytes, and can be used to generate random material for encryption keys</td>
</tr>
<tr>
<td>RANDOMNUMBER Function</td>
<td>Returns a random BINARY_INTEGER</td>
</tr>
<tr>
<td>RANDOMNUMBER Function</td>
<td>Returns a random 128-bit integer of the NUMBER datatype</td>
</tr>
</tbody>
</table>

### 38.9.1 DECRYPT Function

This function decrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

**Syntax**

```sql
DBMS_CRYPTO.DECRYPT(
    src IN RAW,
    typ IN PLS_INTEGER,
    key IN RAW,
    iv IN RAW DEFAULT NULL)
RETURN RAW;
```
Pragmas
pragma restrict_references(decrypt,WNDS,RNDS,WNPS,RNPS);

Parameters

Table 38-11  DECRYPT Function Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>RAW data to be decrypted.</td>
</tr>
<tr>
<td>typ</td>
<td>Stream or block cipher type and modifiers to be used.</td>
</tr>
<tr>
<td>key</td>
<td>Key to be used for decryption.</td>
</tr>
<tr>
<td>iv</td>
<td>Optional initialization vector for block ciphers. Default is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

- To retrieve original plaintext data, **DECRYPT** must be called with the same cipher, modifiers, key, and IV that was used to encrypt the data originally.

## See Also:

"Usage Notes" for the **ENCRYPT** function for additional information about the ciphers and modifiers available with this package.

- If **VARCHAR2** data is converted to **RAW** before encryption, then it must be converted back to the appropriate database character set by using the **UTL_I18N** package.

## See Also:

**DBMS_CRYPTO Operational Notes** for a discussion of the **VARCHAR2** to **RAW** conversion rules

38.9.2 DECRYPT Procedures

These procedures decrypt **LOB** data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

### Syntax

```
DBMS_CRYPTO.DECRYPT(
    dst IN OUT NOCOPY BLOB,
    src IN            BLOB,
    typ IN            PLS_INTEGER,
    key IN            RAW,
    iv  IN            RAW          DEFAULT NULL);
```

```
DBMS_CRYPTO.DECRYPT(
    dst IN OUT NOCOPY CLOB         CHARACTER SET ANY_CS,
    src IN            BLOB,
```
typ IN PLS_INTEGER,
key IN RAW,
iv IN RAW DEFAULT NULL);

Pragmas
pragma restrict_references(decrypt,WNDS,RNDS,WNPS,RNPS);

Parameters

Table 38-12 DECRYPT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>LOB locator of output data. The value in the output LOB &lt;dst&gt; will be overwritten.</td>
</tr>
<tr>
<td>src</td>
<td>LOB locator of input data.</td>
</tr>
<tr>
<td>typ</td>
<td>Stream or block cipher type and modifiers to be used.</td>
</tr>
<tr>
<td>key</td>
<td>Key to be used for decryption.</td>
</tr>
<tr>
<td>iv</td>
<td>Optional initialization vector for block ciphers. Default is all zeroes.</td>
</tr>
</tbody>
</table>

38.9.3 ENCRYPT Function

This function encrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```sql
DBMS_CRYPTO.ENCRYPT(
    src IN RAW,
    typ IN PLS_INTEGER,
    key IN RAW,
    iv IN RAW DEFAULT NULL)
RETURN RAW;
```

Pragmas
pragma restrict_references(encrypt,WNDS,RNDS,WNPS,RNPS);

Parameters

Table 38-13 ENCRYPT Function Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>RAW data to be encrypted.</td>
</tr>
<tr>
<td>typ</td>
<td>Stream or block cipher type and modifiers to be used.</td>
</tr>
<tr>
<td>key</td>
<td>Encryption key to be used for encrypting data.</td>
</tr>
<tr>
<td>iv</td>
<td>Optional initialization vector for block ciphers. Default is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

- Block ciphers may be modified with chaining and padding type modifiers. The chaining and padding type modifiers are added to the block cipher to produce a
cipher suite. Cipher Block Chaining (CBC) is the most commonly used chaining type, and PKCS #5 is the recommended padding type. See Table 38-7 and Table 38-8 for block cipher chaining and padding modifier constants that have been defined for this package.

- To improve readability, you can define your own package-level constants to represent the cipher suites you use for encryption and decryption. For example, the following example defines a cipher suite that uses DES, cipher block chaining mode, and no padding:

```sql
DES_CBC_NONE CONSTANT PLS_INTEGER := DBMS_CRYPTO.ENCRYPT_DES
  + DBMS_CRYPTO.CHAIN_CBC
  + DBMS_CRYPTO.PAD_NONE;
```

See Table 38-6 for the block cipher suites already defined as constants for this package.

- To encrypt `VARCHAR2` data, it should first be converted to the AL32UTF8 character set.

> See Also:
The discussion of conversion rules under `DBMS_CRYPTO Operational Notes`

- Stream ciphers, such as RC4, are not recommended for stored data encryption.

### 38.9.4 ENCRYPT Procedures

These procedures encrypt LOB data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

**Syntax**

```sql
DBMS_CRYPTO.ENCRYPT(
  dst IN OUT NOCOPY BLOB,
  src IN            BLOB,
  typ IN            PLS_INTEGER,
  key IN            RAW,
  iv  IN            RAW          DEFAULT NULL);

DBMS_CRYPTO.ENCRYPT(
  dst IN OUT NOCOPY BLOB,
  src IN            CLOB         CHARACTER SET ANY_CS,
  typ IN            PLS_INTEGER,
  key IN            RAW,
  iv  IN            RAW          DEFAULT NULL);
```

**Pragmas**

```sql
pragma restrict_references(encrypt,WNDS,RNDS,WNPS,RNPS);
```

**Parameters**

**Table 38-14   ENCRYPT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>LOB locator of output data. The value in the output LOB <code>&lt;dst&gt;</code> will be overwritten.</td>
</tr>
<tr>
<td>src</td>
<td>LOB locator of input data.</td>
</tr>
</tbody>
</table>
Table 38-14  (Cont.) ENCRYPT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typ</td>
<td>Stream or block cipher type and modifiers to be used.</td>
</tr>
<tr>
<td>key</td>
<td>Encryption key to be used for encrypting data.</td>
</tr>
<tr>
<td>iv</td>
<td>Optional initialization vector for block ciphers. Default is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

See DBMS_DEBUG Operational Notes for more information about the conversion rules for the ENCRYPT procedure.

38.9.5 HASH Function

A one-way hash function takes a variable-length input string, the data, and converts it to a fixed-length (generally smaller) output string called a hash value. The hash value serves as a unique identifier (like a fingerprint) of the input data. You can use the hash value to verify whether data has been changed or not.

Note that a one-way hash function is a hash function that works in one direction. It is easy to compute a hash value from the input data, but it is hard to generate data that hashes to a particular value. Consequently, one-way hash functions work well to ensure data integrity. Refer to "When to Use Hash or Message Authentication Code (MAC) Functions" in DBMS_CRYPTO Operational Notes for more information about using one-way hash functions.

This function applies to data one of the supported cryptographic hash algorithms listed in Table 38-3.

Syntax

```
DBMS_CRYPTO.Hash (
   src IN RAW,
   typ IN PLS_INTEGER)
RETURN RAW;
```

```
DBMS_CRYPTO.Hash (
   src IN BLOB,
   typ IN PLS_INTEGER)
RETURN RAW;
```

```
DBMS_CRYPTO.Hash (
   src IN CLOB CHARACTER SET ANY_CS,
   typ IN PLS_INTEGER)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(hash,WNDS,RNDS,WNPS,RNPS);
```
Parameters

Table 38-15   HASH Function Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>The source data to be hashed.</td>
</tr>
<tr>
<td>typ</td>
<td>The hash algorithm to be used.</td>
</tr>
</tbody>
</table>

Usage Note

Oracle recommends that you use the SHA-1 (Secure Hash Algorithm) or SHA-2 because it is more resistant to brute-force attacks than MD4 or MD5. If you must use a Message Digest algorithm, then MD5 provides greater security than MD4.

38.9.6 MAC Function

This function applies Message Authentication Code (MAC) algorithms to data to provide keyed message protection.

A MAC is a key-dependent one-way hash function. MACs have the same properties as the one-way hash function described in HASH Function, but they also include a key. Only someone with the identical key can verify the hash. Also refer to “When to Use Hash or Message Authentication Code (MAC) Functions” in DBMS_CRYPTO Operational Notes for more information about using MACs.

See Table 38-4 for a list of MAC algorithms that have been defined for this package.

Syntax

```sql
DBMS_CRYPTO.MAC (
    src IN RAW,
    typ IN PLS_INTEGER,
    key IN RAW
) RETURN RAW;

DBMS_CRYPTO.MAC (
    src IN BLOB,
    typ IN PLS_INTEGER,
    key IN RAW
) RETURN RAW;

DBMS_CRYPTO.MAC (
    src IN CLOB CHARACTER SET ANY_CS,
    typ IN PLS_INTEGER,
    key IN RAW
) RETURN RAW;
```

Pragmas

`pragma restrict_references(mac,WNDS,RNDS,WNPS,RNPS);`
Parameters

Table 38-16  MAC Function Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Source data to which MAC algorithms are to be applied.</td>
</tr>
<tr>
<td>typ</td>
<td>MAC algorithm to be used.</td>
</tr>
<tr>
<td>key</td>
<td>Key to be used for MAC algorithm.</td>
</tr>
</tbody>
</table>

38.9.7 RANDOMBYTES Function

This function returns a RAW value containing a cryptographically secure pseudo-random sequence of bytes, which can be used to generate random material for encryption keys.

The RANDOMBYTES function is based on the RSA X9.31 PRNG (Pseudo-Random Number Generator).

Syntax

```
DBMS_CRYPTO.RANDOMBYTES (
    number_bytes IN POSITIVE)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(randombytes,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 38-17  RANDOMBYTES Function Parameter

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number_bytes</td>
<td>The number of pseudo-random bytes to be generated.</td>
</tr>
</tbody>
</table>

Usage Note

The number_bytes value should not exceed the maximum length of a RAW variable.

38.9.8 RANDOMINTEGER Function

This function returns an integer in the complete range available for the Oracle BINARY_INTEGER datatype.

Syntax

```
DBMS_CRYPTO.RANDOMINTEGER
RETURN BINARY_INTEGER;
```

Pragmas

```
pragma restrict_references(randominteger,WNDS,RNDS,WNPS,RNPS);
```
38.9.9 RANDOMNUMBER Function

This function returns an integer in the Oracle NUMBER datatype in the range of [0..2**128-1].

Syntax

DBMS_CRYPTO.RANDOMNUMBER
RETURN NUMBER;

Pragmas

pragma restrict_references(randomnumber,WNDS,RNDS,WNPS,RNPS);
The `DBMS_CSX_ADMIN` package provides an interface to customize the setup when transporting a tablespace containing binary XML data.

The chapter contains the following topics:

- Overview
- Security Model
- Constants
- Summary of `DBMS_CSX_ADMIN`

### 39.1 DBMS_CSX_ADMIN Overview

This package can be used by DBAs to customize the setup when transporting a tablespace containing binary XML data. The use of the package is not required in order for a transportable tablespace job to run.

By default, all binary XML tables will use the default token table set, which will be replicated during transport on the target database. To avoid the cost of transporting a potentially large token table set, the DBA may opt for registering a new set of token tables for a given tablespace. The package provides routines for token table set registration and lookup.

### 39.2 DBMS_CSX_ADMIN Security Model

Owned by `XDB`, the `DBMS_CSX_ADMIN` package must be created by `SYS` or `XDB`. The `EXECUTE` privilege is granted to `SYS` or `XDB` or `DBA`. Subprograms in this package are executed using the privileges of the current user.

### 39.3 DBMS_CSX_ADMIN Constants

The `DBMS_CSX_ADMIN` package uses several constants.

These are described in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_LEVEL</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>Default token table</td>
</tr>
<tr>
<td>TAB_LEVEL</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Token table set associated with tables, not tablespaces</td>
</tr>
<tr>
<td>TBS_LEVEL</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Token table set associated with a tablespace</td>
</tr>
<tr>
<td>NO_CREATE</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>Token tables already exist, associate them with the given table/tablespace</td>
</tr>
</tbody>
</table>
### Table 39-1  (Cont.) DBMS_CSX_ADMIN Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_INDEXES</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Do not create indexes on the new set of token tables</td>
</tr>
<tr>
<td>WITH_INDEXES</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Create indexes on the token tables</td>
</tr>
<tr>
<td>DEFAULT_TOKS</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>Prepopulate the token tables with default token mappings</td>
</tr>
<tr>
<td>NO_DEFAULT_TOKS</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Do not prepopulate the token tables with default token mappings</td>
</tr>
</tbody>
</table>

### 39.4 Summary of DBMS_CSX_ADMIN Subprograms

This table lists and describes the **DBMS_CSX_ADMIN** Package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETTOKENTABLEINFO Procedure &amp; Function</td>
<td>Returns the GUID of the token table set where token mappings for this table</td>
</tr>
<tr>
<td>GETTOKENTABLEINFOBYTABLES PACE Procedure</td>
<td>Returns the GUID and the token table names for this tablespace</td>
</tr>
<tr>
<td>NAMESPACEIDTABLE Function</td>
<td>Returns default namespace-ID token table</td>
</tr>
<tr>
<td>PATHIDTABLE Function</td>
<td>Returns the default path-ID token table</td>
</tr>
<tr>
<td>QNAMEIDTABLE Function</td>
<td>Returns the default qname-ID token table</td>
</tr>
</tbody>
</table>

### 39.4.1 GETTOKENTABLEINFO Procedure & Function

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

Given the table name and the owner, the first overload of the procedure returns the globally unique identifier (GUID) of the token table set where token mappings for this table can be found. The procedure returns also the names of the token tables, and whether the token table set is the default one.

Given the object number of a table, the second overload of the procedure returns the GUID of the token table set used by the table, and whether this is the default token table set.

#### Syntax

```sql
DBMS_CSX_ADMIN.GETTOKENTABLEINFO (  
  ownername      IN   VARCHAR2,  
  tablename      IN   VARCHAR2,  
  guid           OUT  RAW,  
  qnametable     OUT  VARCHAR2,  
  nmspctable     OUT  VARCHAR2,  
  level          OUT  NUMBER,  
  tabno          OUT  NUMBER);
```
DBMS_CSX_ADMIN.GETTOKENTABLEINFO (  
  tabno            IN    NUMBER,  
  guid             OUT   RAW);  
RETURN BOOLEAN;

Parameters

Table 39-3  GETTOKENTABLEINFO Procedure & Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>Owner of the table</td>
</tr>
<tr>
<td>tablename</td>
<td>Name of the table</td>
</tr>
<tr>
<td>guid</td>
<td>GUID of the token table set used by the given table</td>
</tr>
<tr>
<td>qnametable</td>
<td>Name of the qname-ID table in the new set</td>
</tr>
<tr>
<td>nmspctable</td>
<td>Name of the namespace-ID table in the new set</td>
</tr>
<tr>
<td>level</td>
<td>DEFAULT_LEVEL if default token table set, TBS_LEVEL if same token table set is used by all tables in the same tablespace as the given table, TAB_LEVEL otherwise</td>
</tr>
<tr>
<td>tabno</td>
<td>Table object number</td>
</tr>
</tbody>
</table>

39.4.2 GETTOKENTABLEINFOBYTABLESPACE Procedure

Given a tablespace number, this procedure returns the GUID and the token table names for this tablespace.

Syntax

DBMS_CSX_ADMIN.GETTOKENTABLEINFOBYTABLESPACE (  
  tname            IN    VARCHAR2,  
  tablespaceno     IN    NUMBER,  
  guid             OUT   RAW,  
  qnametable       OUT   VARCHAR2,  
  nmspctable       OUT   VARCHAR2,  
  isdefault        OUT   BOOLEAN,  
  containTokTab    OUT   BOOLEAN);  

Parameters

Table 39-4  GETTOKENTABLEINFOBYTABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tname</td>
<td>Tablespace name</td>
</tr>
<tr>
<td>tablespaceno</td>
<td>Tablespace number</td>
</tr>
<tr>
<td>guid</td>
<td>GUID of the token table set associated with this tablespace (if any)</td>
</tr>
<tr>
<td>qnametable</td>
<td>Name of the qname-ID table</td>
</tr>
<tr>
<td>nmspctable</td>
<td>Name of the namespace-ID table</td>
</tr>
<tr>
<td>isdefault</td>
<td>TRUE if the token table is the default one</td>
</tr>
<tr>
<td>containTokTab</td>
<td>TRUE if the tablespace contains its own token table set</td>
</tr>
</tbody>
</table>
39.4.3 NAMESPACEIDTABLE Function

This procedure returns default namespace-ID token table.

Syntax

```sql
DBMS_CSX_ADMIN.NAMESPACEIDTABLE
RETURN VARCHAR2;
```

39.4.4 PATHIDTABLE Function

This procedure returns the default path-ID token table. This is used for granting permissions on the default path-ID token table for a user before executing `EXPLAIN PLAN` for a query on an XML table with an XML index.

Syntax

```sql
DBMS_CSX_ADMIN.PATHIDTABLE
RETURN VARCHAR2;
```

39.4.5 QNAMEIDTABLE Function

This procedure returns the default qname-ID token table.

Syntax

```sql
DBMS_CSX_ADMIN.QNAMEIDTABLE
RETURN VARCHAR2;
```
40

DBMS_CUBE

DBMS_CUBE contains subprograms that create OLAP cubes and dimensions, and that load and process the data for querying.

See Also:

OLAP Technology in the Oracle Database in Oracle OLAP User’s Guide regarding use of the OLAP option to support business intelligence and analytical applications.

This chapter contains the following topics:

- Using DBMS_CUBE
- Using SQL Aggregation Management
- Upgrading 10g Analytic Workspaces
- Summary of DBMS_CUBE Subprograms

40.1 Using DBMS_CUBE

Cubes and cube dimensions are first class data objects that support multidimensional analytics. They are stored in a container called an analytic workspace. Multidimensional objects and analytics are available with the OLAP option to Oracle Database.

Cubes can be enabled as cube materialized views for automatic refresh of the cubes and dimensions, and for query rewrite. Several DBMS_CUBE subprograms support the creation and maintenance of cube materialized views as a replacement for relational materialized views. These subprograms are discussed in “Using SQL Aggregation Management”.

The metadata for cubes and dimensions is defined in XML documents, called templates, which you can derive from relational materialized views using the CREATE_CUBE or DERIVE_FROM_MVIEW functions. Using a graphical tool named Analytic Workspace Manager, you can enhance the cube with analytic content or create the metadata for new cubes and cube dimensions from scratch.

Several other DBMS_CUBE subprograms provide a SQL alternative to Analytic Workspace Manager for creating an analytic workspace from an XML template and for refreshing the data stored in cubes and dimensions. The IMPORT XML procedure creates an analytic workspace with its cubes and cube dimensions from an XML template. The BUILD procedure loads data into the cubes and dimensions from their data sources and performs whatever processing steps are needed to prepare the data for querying.
40.1.1 DBMS_CUBE Security Model

Certain roles and system privileges are required to use the DBMS_CUBE package.

To create dimensional objects in the user's own schema:

- **OLAP_USER** role
- **CREATE SESSION** privilege

To create dimensional objects in different schemas:

- **OLAP_DBA** role
- **CREATE SESSION** privilege

To create cube materialized views in the user's own schema:

- **CREATE MATERIALIZED VIEW** privilege
- **CREATE DIMENSION** privilege
- **ADVISOR** privilege

To create cube materialized views in different schemas:

- **CREATE ANY MATERIALIZED VIEW** privilege
- **CREATE ANY DIMENSION** privilege
- **ADVISOR** privilege

If the source tables are in a different schema, then the owner of the dimensional objects needs **SELECT** object privileges on those tables.

40.2 Using SQL Aggregation Management

SQL Aggregation Management is a group of PL/SQL subprograms in DBMS_CUBE that supports the rapid deployment of cube materialized views from existing relational materialized views.

Cube materialized views are cubes that have been enhanced to use the automatic refresh and query rewrite features of Oracle Database. A single cube materialized view can replace many of the relational materialized views of summaries on a fact table, providing uniform response time to all summary data.

Cube materialized views bring the fast update and fast query capabilities of the OLAP option to applications that query summaries of detail relational tables. The summary data is generated and stored in a cube, and query rewrite automatically redirects queries to the cube materialized views. Applications experience excellent querying performance.

In the process of creating the cube materialized views, DBMS_CUBE also creates a fully functional analytic workspace including a cube and the cube dimensions. The cube stores the data for a cube materialized view instead of the table that stores the data for a relational materialized view. A cube can also support a wide range of analytic functions that enhance the database with information-rich content.
Cube materialized views are registered in the data dictionary along with all other materialized views. A C$ prefix identifies a cube materialized view.

The DBMS_CUBE subprograms also support life-cycle management of cube materialized views.

**See Also:**

Adding Materialized View Capability to a Cube in Oracle OLAP User’s Guide for more information about cube materialized views and enhanced OLAP analytics.

40.2.1 Subprograms in SQL Aggregation Management

SQL Aggregation Management includes four subprograms.

- CREATE_MVIEW Function
- DERIVE_FROM_MVIEW Function
- DROP_MVIEW Procedure
- REFRESH_MVIEW Procedure

40.2.2 Requirements for the Relational Materialized View

SQL Aggregation Management uses an existing relational materialized view to derive all the information needed to generate a cube materialized view. The relational materialized view determines the detail level of data that is stored in the cube materialized view. The related relational dimension objects determine the scope of the aggregates, from the lowest level specified in the GROUP BY clause of the materialized view subquery, to the highest level of the dimension hierarchy.

The relational materialized view must conform to these requirements:

- Explicit GROUP BY clause for one or more columns.
- No expressions in the select list or GROUP BY clause.
- At least one of these numeric aggregation methods: SUM, MIN, MAX, or AVG.
- No outer joins.
- Summary keys with at least one simple column associated with a relational dimension.
  
  or

  Summary keys with at least one simple column and no hierarchies or levels.
- Numeric datatype of any type for the fact columns. All facts are converted to NUMBER.
- Eligible for rewrite. REWRITE_CAPABILITY should be GENERAL; it cannot be NONE. Refer to the ALL_MVIEWS entry in the Oracle Database Reference.
- Cannot use the DISTINCT or UNIQUE keywords with an aggregate function in the defining query. For example, AVG(DISTINCT units) causes an error in STRICT mode and is ignored in LOOSE mode.

You can choose between two modes when rendering the cube materialized view, LOOSE and STRICT. In STRICT mode, any deviation from the requirements raises an exception and
prevents the materialized view from being created. In **LOOSE** mode (the default), some deviations are allowed, but they affect the content of the materialized view. These elements in the relational materialized view generate warning messages:

- Complex expressions in the defining query are ignored and do not appear in the cube materialized view.
- The **AVG** function is changed to **SUM** and **COUNT**.
- The **COUNT** function without a **SUM**, **MIN**, **MAX**, or **AVG** function is ignored.
- The **STDDEV** and **VARIANCE** functions are ignored.

You can also choose how conditions in the **WHERE** clause are filtered. When filtering is turned off, the conditions are ignored. When turned on, valid conditions are rendered in the cube materialized view, but asymmetric conditions among dimension levels raise an exception.

### 40.2.3 Permissions for Managing and Querying Cube Materialized Views

Certain permissions are required to manage and query cube materialized views. To create cube materialized views, you must have these privileges:

- **CREATE [ANY] MATERIALIZED VIEW** privilege
- **CREATE [ANY] DIMENSION** privilege
- **ADVISOR** privilege

To access cube materialized views from another schema using query rewrite, you must have these privileges:

- **GLOBAL QUERY REWRITE** privilege
- **SELECT** or **READ** privilege on the relational source tables
- **SELECT** or **READ** privilege on the analytic workspace (**AW$**name) that supports the cube materialized view
- **SELECT** or **READ** privilege on the cube
- **SELECT** or **READ** privilege on the dimensions of the cube

Note that you need **SELECT** or **READ** privileges on the database objects that support the cube materialized views, but not on the cube materialized views.

### 40.2.4 Example of SQL Aggregation Management

Six examples of SQL Aggregate Management are given. All these examples use the sample Sales History schema, which is installed in Oracle Database with two relational materialized views: **CAL_MONTH_SALES_MV** and **FWEEK_PSCAT_SALES_MV**.

- About Relational Materialized View **CAL_MONTH_SALES_MV**
- Creating the Cube Materialized View
- Disabling the Relational Materialized Views
- Creating Execution Plans for Cube Materialized Views
• Maintaining Cube Materialized Views
• New Database Objects

About Relational Materialized View CAL_MONTH_SALES_MV

This example uses CAL_MONTH_SALES_MV as the basis for creating a cube materialized view. The following query was used to create CAL_MONTH_SALES_MV. CAL_MONTH_SALES_MV summarizes the daily sales data stored in the SALES table by month.

```
SELECT query FROM user_mviews
WHERE mview_name='CAL_MONTH_SALES_MV';
```

```
---
SELECT   t.calendar_month_desc
,        sum(s.amount_sold) AS dollars
FROM     sales s
,        times t
WHERE    s.time_id = t.time_id
GROUP BY t.calendar_month_desc
---
```

DBMS_CUBE uses relational dimensions to derive levels and hierarchies for the cube materialized view. The SH schema has relational dimensions for most dimension tables in the schema, as shown by the following query.

```
SELECT dimension_name FROM user_dimensions;
```

```
---
CUSTOMERS_DIM
PRODUCTS_DIM
TIMES_DIM
CHANNELS_DIM
PROMOTIONS_DIM
---
```

Creating the Cube Materialized View

This PL/SQL script uses the CREATE_MVIEW function to create a cube materialized view from CAL_MONTH_SALES_MV. CREATE_MVIEW sets the optional BUILD parameter to refresh the cube materialized view immediately.

```
SET serverout ON format wrapped
DECLARE
  salesaw  varchar2(30);
BEGIN
  salesaw := dbms_cube.create_mview('SH', 'CAL_MONTH_SALES_MV',
    'build=immediate');
END;
/
```

These messages confirm that the script created and refreshed CB$CAL_MONTH_SALES successfully:

Completed refresh of cube mview "SH","CB$CAL_MONTH_SALES" at 20130212 08:42:58.003.
Created cube organized materialized view "CB$CAL_MONTH_SALES" for rewrite at 200130212 08:42:58.004.
The following query lists the materialized views in the SH schema:

```
SELECT mview_name FROM user_mviews;
```

<table>
<thead>
<tr>
<th>MVIEW_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB$CAL_MONTH_SALES</td>
</tr>
<tr>
<td>CB$TIMES_DIM_D1_CAL_ROLLUP</td>
</tr>
<tr>
<td>CAL_MONTH_SALES_MV</td>
</tr>
<tr>
<td>FWEEK_PSCAT_SALES_MV</td>
</tr>
</tbody>
</table>

Two new materialized views are registered in the data dictionary:

- CB$CAL_MONTH_SALES: Cube materialized view
- CB$TIMES_DIM_D1_CAL_ROLLUP: Cube dimension materialized view for the TIME_DIM Calendar Rollup hierarchy

Cube dimension materialized views support refresh of the cube materialized view. You do not directly administer dimension materialized views.

Disabling the Relational Materialized Views

After creating a cube materialized view, disable query rewrite on all relational materialized views for the facts now supported by the cube materialized view. You can drop them when you are sure that you created the cube materialized view with the optimal parameters.

```
ALTER MATERIALIZED VIEW cal_month_sales_mv DISABLE QUERY REWRITE;
```

Materialized view altered.

You can also use the DISABLEQRW parameter in the CREATE_MVIEW function, which disables query rewrite on the source materialized view as described in Table 40-7.

Creating Execution Plans for Cube Materialized Views

You can create execution plans for cube materialized views the same as for relational materialized views. The following command generates an execution plan for a query against the SALES table, which contains data at the day level. The answer set requires data summarized by quarter. Query rewrite would not use the original relational materialized view for this query, because its data is summarized by month. However, query rewrite can use the new cube materialized view for summary data for months, quarters, years, and all years.

```
EXPLAIN PLAN FOR SELECT
t.calendar_quarter_desc, 
sum(s.amount_sold) AS dollars 
FROM sales s, 
times t 
WHERE s.time_id = t.time_id 
AND t.calendar_quarter_desc LIKE '2001%' 
GROUP BY t.calendar_quarter_desc 
ORDER BY t.calendar_quarter_desc;
```

The query returns these results:

<table>
<thead>
<tr>
<th>CALENDAR_QUARTER_DESC</th>
<th>DOLLARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001-01</td>
<td>6547097.44</td>
</tr>
<tr>
<td>2001-02</td>
<td>6922468.39</td>
</tr>
</tbody>
</table>
The execution plan shows that query rewrite returned the summary data from the cube materialized view, \texttt{CB\$CAL\_MONTH\_SALES}, instead of recalculating it from the \texttt{SALES} table.

```
SELECT plan_table_output FROM TABLE(dbms_xplan.display());
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name               | Rows  | Bytes | Cost (%CPU) | Time     |
------------------------------------------------------------------------------------------------------------------
| 0   | SELECT STATEMENT               |                    | 1     | 30    | 3 (34)       | 00:00:01 |
| 1   |  SORT GROUP BY                 |                    | 1     | 30    | 3 (34)       | 00:00:01 |
|* 2  |   MAT\_VIEW\_REWRITE CUBE ACCESS | CB\$CAL\_MONTH\_SALES | 1     | 30    | 2 (0)        | 00:00:01 |
------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
2 - filter("CB\$CAL\_MONTH\_SALES"."D1\_CALENDAR\_QUARTER\_DESC" LIKE '2001%' AND "CB\$CAL\_MONTH\_SALES"."SYS\_GID"=63)
```

15 rows selected.

### Maintaining Cube Materialized Views

You can create a cube materialized view that refreshes automatically. However, you can force a refresh at any time using the \texttt{REFRESH\_MVIEW} Procedure:

```
BEGIN
    dbms_cube.refresh_mview('SH', 'CB$CAL\_MONTH\_SALES');
END;
/
```

Completed refresh of cube mview "SH"."CB\$CAL\_MONTH\_SALES" at 20130212 14:30:59.534.

If you want to drop a cube materialized view, use the \texttt{DROP\_MVIEW} Procedure so that all supporting database objects (analytic workspace, cube, cube dimensions, and so forth) are also dropped:

```
BEGIN
    dbms_cube.drop_mview('SH', 'CB$CAL\_MONTH\_SALES');
END;
/
```

Dropped cube organized materialized view "SH"."CAL\_MONTH\_SALES" including container analytic workspace "SH"."CAL\_MONTH\_SALES\_AW" at 20130212 13:38:47.878.

### New Database Objects

The \texttt{CREATE\_MVIEW} function creates several first class database objects in addition to the cube materialized views. You can explore these objects through the data dictionary by querying views such as \texttt{ALL\_CUBES} and \texttt{ALL\_CUBE\_DIMENSIONS}.

This example created the following supporting objects:

- **Analytic workspace** \texttt{CAL\_MONTH\_SALES\_AW} (\texttt{AW$CAL\_MONTH\_SALES\_AW} table)
- **Cube** \texttt{CAL\_MONTH\_SALES}
• Cube dimension TIMES_DIM_D1
• Dimension hierarchy CAL_ROLLUP
• Dimension levels ALL_TIMES_DIM, YEAR, QUARTER, and MONTH
• Numerous attributes for levels in the CAL_ROLLUP hierarchy

40.3 Upgrading 10g Analytic Workspaces

You can upgrade an OLAP 10g analytic workspace to OLAP 12c by saving the OLAP 10g objects as an XML template and importing the XML into a different schema. The original analytic workspace remains accessible and unchanged by the upgrade process.

Oracle OLAP metadata is the same in OLAP 11g and OLAP 12c so you do not need to upgrade an OLAP 11g analytic workspace to OLAP 12c. This topic describes upgrading an Oracle OLAP 10g analytic workspace to OLAP 12c.

Tip:
Oracle recommends using Analytic Workspace Manager for performing upgrades. See Upgrading Metadata From Oracle OLAP 10g in Oracle OLAP User’s Guide.

These subprograms in DBMS_CUBE support the upgrade process:

• CREATE_EXPORT_OPTIONS Procedure
• CREATE_IMPORT_OPTIONS Procedure
• EXPORT_XML Procedure
• EXPORT_XML_TO_FILE Procedure
• IMPORT_XML Procedure
• INITIALIZE_CUBE_UPGRADE Procedure
• UPGRADE_AW Procedure

Prerequisites:

• The OLAP 10g analytic workspace can use OLAP standard form metadata.
• Customizations to the OLAP 10g analytic workspace may not be exported to the XML template. You must re-create them in OLAP 12c.
• The original relational source data must be available to load into the new analytic workspace. If the data is in a different schema or the table names are different, then you must remap the dimensional objects to the new relational sources after the upgrade.
• You can create the OLAP 12c analytic workspace in the same schema as the OLAP 10g analytic workspace. However, if you prefer to create it in a different schema, then create a new user with the following privileges:
  – SELECT or READ privileges on the OLAP 10g analytic workspace (GRANT SELECT ON schema.AW$analytic_workspace).
- SELECT or READ privileges on all database tables and views that contain the source data for the OLAP 10g analytic workspace.
- Appropriate privileges for an OLAP administrator.
- Same default tablespace as the Oracle 10g user.

See the Oracle OLAP User's Guide.

Correcting Naming Conflicts

The namespaces are different in OLAP 10g than those in OLAP 12c. For a successful upgrade, you must identify any 10g object names that are used multiple times under the 12c naming rules and provide unique names for them.

The following namespaces control the uniqueness of OLAP object names in Oracle 12c:

- **Schema**: The names of cubes, dimensions, and measure folders must be unique within a schema. They cannot conflict with the names of tables, views, indexes, relational dimensions, or any other first class objects. However, these OLAP 12c object names do not need to be distinct from 10g object names, because they are in different namespaces.
- **Cube**: The names of measures must be unique within a cube.
- **Dimension**: The names of hierarchies, levels, and attributes must be unique within a dimension. For example, a dimension cannot have a hierarchy named Customers and a level named Customers.

You can use an initialization table and a rename table to rename objects in the upgraded 12c analytic workspace.

Initialization Table

The INITIALIZE_CUBE_UPGRADE procedure identifies ambiguous names under the OLAP 12c naming rules. For example, a 10g dimension might have a hierarchy and a level with the same name. Because hierarchies and levels are in the same 12c namespace, the name is not unique in 12c; to a 12c client, the hierarchy and the level cannot be differentiated by name.

INITIALIZE_CUBE_UPGRADE creates and populates a table named CUBE_UPGRADE_INFO with unique names for these levels, hierarchies, and attributes. By using the unique names provided in the table, a 12c client can browse the OLAP 12c metadata. You cannot attach an OLAP 12c client to the analytic workspace or perform an upgrade without a CUBE_UPGRADE_INFO table, if the 10g metadata contains ambiguous names.

You can edit CUBE_UPGRADE_INFO to change the default unique names to names of your choosing. You can also add rows to change the names of any other objects. When using an 12c client, you see the new object names. When using an 10g client, you see the original names. However, the INITIALIZE_CUBE_UPGRADE procedure overwrites this table, so you may prefer to enter customizations in a rename table.

During an upgrade from OLAP 10g, the unique object names in CUBE_UPGRADE_INFO are used as the names of 12c objects in the new analytic workspace. However, INITIALIZE_CUBE_UPGRADE does not automatically provide unique names for cubes, dimensions, and measure folders. To complete an upgrade, you must assure that these objects have unique names within the 12c namespace. You can provide these objects with new names in the CUBE_UPGRADE_INFO table or in a rename table.
OLAP 12c clients automatically use CUBE_UPGRADE_INFO when it exists in the same schema as the OLAP 10g analytic workspace.

See Also:
"INITIALIZE_CUBE_UPGRADE Procedure"

Rename Table

You can create a rename table that contains new object names for an OLAP 12c analytic workspace. You can then use the rename table in the CREATE_IMPORT_OPTIONS and UPGRADE_AW procedures.

When upgrading within the same schema, you must provide a unique name for the 12c analytic workspace. The UPGRADE_AW procedure provides a parameter for this purpose; otherwise, you must provide the new name in the rename table. The duplication of cube names does not create ambiguity because the 12c cubes are created in a different namespace than the 10g cubes.

The names provided in a rename table are used only during an upgrade and overwrite any names entered in the CUBE_UPGRADE_INFO table.

To create a rename table:

1. Open SQL*Plus or another SQL client, and connect to Oracle Database as the owner of the 10g analytic workspace.
2. Issue a command like the following:

   ```sql
   CREATE TABLE table_name (  
     source_id    VARCHAR2(300),  
     new_name     VARCHAR2(30),  
     object_type  VARCHAR2(30));
   ```

3. Populate the rename table with the appropriate values, as follows.

   * `table_name` is the name of the rename table.
   * `source_id` is the identifier for an object described in the XML document supplied to IMPORT_XML. The identifier must have this format:

     `schema_name.object_name[.subobject_name]`

   * `new_name` is the object name given during the import to the object specified by `source_id`.
   * `object_type` is the object type as described in the XML, such as StandardDimension or DerivedMeasure.

   For example, these SQL statements populate the table with new names for the analytic workspace, a cube, and four dimensions:

   ```sql
   INSERT INTO my_object_map VALUES('GLOBAL_AW.GLOBAL10.AW', 'GLOBAL12', 'AW');
   INSERT INTO my_object_map VALUES('GLOBAL_AW.UNITS_CUBE', 'UNIT_SALES_CUBE', 'Cube');
   INSERT INTO my_object_map VALUES('GLOBAL_AW.CUSTOMER', 'CUSTOMERS', 'StandardDimension');
   ```
See Also:

"CREATE_IMPORT_OPTIONS Procedure"

Simple Upgrade

A simple upgrade creates an OLAP 12c analytic workspace from an OLAP 10g analytic workspace.

To perform a simple upgrade of an Oracle OLAP 10g analytic workspace:

1. Open SQL*Plus or a similar SQL command-line interface and connect to Oracle Database 12c as the schema owner for the OLAP 12c analytic workspace.
2. To rename any objects in the 12c analytic workspace, create a rename table as described in the Rename Table section. (Optional)
3. Perform the upgrade, as described in "UPGRADE_AW Procedure".
4. Use the DBMS_CUBE.BUILD procedure to load data into the cube.

Example 40-1  Performing a Simple Upgrade to the GLOBAL Analytic Workspace

This example creates an OLAP 12c analytic workspace named GLOBAL12 from an OLAP 10g analytic workspace named GLOBAL10. GLOBAL10 contains no naming conflicts between cubes, dimensions, measure folders, or tables in the schema, so a rename table is not needed in this example.

BEGIN

   -- Upgrade the analytic workspace
   dbms_cube.upgrade_aw(sourceaw =>'GLOBAL10', destaw => 'GLOBAL12');

   -- Load and aggregate the data
   dbms_cube.build(script => 'UNITS_CUBE, PRICE_AND_COST_CUBE');

END;
/

Custom Upgrade

A custom upgrade enables you to set the export and import options.

To perform a custom upgrade of an Oracle OLAP 10g analytic workspace:

1. Open SQL*Plus or a similar SQL command-line interface and connect to Oracle Database 12c as the schema owner of the OLAP 12c analytic workspace.
2. Generate an initialization table, as described in the Initialization Table section. Review the new, default object names and modify them as desired.
3. Create a rename table, as described in the Rename Table section. If you are upgrading in the same schema, you must use a rename table to provide a unique name for the 12c analytic workspace. Otherwise, a rename table is needed only if names are duplicated among the cubes, dimensions, and measure folders of the analytic workspace, or
between those names and the existing cubes, dimensions, measure folders, or
tables of the destination schema.

4. Create a SQL script that does the following:
   a. Create an XML document for the export options, as described in
      "CREATE_EXPORT_OPTIONS Procedure". The SUPPRESS_NAMESPACE option
      must be set to TRUE for the upgrade to occur.
   b. Create an XML document for the import options, as described in
      "CREATE_IMPORT_OPTIONS Procedure".
   c. Create an XML template in OLAP 12c format, as described in "EXPORT_XML
      Procedure".
   d. Create an OLAP 12c analytic workspace from the XML template, as described
      in "IMPORT_XML Procedure".

5. Load and aggregate the data in the new analytic workspace, as described in
   "BUILD Procedure".

Example 40-2  Performing a Custom Upgrade to the GLOBAL Analytic
Workspace

This example upgrades the GLOBAL10 analytic workspace from OLAP 10g metadata to
OLAP 12c metadata in the GLOBAL_AW schema.

The rename table provides the new name of the analytic workspace. These
commands define the rename table.

```sql
CREATE TABLE my_object_map(
    source_id    VARCHAR2(300),
    new_name     VARCHAR2(30),
    object_type  VARCHAR2(30));
INSERT INTO my_object_map VALUES('GLOBAL_AW.GLOBAL10.AW',  'GLOBAL12', 'AW');
COMMIT;
```

Following is the script for performing the upgrade.

```sql
set serverout on
DECLARE
    importClob    clob;
    exportClob    clob;
    exportOptClob clob;
    importOptClob clob;
BEGIN

    -- Create table of reconciled names
    dbms_cube.initialize_cube_upgrade;

    -- Create a CLOB containing the export options
    dbms_lob.createtemporary(exportOptClob, TRUE);
    dbms_cube.create_export_options(out_options_xml=>exportOptClob,
        suppress_namespace=>TRUE, preserve_table_owners=>TRUE);

    -- Create a CLOB containing the import options
    dbms_lob.createtemporary(importOptClob, TRUE);
    dbms_cube.create_import_options(out_options_xml=>importOptClob, rename_table
        => 'MY_OBJECT_MAP');

```
-- Create CLOBs for the metadata
dbms_lob.createtemporary(importClob, TRUE);
dbms_lob.createtemporary(exportClob, TRUE);

-- Export metadata from a 10g analytic workspace to a CLOB
dbms_cube.export_xml(object_ids=>'GLOBAL_AW', options_xml=>exportOptClob,
out_xml=>exportClob);

-- Import metadata from the CLOB
dbms_cube.import_xml(in_xml => exportClob, options_xml=>importOptClob,
out_xml=>importClob);

-- Load and aggregate the data
dbms_cube.build('UNITS_CUBE, PRICE_AND_COST_CUBE');

END;
/

40.4 Summary of DBMS_CUBE Subprograms

This table lists and describes the DBMS_CUBE procedure subprograms.

Table 40-1  DBMS_CUBE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD Procedure</td>
<td>Loads data into one or more cubes and dimensions, and prepares the data for querying.</td>
</tr>
<tr>
<td>CREATE_EXPORT_OPTIONS Procedure</td>
<td>Creates an input XML document of processing options for the EXPORT_XML procedure.</td>
</tr>
<tr>
<td>CREATE_IMPORT_OPTIONS Procedure</td>
<td>Creates an input XML document of processing options for the IMPORT_XML procedure.</td>
</tr>
<tr>
<td>CREATE_MVIEW Function</td>
<td>Creates a cube materialized view from the definition of a relational materialized view.</td>
</tr>
<tr>
<td>DERIVE_FROM_MVIEW Function</td>
<td>Creates an XML template for a cube materialized view from the definition of a relational materialized view.</td>
</tr>
<tr>
<td>DROP_MVIEW Procedure</td>
<td>Drops a cube materialized view.</td>
</tr>
<tr>
<td>EXPORT_XML Procedure</td>
<td>Exports the XML of an analytic workspace to a CLOB.</td>
</tr>
<tr>
<td>EXPORT_XML_TO_FILE Procedure</td>
<td>Exports the XML of an analytic workspace to a file.</td>
</tr>
<tr>
<td>IMPORT_XML Procedure</td>
<td>Creates, modifies, or drops an analytic workspace by using an XML template</td>
</tr>
<tr>
<td>INITIALIZE_CUBE_UPGRADE Procedure</td>
<td>Processes Oracle OLAP 10g objects with naming conflicts to enable Oracle 12c clients to access them.</td>
</tr>
<tr>
<td>REFRESH_MVIEW Procedure</td>
<td>Refreshes a cube materialized view.</td>
</tr>
<tr>
<td>UPGRADE_AW Procedure</td>
<td>Upgrades an analytic workspace from Oracle OLAP 10g to 12c.</td>
</tr>
<tr>
<td>VALIDATE_XML Procedure</td>
<td>Checks the XML to assure that it is valid, without committing the results to the database.</td>
</tr>
</tbody>
</table>
40.4.1 BUILD Procedure

This procedure loads data into one or more cubes and dimensions, and generates aggregate values in the cubes. The results are automatically committed to the database.

Syntax

```sql
DBMS_CUBE.BUILD (  
  script                IN  VARCHAR2,  
  method                IN  VARCHAR2        DEFAULT NULL,  
  refresh_after_errors  IN  BOOLEAN         DEFAULT FALSE,  
  parallelism           IN  BINARY_INTEGER  DEFAULT 0,  
  atomic_refresh        IN  BOOLEAN         DEFAULT FALSE,  
  automatic_order       IN  BOOLEAN         DEFAULT TRUE,  
  add_dimensions        IN  BOOLEAN         DEFAULT TRUE,  
  scheduler_job         IN  VARCHAR2        DEFAULT NULL,  
  master_build_id       IN  BINARY_INTEGER  DEFAULT 0,  
  nested                IN  BOOLEAN         DEFAULT FALSE),  
  job_class             IN  VARCHAR2        DEFAULT 'DEFAULT_JOB_CLASS'
)```

Parameters

Table 40-2  BUILD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script</td>
<td>A list of cubes and dimensions and their build options (see &quot;SCRIPT Parameter&quot;).</td>
</tr>
<tr>
<td>method</td>
<td>A full or a fast (partial) refresh. In a fast refresh, only changed rows are inserted in the cube and the affected areas of the cube are re-aggregated. You can specify a method for each cube and dimension in sequential order, or a single method to apply to all cubes and dimensions. If you list more objects than methods, then the last method applies to the additional objects.</td>
</tr>
<tr>
<td></td>
<td>• C: Complete refresh clears all dimension values before loading. (Default)</td>
</tr>
<tr>
<td></td>
<td>• F: Fast refresh of a cube materialized view, which performs an incremental refresh and re-aggregation of only changed rows in the source table.</td>
</tr>
<tr>
<td></td>
<td>• ?: Fast refresh if possible, and otherwise a complete refresh.</td>
</tr>
<tr>
<td></td>
<td>• P: Recomputes rows in a cube materialized view that are affected by changed partitions in the detail tables.</td>
</tr>
<tr>
<td></td>
<td>• S: Fast solve of a compressed cube. A fast solve reloads all the detail data and re-aggregates only the changed values.</td>
</tr>
<tr>
<td>refresh_after_errors</td>
<td>TRUE to roll back just the cube or dimension with errors, and then continue building the other objects.</td>
</tr>
<tr>
<td>parallelism</td>
<td>Number of parallel processes to allocate to this job (see Usage Notes).</td>
</tr>
</tbody>
</table>
### Table 40-2  (Cont.) BUILD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic_refresh</td>
<td>TRUE prevents users from accessing intermediate results during a build. It freezes the current state of an analytic workspace at the beginning of the build to provide current sessions with consistent data. This option thaws the analytic workspace at the end of the build to give new sessions access to the refreshed data. If an error occurs during the build, then all objects are rolled back to the frozen state. FALSE enables users to access intermediate results during an build.</td>
</tr>
<tr>
<td>automatic_order</td>
<td>TRUE enables optimization of the build order. Dimensions are loaded before cubes. FALSE builds objects in the order you list them in the script.</td>
</tr>
<tr>
<td>add_dimensions</td>
<td>TRUE automatically includes all the dimensions of the cubes in the build, whether or not you list them in the script. If a cube materialized view with a particular dimension is fresh, then that dimension is not reloaded. You can list a cube once in the script. FALSE includes only dimensions specifically listed in the script.</td>
</tr>
<tr>
<td>scheduler_job</td>
<td>Any text identifier for the job, which will appear in the log table. The string does not need to be unique.</td>
</tr>
<tr>
<td>master_build_id</td>
<td>A unique name for the build.</td>
</tr>
<tr>
<td>nested</td>
<td>TRUE performs nested refresh operations for the specified set of cube materialized views. Nested refresh operations refresh all the depending materialized views and the specified set of materialized views based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables. All objects must reside in a single analytic workspace.</td>
</tr>
<tr>
<td>job_class</td>
<td>The class this job is associated with.</td>
</tr>
</tbody>
</table>

**SCRIPT Parameter**

The **SCRIPT** parameter identifies the objects to include in the build, and specifies the type of processing to perform on each one. The parameter has this syntax:

```
[VALIDATE | NO COMMIT] objects [ USING ( commands ) ][, ...]
```

Where:

- **VALIDATE** checks all steps of the build and sends the planned steps to CUBE_BUILD_LOG without executing the steps. You can view all generated SQL in the OUTPUT column of the log table.

- **NO COMMIT** builds the objects in the current attach mode (or Read Only when the analytic workspace is not attached) but does not commit the changes. This option supports what-if analysis, since it enables you to change data values temporarily. See "SCRIPT Parameter: USING Clause: SET command".

- **objects** is the qualified name of one or more cubes or dimensions, separated by commas, in the form [aw_name.]object, such as UNITS_CUBE or GLOBAL.UNITS_CUBE.
SCRIPT Parameter: USING Clause

The **USING** clause specifies the processing options. It consists of one or more commands separated by commas.

**Note:**

A cube with a rewrite materialized view cannot have a **USING** clause, except for the **ANALYZE** command. It uses the default build options.

The commands can be any of the following.

- **AGGREGATE USING [MEASURE]**
  
  Generates aggregate values using the syntax described in "SCRIPT Parameter: USING Clause: AGGREGATE command".

- **ANALYZE**
  
  Runs `DBMS_AW_STATS.ANALYZE`, which generates and stores optimizer statistics for cubes and dimensions.

- **CLEAR [VALUES | LEAVES | AGGREGATES] [SERIAL | PARALLEL]**
  
  Prepares the cube for a data refresh. It can also be used on dimensions, but **CLEAR** removes all dimension keys, and thus deletes all data values for cubes that use the dimension.

  These optional arguments control the refresh method. If you omit the argument, then the behavior of **CLEAR** depends on the refresh method. The 'C' (complete) refresh method runs **CLEAR VALUES**, and all other refresh methods run **CLEAR LEAVES**.

  - **VALUES**: Clears all data in the cube. All facts must be reloaded and all aggregates must be recomputed. This option supports the **COMPLETE** refresh method. (Default for the C and F methods)
  
  - **LEAVES**: Clears the detail data and retains the aggregates. All facts must be reloaded, and the aggregates for any new or changed facts must be computed. This option supports the **FAST** refresh method. (Default for the ? method)
  
  - **AGGREGATES**: Retains the detail data and clears the aggregates. All aggregates must be recomputed.

  These optional arguments control the load method, and can be combined with any of the refresh options:

  - **PARALLEL**: Each partition is cleared separately. (Default)
  
  - **SERIAL**: All partitions are cleared together.

  If you omit the CLEAR command, `DBMS_CUBE` loads new and updated facts, but does not delete any old detail data. This is equivalent to a **LOAD NO SYNC** for dimensions.

- **COMPILE [SORT | NO SORT | SORT ONLY]**
  
  Creates the supporting structures for the dimension. (Dimensions only)
These options control the use of a sort order attribute:

- **SORT**: The user-defined sort order attribute populates the sort column in the embedded-total (ET) view. (Default)
- **NO SORT**: Any sort order attribute is ignored. This option is for very large dimensions where sorting could consume too many resources.
- **SORT ONLY**: The compile step only runs the sort.

**EXECUTE PLSQL string**

Executes a PL/SQL command or script in the database.

**EXECUTE OLAP DML string [PARALLEL | SERIAL]**

Executes an OLAP DML command or program in the analytic workspace. The options control execution of the command or program:

- **PARALLEL**: Execute the command or program once for each partition. This option can be used to provide a performance boost to complex DML operations, such as forecasts and models.
- **SERIAL**: Execute the command or program once for the entire cube. (Default)

**[INSERT | MERGE] INTO [ALL HIERARCHIES | HIERARCHIES (dimension.hierarchy)] VALUES (dim_key, parent, level_name)**

Adds a dimension member to one or more hierarchies. INSERT throws an error if the member already exists, while MERGE does not. See "Dimension Maintenance Example".

- **dimension.hierarchy**: The name of a hierarchy the new member belongs to. Enclose each part of the name in double quotes, for example, "PRODUCT"."PRIMARY".
- **dim_key**: The DIM_KEY value of the dimension member.
- **parent**: The parent of the dimension key.
- **level_name**: The level of the dimension key.

**UPDATE [ALL HIERARCHIES | HIERARCHIES (dimension.hierarchy)] SET PARENT = parent, LEVEL=level_name WHERE MEMBER = dim_key**

Alters the level or parent of an existing dimension member. See INSERT for a description of the options. Also see "Dimension Maintenance Example".

**DELETE FROM DIMENSION WHERE MEMBER=dim_key**

Deletes a dimension member. See "Dimension Maintenance Example".

**dim_key**: The DIM_KEY value of the dimension member to be deleted.

**SET dimension.attribute[qdr] = CAST('attribute_value' AS VARCHAR2))**

Sets the value of an attribute for a dimension member. See "Dimension Maintenance Example".

- **dimension.attribute**: The name of the attribute. Enclose each part of the name in double quotes, for example, "PRODUCT"."LONG_DESCRIPTION".
- **qdr**: The dimension member being given an attribute value in the form of a qualified data reference, such as "PRODUCT"='OPT MOUSE'.
- **attribute_value**: The value of the attribute, such as 'Optical Mouse'.

**FOR dimension_clause measure_clause BUILD (commands)**
Restricts the build to particular measures and dimension values, using the following arguments. See “FOR Clause Example”.

- dimension_clause:
  
  ```
  dimension ALL | NONE | WHERE condition | LEVELS (level [, level...])
  ```

  `dimension` is the name of a dimension of the cube.

  `ALL` sets the dimension status to all members before executing the list of commands.

  `NONE` loads values for no dimension members.

  `WHERE` loads values for those dimension members that match the condition.

  `LEVELS` loads values for dimension members in the named levels.

  `level` is a level of the named dimension.

- measure_clause:
  
  ```
  MEASURES (measure [, measure...])
  ```

  `measure` is the name of a measure in the cube.

- commands: Any of the other USING commands.

• LOAD [SYNCH | NO SYNCH | RETAIN] [PRUNE | PARALLEL | SERIAL] [WHERE condition]

Loads data into the dimension or cube.

- `WHERE` limits the load to those values in the mapped relational table that match the condition.

- `condition` is a valid predicate based on the columns of the mapped table. See the “Examples”.

These optional arguments apply only to dimensions:

- `SYNCH` matches the dimension keys to the relational data source. (Default)

- `NO SYNCH` loads new dimension keys but does not delete old keys.

  If the parent of a dimension key has changed in the relational data source, this option allows the load to change the parent/child relation in the analytic workspace.

- `RETAIN` loads new dimension keys but does not delete old keys.

  This option does not allow the parent of a dimension key to change. If the parent has changed, the load rejects the record. The rejection generates an error in the rejected records log, if the log is enabled.

These optional arguments apply only to cubes:

- `PRUNE`: Runs a full table scan on the fact table to determine which partitions to load. For example, if a cube is partitioned by month and the fact table has values only for the last two months, then jobs are only started to load the partitions for the last two months.

- `PARALLEL`: Each partition is loaded separately. (Default)

- `SERIAL`: All partitions are loaded in one SELECT statement.
• **MODEL model_name [PARALLEL | SERIAL]**
  
  Executes a model previously created for the cube. It accepts these arguments:
  
  – **PARALLEL**: The model runs separately on each partition.
  – **SERIAL**: The model runs on all cubes at the same time. (Default)

• **SET**

  Supports write-back to the cube using the syntax described in "SCRIPT Parameter: USING Clause: SET command". (Cubes only)

• **SOLVE [PARALLEL | SERIAL]**

  Aggregates the cube using the rules defined for the cube, including the aggregation operator and the precompute specifications. (Cubes only)

  It accepts these arguments:
  
  – **PARALLEL**: Each partition is solved separately. (Default)
  – **SERIAL**: All partitions are solved at the same time.

**SCRIPT Parameter: USING Clause: AGGREGATE command**

The `AGGREGATE` command in a script specifies the aggregation rules for one or more measures.

```plaintext
AGGREGATE has the following syntax:

{ AGGREGATE USING MEASURE
  WHEN measure1 THEN operator1
  WHEN measure2 THEN operator2...
  ELSE default_operator
  |
  [AGGREGATE USING] operator_clause |
  processing_options
  OVER { ALL | dimension | dimension HIERARCHIES (hierarchy)}

USING MEASURE Clause

This clause enables you to specify different aggregation operators for different measures in the cube.

Operator Clause

The `operator_clause` has this syntax:

operator(WEIGHTBY expression | SCALEBY expression)

WEIGHTBY multiplies each data value by an expression before aggregation.

SCALEBY adds the value of an expression to each data value before aggregation.
```
Table 40-3  Aggregation Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>WEIGHTBY</td>
<td>Adds data values, then divides the sum by the number of data values that were added together.</td>
</tr>
<tr>
<td>FIRST</td>
<td>WEIGHTBY</td>
<td>The first real data value.</td>
</tr>
<tr>
<td>HIER_AVG</td>
<td>WEIGHTBY</td>
<td>Adds data values, then divides the sum by the number of the children in the dimension hierarchy. Unlike AVERAGE, which counts only non-NA children, HAVERAGE counts all of the logical children of a parent, regardless of whether each child does or does not have a value.</td>
</tr>
<tr>
<td>HIER_FIRST</td>
<td>WEIGHTBY</td>
<td>The first data value in the hierarchy, even when that value is NA.</td>
</tr>
<tr>
<td>HIER_LAST</td>
<td>WEIGHTBY</td>
<td>The last data value in the hierarchy, even when that value is NA.</td>
</tr>
<tr>
<td>LAST</td>
<td>WEIGHTBY</td>
<td>The last real data value.</td>
</tr>
<tr>
<td>MAX</td>
<td>WEIGHTBY</td>
<td>The largest data value among the children of each parent.</td>
</tr>
<tr>
<td>MIN</td>
<td>WEIGHTBY</td>
<td>The smallest data value among the children of each parent.</td>
</tr>
<tr>
<td>NO AGGREGATION</td>
<td>No option</td>
<td>Do not aggregate the values of the dimension or dimensions. Leave all aggregated values as NA.</td>
</tr>
<tr>
<td>SUM</td>
<td>SCALEBY</td>
<td>Adds data values. (Default)</td>
</tr>
</tbody>
</table>

Processing Options

You can specify these processing options for aggregation:

• (ALLOW | DISALLOW) OVERFLOW
  Specifies whether to allow decimal overflow, which occurs when the result of a calculation is very large and can no longer be represented by the exponent portion of the numerical representation.
  – ALLOW: A calculation that generates overflow executes without error and produces null results. (Default)
  – DISALLOW: A calculation involving overflow stops executing and generates an error message.

• (ALLOW | DISALLOW) DIVISION BY ZERO
  Specifies whether to allow division by zero.
  – ALLOW: A calculation involving division by zero executes without error but returns a null value. (Default)
  – DISALLOW: A calculation involving division by zero stops executing and generates an error message.

• (CONSIDER | IGNORE) NULLS
  Specifies whether nulls are included in the calculations.
– **CONSIDER:** Nulls are included in the calculations. A calculation that includes a null value returns a null value.

– **IGNORE:** Only actual data values are used in calculations. Nulls are treated as if they do not exist. (Default)

• **MAINTAIN COUNT**

Stores an up-to-date count of the number of dimension members for use in calculating averages. Omit this option to count the members on the fly.

**SCRIPT Parameter: USING Clause: SET command**

The **SET** command in a script assigns values to one or more cells in a stored measure. It has this syntax:

```
SET target = expression
```

**Where:**

- **target** is a measure or a qualified data reference.
- **expression** returns values of the appropriate datatype for **target**.

**Qualified Data References**

Qualified data references (QDRs) limit a dimensional object to a single member in one or more dimensions for the duration of a query.

A QDR has the following syntax:

```
expression [{ dimension = member }[ , { dimension = member } ...] ]
```

**Where:**

- **expression** is a dimensional expression, typically the name of a measure.
- **dimension** is a primary dimension of expression.
- **member** is a value of dimension.

The outside square brackets shown in bold are literal syntax elements; they do not indicate an optional argument. The inside square brackets shown in regular text delimit an optional argument and are not syntax elements.

This example returns Sales values for calendar year 2007:

```
global.sales[global.time = 'CY2007'
]
```

The next example returns Sales values only for the United States in calendar year 2007:

```
sales[customer = 'US', time = 'CY2007'
]
```

See the Examples for qualified data references in SET commands.

**Usage Notes**

**Build Methods**

The **C, S, and ?** methods always succeed and can be used on any cube.
The F and P methods require that the cube have a materialized view that was created as a fast or a rewrite materialized view.

Parallelism

Partitioned cubes can be loaded and aggregated in parallel processes. For example, a cube with five partitions can use up to five processes. Dimensions are always loaded serially.

The number of parallel processes actually allocated by a build is controlled by the smallest of these factors:

- Number of cubes in the build and the number of partitions in each cube.
- Setting of the PARALLELISM argument of the BUILD procedure.
- Setting of the JOB_QUEUE_PROCESSES database initialization parameter.

Suppose UNITS_CUBE has 12 partitions, PARALLELISM is set to 10, and JOB_QUEUE_PROCESSES is set to 4. OLAP uses four processes, which appear as slave processes in the build log.

The SQL engine may allocate additional processes when the PARALLEL_DEGREE_POLICY database initialization parameter is set to AUTO or LIMITED. For example, if OLAP allocates four processes, the SQL engine might determine that two of those processes should be done by four processes instead, for a total of six processes.

Build Logs

OLAP generates three logs that provide diagnostic information about builds:

- Cube build log
- Rejected records log
- Cube dimension compile log

Analytic Workspace Manager creates these logs automatically as tables in the same schema as the analytic workspace. If you do not use Analytic Workspace Manager, you can create and manage the logs in PL/SQL using the DBMS_CUBE_LOG package.

You can also create the cube log file by running $ORACLE_HOME/olap/admin/utlolaplog.sql. This script creates three additional views:

- CUBE_BUILD_LATEST: Returns rows only from the last build.
- CUBE_BUILD_REPORT: Returns one row for each command with elapsed times.
- CUBE_BUILD_REPORT_LATEST: Returns a report like CUBE_BUILD_REPORT only from the last build.

This report shows a successfully completed build of the objects in the GLOBAL analytic workspace, which has four dimensions and two cubes.

SELECT command, status, build_object, build_object_type type
FROM cube_build_report_latest;

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>STATUS</th>
<th>BUILD_OBJECT</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD</td>
<td>COMPLETED</td>
<td>-----------------</td>
<td>BUILD</td>
</tr>
<tr>
<td>FREEZE</td>
<td>COMPLETED</td>
<td>BUILD</td>
<td></td>
</tr>
<tr>
<td>LOAD NO SYNCH</td>
<td>COMPLETED</td>
<td>CHANNEL</td>
<td>DIMENSION</td>
</tr>
</tbody>
</table>
31 rows selected.

Examples

This example uses the default parameters to build `UNITS_CUBE`.

EXECUTE DBMS_CUBE.BUILD('GLOBAL.UNITS_CUBE');

The next example builds `UNITS_CUBE` and explicitly builds two of its dimensions, `TIME` and `CHANNEL`. The dimensions use the complete (C) method, and the cube uses the fast solve (S) method.

BEGIN
  DBMS_CUBE.BUILD(
    script=>'GLOBAL."TIME", GLOBAL.CHANNEL, GLOBAL.UNITS_CUBE',
    method=>'CCS',
    parallelism=>2);
END;
/

The following example loads only the selection of data identified by the WHERE clause:

BEGIN
  DBMS_CUBE.BUILD(q'
GLOBAL."TIME",
GLOBAL.CHANNEL,
GLOBAL.CUSTOMER,
GLOBAL.PRODUCT,
GLOBAL.UNITS_CUBE USING (LOAD NO SYNCH
  WHERE UNITS_FACT.MONTH_ID LIKE '2006%' AND UNITS_FACT.SALES > 5000)');
FOR Clause Example

In this example, the Time dimension is partitioned by calendar year, and DBMS_CUBE builds only the partition identified by CY2006. The HIER_ANCESTOR is an analytic function in the OLAP expression syntax.

BEGIN
    dbms_cube.build(q'!
    UNITS_CUBE USING
    {
        FOR "TIME"
            WHERE HIER_ANCESTOR(WITHIN "TIME".CALENDAR LEVEL "TIME".CALENDAR_YEAR) = 'CY2006'
                BUILD (LOAD, SOLVE)
        },
        parallelism=>1);
    END;
/

The next example uses a FOR clause to limit the build to the SALES measure in 2006. All objects are built using the complete (C) method.

BEGIN
    DBMS_CUBE.BUILD(
        script => '
        GLOBAL."TIME",
        GLOBAL.CHANNEL,
        GLOBAL.CUSTOMER,
        GLOBAL.PRODUCT,
        GLOBAL.UNITS_CUBE USING
        {
            FOR MEASURES(GLOBAL.UNITS_CUBE.SALES)
                BUILD(LOAD NO SYNCH WHERE GLOBAL.UNITS_FACT.MONTH_ID LIKE "'2006'"
        },
        method => 'C',
        parallelism => 2);
    END;
/

Write-Back Examples

The following examples show various use of the SET command in a USING clause.

This example sets Sales Target to Sales increased by 5%:

DBMS_CUBE.BUILD('UNITS_CUBE USING(
    SET UNITS_CUBE.SALES_TARGET = UNITS_CUBE.SALES * 1.05, SOLVE)');

This example sets the price of the Deluxe Mouse in May 2007 to $29.99:

DBMS_CUBE.BUILD('PRICE_CUBE USING(
    SET PRICE_CUBE.UNIT_PRICE["TIME"='2007.05', "PRODUCT"='DLX MOUSE']
    = 29.99, SOLVE)');

The next example contains two SET commands, but does not reaggregate the cube:

DBMS_CUBE.BUILD('PRICE_CUBE USING(
    SET PRICE_CUBE.UNIT_PRICE["TIME"='2006.12', "PRODUCT"='DLX MOUSE'])
Dimension Maintenance Example

This script shows dimension maintenance. It adds a new dimension member named OPT MOUSE to all hierarchies, alters its position in the Primary hierarchy, assigns it a long description, then deletes it from the dimension.

BEGIN
dbms_output.put_line('Add optical mouse');
dbms_cube.build(q'!
"PRODUCT" using (MERGE INTO ALL HIERARCHIES
VALUES ('ITEM_OPT MOUSE', 'CLASS_SFT', "PRODUCT"."FAMILY"))
!');

dbms_output.put_line('Alter optical mouse');
dbms_cube.build(q'!
"PRODUCT" using (UPDATE HIERARCHIES ("PRODUCT"."PRIMARY"
SET PARENT = 'FAMILY_ACC', LEVEL = "PRODUCT"."ITEM"
WHERE MEMBER = 'ITEM_OPT MOUSE')
!');

dbms_output.put_line('Provide attributes to optical mouse');
dbms_cube.build(q'!
"PRODUCT" USING (SET "PRODUCT"."LONG_DESCRIPTION"["PRODUCT" = 'ITEM_OPT MOUSE']
 = CAST('Optical Mouse' AS VARCHAR2))
!');

dbms_output.put_line('Delete optical mouse');
dbms_cube.build(q'!
"PRODUCT" USING (DELETE FROM DIMENSION WHERE MEMBER='ITEM_OPT MOUSE')
!');
END;
/

OLAP DML Example

This example uses the OLAP DML to add comments to the cube build log:

BEGIN
DBMS_CUBE.BUILD(q'!
    global.units_cube USING {
        EXECUTE OLAP DML 'SHOW STATLEN(units_cube_prt_list)' PARALLEL,
        EXECUTE OLAP DML 'SHOW LIMIT(units_cube_prt_list KEEP ALL)' PARALLEL,
        EXECUTE OLAP DML 'SHOW STATLEN(time)' parallel,
        EXECUTE OLAP DML 'SHOW LIMIT(time KEEP time_levelrel ''CALENDAR_YEAR'')' parallel!
    };
    parallelism=>2,
    add_dimensions=>false);
END;
/

This query shows the comments in the cube build log:

SELECT partition, slave_number, TO_CHAR(output) output
FROM cube_build_log
WHERE command = 'OLAP DML'
    AND status = 'COMPLETED'
ORDER BY slave_number, time;

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>SLAVE_NUMBER</th>
<th>OUTPUT</th>
</tr>
</thead>
</table>
| P10:CY2007 | 1            | <OLAPDMLEExpression
  Expression="TO_CHAR(statlen(units_cubeprt_list))"
  Value="1"/> |
| P10:CY2007 | 1            | <OLAPDMLEExpression
  Expression="TO_CHAR(limit(units_cubeprt_list keep all))"
  Value="P10"/> |
| P10:CY2007 | 1            | <OLAPDMLEExpression
  Expression="TO_CHAR(statlen(time))"
  Value="17"/> |
| P10:CY2007 | 1            | <OLAPDMLEExpression
  Expression="TO_CHAR(limit(time keep time_levelrel &apos;CALENDAR_YEAR&apos;))"
  Value="CALENDAR_YEAR_CY2007"/> |
| P9:CY2006  | 2            | <OLAPDMLEExpression
  Expression="TO_CHAR(statlen(units_cubeprt_list))"
  Value="1"/> |
| P9:CY2006  | 2            | <OLAPDMLEExpression
  Expression="TO_CHAR(limit(units_cubeprt_list keep all))"
  Value="P9"/> |
| P9:CY2006  | 2            | <OLAPDMLEExpression
  Expression="TO_CHAR(statlen(time))"
  Value="17"/> |

40.4.2 CREATE_EXPORT_OPTIONS Procedure

This procedure creates an input XML document that describes processing options for the EXPORT_XML Procedure and the EXPORT_XML_TO_FILE Procedure.

Syntax

```sql
DBMS_CUBE.CREATE_EXPORT_OPTIONS (  
  out_options_xml IN/OUT CLOB,  
  target_version IN VARCHAR2 DEFAULT NULL,  
  suppress_owner IN BOOLEAN DEFAULT FALSE,  
  suppress_namespace IN BOOLEAN DEFAULT FALSE,  
  preserve_table_owners IN BOOLEAN DEFAULT FALSE,  
  metadata_changes IN CLOB DEFAULT NULL);
```
Parameters

Table 40-4   CREATE_EXPORT_OPTIONS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_options_xml</td>
<td>Contains the generated XML document, which can be passed into the options_xml parameter of the EXPORT_XML Procedure.</td>
</tr>
<tr>
<td>target_version</td>
<td>Specifies the version of Oracle Database in which the XML document generated by EXPORT_XML or EXPORT_XML_TO_FILE will be imported. You can specify two to five digits, such as 12.1 or 12.1.0.1.0. This parameter defaults to the current database version, and so can typically be omitted.</td>
</tr>
<tr>
<td>suppress_owner</td>
<td>Controls the use of the Owner attribute in XML elements and the owner qualifier in object names. Enter True to drop the owner from the XML, or enter False to retain it. Enter True if you plan to import the exported metadata into a different schema.</td>
</tr>
<tr>
<td>suppress_namespace</td>
<td>Controls the use of Namespace attributes in XML elements and the namespace qualifier in object names. Enter True to drop the namespace from the XML, or enter False to retain it (default). Enter True when upgrading to Oracle OLAP 12c metadata. Namespaces allow objects created in Oracle 10g to coexist with objects created in Oracle 12c. You cannot set or change namespaces.</td>
</tr>
<tr>
<td>preserve_table_owners</td>
<td>Controls the use of the owner in qualifying table names in the mapping elements, such as GLOBAL.UNITS_HISTORY_FACT instead of UNITS_HISTORY_FACT. Enter True to retain the table owner, or enter False to default to the current schema for table mappings. If you plan to import the exported metadata to a different schema, you must set this option to True to load data from tables and views in the original schema, unless the destination schema has its own copies of the tables and views.</td>
</tr>
<tr>
<td>metadata_changes</td>
<td>Contains an 12c XML description of an object that overwrites the exported object description. The XML document must contain all parent XML elements of the modified element with the attributes needed to uniquely identify them. Use the Name attribute if it exists. See the Examples.</td>
</tr>
</tbody>
</table>

Examples

The following example generates an XML document of export options:

```
DECLARE
    optionsClob  CLOB;
BEGIN
    dbms_lob.createtemporary(optionsClob, false, dbms_lob.CALL);
    dbms_cube.create_export_options(out_options_xml=>optionsClob,
                                     suppress_namespace=>TRUE);
    dbms_output.put_line(optionsClob);
END;
/
```

The `DBMS_OUTPUT.PUT_LINE` procedure displays this XML document (formatted for readability):

```
<?xml version="1.0"?>
<Export TargetVersion="12.1.0.1">
```
The next example generates an XML document with a metadata change to the mapping of the American long description attribute of the `CHANNEL` dimension.

```sql
DECLARE
    importClob clob;
    exportClob clob;
    overClob clob;
    exportOptClob clob;
    importOptClob clob;
BEGIN
    dbms_lob.createtemporary(overClob, TRUE);
    dbms_lob.open(overClob, DBMS_LOB.LOB_READWRITE);
    dbms_lob.writeappend(overClob,58, '<Metadata Version="1.3"
MinimumDatabaseVersion="12.1.0.1"/>
    dbms_lob.writeappend(overClob,34, '<StandardDimension Name="CHANNEL">
    dbms_lob.writeappend(overClob,75, '<Description Type="Description"
Language="AMERICAN" Value="Sales Channel"/>
    dbms_lob.writeappend(overClob,20, '</StandardDimension>
    dbms_lob.writeappend(overClob,11, '</Metadata>
    dbms_lob.close(overClob);

    -- Enable Oracle Database 12c Release 1 (12.1) clients to access 10g metadata
    dbms_cube.initialize_cube_upgrade;

    -- Create a CLOB containing the export options
    dbms_lob.createtemporary(exportOptClob, TRUE);
    dbms_cube.create_export_options(out_options_xml=>exportOptClob,
suppress_namespace=>TRUE, metadata_changes=>overClob);

    -- Create a CLOB containing the import options
    dbms_lob.createtemporary(importOptClob, TRUE);
    dbms_cube.create_import_options(out_options_xml=>importOptClob, rename_table => 'MY_OBJECT_MAP');

    -- Create CLOBs for the metadata
    dbms_lob.createtemporary(importClob, TRUE);
    dbms_lob.createtemporary(exportClob, TRUE);

    -- Export metadata from a 10g analytic workspace to a CLOB
    dbms_cube.export_xml(object_ids=>'GLOBAL_AW', options_xml=>exportOptClob,
out_xml=>exportClob);

    -- Import metadata from the CLOB
    dbms_cube.import_xml(in_xml => exportClob, options_xml=>importOptClob,
out_xml=>importClob);

    -- Load and aggregate the data
    dbms_cube.build(script=>'UNITS_CUBE, PRICE_AND_COST_CUBE');
END;
/
The following is the content of `exportClob` (formatting added for readability). The XML document changes the description of Channel to Sales Channel.

```xml
<Metadata Version="1.3" MinimumDatabaseVersion="12.1.0.1">
  <StandardDimension Name="CHANNEL">
    <Description Type="Description" Language="AMERICAN" Value="Sales Channel"/>
  </StandardDimension>
</Metadata>
```

Related Topics

- **EXPORT_XML Procedure**
  This procedure writes OLAP metadata to a CLOB.

- **EXPORT_XML_TO_FILE Procedure**
  This procedure exports OLAP metadata to a file. This file can be imported into a new or existing analytic workspace using the `IMPORT_XML` procedure. In this way, you can create a copy of the analytic workspace in another schema or database.

### 40.4.3 CREATE_IMPORT_OPTIONS Procedure

This procedure creates an input XML document that describes processing options for the `IMPORT_XML` Procedure.

**Syntax**

```sql
DBMS_CUBE.CREATE_IMPORT_OPTIONS (out_options_xml  IN/OUT  CLOB,
                                   validate_only    IN      BOOLEAN   DEFAULT FALSE,
                                   rename_table     IN      VARCHAR2  DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out_options_xml</code></td>
<td>Contains the generated XML document, which can be passed to the <code>options_xml</code> parameter of the <code>IMPORT_XML</code> Procedure.</td>
</tr>
<tr>
<td><code>validate_only</code></td>
<td>TRUE causes the <code>IMPORT_XML</code> procedure to validate the metadata described in the input file or the <code>in_xml</code> parameter, without committing the changes to the metadata.</td>
</tr>
<tr>
<td><code>rename_table</code></td>
<td>The name of a table identifying new names for the imported objects, in the form <code>[schema_name.]table_name</code>. The <code>IMPORT_XML</code> procedure creates objects using the names specified in the table instead of the ones specified in the XML document. See the Usage Notes for the format of the rename table.</td>
</tr>
</tbody>
</table>

**Usage Notes**

See the information about using a rename table in DBMS_CUBE - Upgrading 10g Analytic Workspaces.

**Examples**

This example specifies validation only and a rename table. For an example of the import CLOB being used in an import, see "IMPORT_XML Procedure".
DECLARE
importClob  clob;

BEGIN
    dbms_lob.createtemporary(importClob, TRUE);
    dbms_cube.create_import_options(out_options_xml => importClob, rename_table => 'MY_OBJECT_MAP', validate_only => TRUE);
    dbms_output.put_line(importClob);
END;
/

It generates the following XML document:

<?xml version="1.0"?>
<Import>
    <ImportOptions>
        <Option Name="ValidateOnly" Value="TRUE"/>
        <Option Name="RenameTable" Value="MY_OBJECT_MAP"/>
    </ImportOptions>
</Import>

Related Topics

• IMPORT_XML Procedure
  This procedure creates, modifies, or drops an analytic workspace by using an XML template.

40.4.4 CREATE_MVIEW Function

This function creates a cube materialized view from the definition of a relational materialized view.

Syntax

DBMS_CUBE.CREATE_MVIEW ( 
    mvowner        IN  VARCHAR2, 
    mvname         IN  VARCHAR2, 
    sam_parameters IN  CLOB  DEFAULT NULL) 
RETURN VARCHAR2;

Parameters

Table 40-6  CREATE_MVIEW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvowner</td>
<td>Owner of the relational materialized view.</td>
</tr>
<tr>
<td>mvname</td>
<td>Name of the relational materialized view. For restrictions, see &quot;Requirements for the Relational Materialized View&quot;. A single cube materialized view can replace many of the relational materialized views for a table. Choose the materialized view that has the lowest levels of the dimension hierarchies that you want represented in the cube materialized view.</td>
</tr>
<tr>
<td>sam_parameters</td>
<td>Parameters in the form 'parameter1=value1, parameter2=value2,...'. See &quot;SQL Aggregation Management Parameters&quot;.</td>
</tr>
</tbody>
</table>
SQL Aggregation Management Parameters

The `CREATE_MVIEW` and `DERIVE_FROM_MVIEW` functions use the SQL aggregation management (SAM) parameters described in Table 40-7. Some parameters support the development of cubes with advanced analytics. Other parameters support the development of Java applications. The default settings are appropriate for cube materialized views that are direct replacements for relational materialized views.

Table 40-7  SQL Aggregation Management Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| ADDTOPS       | Adds a top level and a level member to every dimension hierarchy in the cube. If the associated relational dimension has no hierarchy, then a dimension hierarchy is created.  
   TRUE: Creates levels named `ALL_dimension` with level members `All_dimension`. (Default)  
   FALSE: Creates only the hierarchies and levels identified by the relational dimensions. |
| ADDUNIQUEKEYPREFIX | Controls the creation of dimension keys.  
   TRUE: Creates cube dimension keys by concatenating the level name with the relational dimension key. This practice assures that the dimension keys are unique across all levels, such as `CITY_NEW_YORK` and `STATE_NEW_YORK`. (Default)  
   FALSE: Uses the relational dimension keys as cube dimension keys. |
| ATRMAPTYPE    | Specifies whether attributes are mapped by hierarchy levels, dimension levels, or both.  
   HIER_LEVEL: Maps attributes to the levels of a particular dimension hierarchy. (Default)  
   DIM_LEVEL: Maps attributes to the levels of the dimension regardless of hierarchy.  
   BOTH: Maps attributes to both dimension and hierarchy levels.  
   AUTO: Maps attributes to the levels of the dimension for a star schema and to the levels of a particular dimension hierarchy for a snowflake schema. |
| AWNAME        | Provides the name of the analytic workspace that owns the cube. Choose a simple database object name of 1 to 30 bytes. The default name is `fact_tablename_AWn`. |
| BUILD         | Specifies whether a data refresh will immediately follow creation of the cube materialized view.  
   IMMEDIATE: Refreshes immediately.  
   DEFERRED: Does not perform a data refresh. (Default)  
   **Note:** Only the `CREATE_MVIEW` function uses this parameter. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBEMVOPTION</td>
<td>Controls validation and creation of a cube materialized view. Regardless of this setting, the function creates an analytic workspace containing a cube and its related cube dimensions.</td>
</tr>
<tr>
<td>COMPLETE_REFRESH</td>
<td>Creates a complete refresh cube materialized view (full update).</td>
</tr>
<tr>
<td>FAST_REFRESH</td>
<td>Creates a fast refresh materialized view (incremental update).</td>
</tr>
<tr>
<td>REWRITE READY</td>
<td>Runs validation checks for a rewrite cube materialized view, but does not create it.</td>
</tr>
<tr>
<td>REWRITE</td>
<td>Creates a rewrite cube materialized view.</td>
</tr>
<tr>
<td>REWRITE WITH ATTRIBUTES</td>
<td>Creates a rewrite cube materialized view that includes columns with dimension attributes, resulting in faster query response times. (Default)</td>
</tr>
<tr>
<td>Note</td>
<td>The following settings do not create a cube materialized view. Use Analytic Workspace Manager to drop an analytic workspace that does not have a cube materialized view. You can use the DROP_MVIEW procedure to delete an analytic workspace only when it supports a cube materialized view.</td>
</tr>
<tr>
<td>NONE</td>
<td>Does not create a cube materialized view.</td>
</tr>
<tr>
<td>COMPLETE_REFRESH_READY</td>
<td>Runs validation checks for a complete refresh cube materialized view, but does not create it.</td>
</tr>
<tr>
<td>FAST_REFRESH READY</td>
<td>Runs validation checks for fast refresh, but does not create the cube materialized view.</td>
</tr>
<tr>
<td>CUBENAME</td>
<td>Provides the name of the cube derived from the relational materialized view. Choose simple database object name of 1 to 30 bytes. The default name is fact_tablename_Cn.</td>
</tr>
<tr>
<td>DIMJAVABINDVARS</td>
<td>Supports access by Java programs to the XML document.</td>
</tr>
<tr>
<td>TRUE</td>
<td>Generates an XML template that uses Java bind variable notation for the names of dimensions. No XML validation is performed. You cannot use the IMPORT_XML procedure to create a cube using this template.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Generates an XML template that does not support Java bind variables. (Default)</td>
</tr>
<tr>
<td>DISABLEQRW</td>
<td>Controls disabling of query rewrite on the source relational materialized view.</td>
</tr>
<tr>
<td>TRUE</td>
<td>Issues an ALTER MATERIALIZED VIEW mview_name DISABLE QUERY REWRITE command.</td>
</tr>
<tr>
<td>FALSE</td>
<td>No action.</td>
</tr>
<tr>
<td>Note</td>
<td>Only the CREATE_MVIEW function with BUILD=IMMEDIATE uses this parameter.</td>
</tr>
<tr>
<td>EXPORTXML</td>
<td>Exports the XML that defines the dimensional objects to a file, which you specify as dir/filename. Both the directory and the file name are case sensitive.</td>
</tr>
<tr>
<td>dir</td>
<td>Name of a database directory.</td>
</tr>
<tr>
<td>filename</td>
<td>The name of the file, typically given an XML filename extension.</td>
</tr>
</tbody>
</table>
Table 40-7  (Cont.) SQL Aggregation Management Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILTERPARTITIONANCESTORLEVELS</td>
<td>Controls the generation of aggregate values above the partitioning level of a partitioned cube.</td>
</tr>
<tr>
<td>TRUE:</td>
<td>Removes levels above the partitioning level from the cube. Requests for summary values above the partitioning level are solved by SQL.</td>
</tr>
<tr>
<td>FALSE:</td>
<td>All levels are retained in the cube. Requests for summary values are solved by OLAP. (Default)</td>
</tr>
<tr>
<td>LOGDEST</td>
<td>Directs and stores log messages. By default, the messages are not available.</td>
</tr>
<tr>
<td>SERVEROUT:</td>
<td>Sends messages to server output (typically the screen), which is suitable when working interactively such as in SQL*Plus or SQL Developer.</td>
</tr>
<tr>
<td>TRACEFILE:</td>
<td>Sends messages to the session trace file.</td>
</tr>
<tr>
<td>PARTITIONOPTION</td>
<td>Controls partitioning of the cube.</td>
</tr>
<tr>
<td>NONE:</td>
<td>Prevents partitioning.</td>
</tr>
<tr>
<td>DEFAULT:</td>
<td>Allows the Sparsity Advisor to determine whether partitioning is needed and how to partition the cube. (Default)</td>
</tr>
<tr>
<td>FORCE:</td>
<td>Partitions the cube even when the Sparsity Advisor recommends against it. The Sparsity Advisor identifies the best dimension, hierarchy, and level to use for partitioning.</td>
</tr>
<tr>
<td>dimension.hierarchy.level:</td>
<td>Partitions the cube using the specified dimension, hierarchy, and level.</td>
</tr>
<tr>
<td>POPULATELINEAGE</td>
<td>Controls the appearance of attributes in a cube materialized view.</td>
</tr>
<tr>
<td>TRUE:</td>
<td>Includes all dimension attributes in the cube materialized view. (Default)</td>
</tr>
<tr>
<td>FALSE:</td>
<td>Omits all dimension attributes from the cube materialized view.</td>
</tr>
<tr>
<td>PRECOMPUTE</td>
<td>Identifies a percentage of the data that is aggregated and stored. The remaining values are calculated as required by queries during the session.</td>
</tr>
<tr>
<td>precompute_percentage</td>
<td>Specify the top percentage for partitioned cubes. The default value is 35:0, which specifies precomputing 35% of the bottom partition and 0% of the top partition. If the cube is not partitioned, then the second number is ignored.</td>
</tr>
<tr>
<td>REMAPCOMPOSITEKEYS</td>
<td>Controls how multicolumn keys are rendered in the cube.</td>
</tr>
<tr>
<td>TRUE:</td>
<td>Creates a unique key attribute whose values are concatenated string expressions with an underscore between the column values. For example, the value BOSTON_MA_USA might be an expression produced from a multicolumn key composed of CITY, STATE, and COUNTRY columns. In addition, an attribute is created for each individual column to store the relational keys. (Default)</td>
</tr>
<tr>
<td>FALSE:</td>
<td>Creates a unique key attribute for each column.</td>
</tr>
</tbody>
</table>
Table 40-7  (Cont.) SQL Aggregation Management Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENDERINGMODE</td>
<td>Controls whether a loss in fidelity between the relational materialized view and the cube materialized view results in a warning message or an exception. See &quot;Requirements for the Relational Materialized View&quot;.</td>
</tr>
<tr>
<td>LOOSE</td>
<td>Losses are noted in the optional logs generated by the <code>CREATE_MVIEW Function</code> and the <code>DERIVE_FROM_MVIEW Function</code>. No exceptions are raised. (Default)</td>
</tr>
<tr>
<td>STRICT</td>
<td>Any loss in fidelity raises an exception so that no XML template is created.</td>
</tr>
<tr>
<td>SEEFILTERS</td>
<td>Controls whether conditions in the <code>WHERE</code> clause of the relational materialized view's defining query are retained or ignored.</td>
</tr>
<tr>
<td>TRUE</td>
<td>Renders valid conditions in the XML template. (Default)</td>
</tr>
<tr>
<td>FALSE</td>
<td>Ignores all conditions.</td>
</tr>
<tr>
<td>UNIQUENAMES</td>
<td>Controls whether top level dimensional objects have unique names. Cross namespace conflicts may occur because dimensional objects have different namespaces than relational objects.</td>
</tr>
<tr>
<td>TRUE</td>
<td>Modifies all relational names when they are rendered in the cube. (Default)</td>
</tr>
<tr>
<td>FALSE</td>
<td>Duplicates relational names in the cube unless a naming conflict is detected. In that case, a unique name is created.</td>
</tr>
<tr>
<td>UNKNOWNKEYASDIM</td>
<td>Controls handling of simple columns with no levels or hierarchies in the <code>GROUP BY</code> clause of the relational materialized view's defining query.</td>
</tr>
<tr>
<td>TRUE</td>
<td>Renders a simple column without a relational dimension as a cube dimension with no levels or hierarchies.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Raises an exception when no relational dimension is found for the column. (Default)</td>
</tr>
<tr>
<td>VALIDATEXML</td>
<td>Controls whether the generated XML document is validated.</td>
</tr>
<tr>
<td>TRUE</td>
<td>Validates the template using the <code>VALIDATE_XML procedure</code>. (Default)</td>
</tr>
<tr>
<td>FALSE</td>
<td>No validation is done.</td>
</tr>
</tbody>
</table>

Returns

The name of the cube materialized view created by the function.

Usage Notes

See "Using SQL Aggregation Management"

Examples

All examples for the SQL Aggregate Management subprograms use the sample Sales History schema, which is installed in Oracle Database with two relational materialized views: `CAL_MONTH_SALES_MV` and `FWEEK_PSCAT_SALES_MV`.

The following script creates a cube materialized view using `CAL_MONTH_SALES_MV` as the relational materialized view. It uses all default options.
SET serverout ON format wrapped

DECLARE
  salesaw  varchar2(30);
BEGIN
  salesaw := dbms_cube.create_mview('SH', 'CAL_MONTH_SALES_MV');
END;
/

The next example sets several parameters for creating a cube materialized view from F WEEK_PSCAT_SALES_MV. These parameters change the cube materialized view in the following ways:

- **ADDTOPS:** Adds a top level consisting of a single value to the hierarchies. All of the dimensions in Sales History have a top level already.
- **PRECOMPUTE:** Changes the percentage of materialized aggregates from 35:0 to 40:10.
- **EXPORTXML:** Creates a text file for the XML document.
- **BUILD:** Performs a data refresh.

DECLARE
  salescubemv  varchar2(30);
  sam_param     clob := 'ADDTOPS=FALSE,'  
                  'PRECOMPUTE=40:10,'  
                  'EXPORTXML=WORK_DIR/sales.xml,'  
                  'BUILD=IMMEDIATE';
BEGIN
  salescubemv := dbms_cube.create_mview('SH', 'FWEEK_PSCAT_SALES_MV',  
                                         sam_param);
END;
/

### 40.4.5 DERIVE_FROM_MVIEW Function

This function generates an XML template that defines a cube with materialized view capabilities, using the information derived from an existing relational materialized view.

**Syntax**

```
DBMS_CUBE.DERIVE_FROM_MVIEW (  
  mvowner       IN  VARCHAR2,  
  mvname        IN  VARCHAR2,  
  sam_parameters IN  CLOB DEFAULT NULL)  
RETURN CLOB;
```

**Parameters**

**Table 40-8  DERIVE_FROM_MVIEW Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvowner</td>
<td>Owner of the relational materialized view.</td>
</tr>
</tbody>
</table>
Table 40-8 (Cont.) DERIVE_FROM_MVIEW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvname</td>
<td>Name of the relational materialized view. For restrictions, see &quot;Requirements for the Relational Materialized View&quot;. A single cube materialized view can replace many of the relational materialized views for a table. Choose the materialized view that has the lowest levels of the dimension hierarchies that you want represented in the cube materialized view.</td>
</tr>
<tr>
<td>sam_parameters</td>
<td>Optional list of parameters in the form 'parameter1=value1, parameter2=value2,...'. See &quot;SQL Aggregation Management Parameters&quot;.</td>
</tr>
</tbody>
</table>

Returns

An XML template that defines an analytic workspace containing a cube enabled as a materialized view.

Usage Notes

To create a cube materialized view from an XML template, use the IMPORT_XML procedure. Then use the REFRESH_MVIEW procedure to refresh the cube materialized view with data.

See "Using SQL Aggregation Management".

Examples

The following example generates an XML template named sales_cube.xml from the CAL_MONTH_SALES_MV relational materialized view in the SH schema.

```sql
DECLARE
    salescubexml clob := null;
    sam_param clob := 'exportXML=WORK_DIR/sales_cube.xml';
BEGIN
    salescubexml := dbms_cube.derive_from_mview('SH', 'CAL_MONTH_SALES_MV', sam_param);
END;
/```

40.4.6 DROP_MVIEW Procedure

This procedure drops a cube materialized view and all associated objects from the database. These objects include the dimension materialized views, cubes, cube dimensions, levels, hierarchies, and the analytic workspace.

Syntax

```
DBMS_CUBE.DROP_MVIEW (
    mvowner IN VARCHAR2,
    mvname IN VARCHAR2,
    sam_parameters IN CLOB DEFAULT NULL);
```
Parameters

Table 40-9   DROP_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvowner</td>
<td>Owner of the cube materialized view</td>
</tr>
<tr>
<td>mvname</td>
<td>Name of the cube materialized view</td>
</tr>
<tr>
<td>sam_parameters</td>
<td>EXPORTXML: Exports the XML that drops the dimensional objects to a file, which you specify as dir/filename. Both the directory and the file name are case sensitive.</td>
</tr>
<tr>
<td>dir</td>
<td>Name of a database directory.</td>
</tr>
<tr>
<td>filename</td>
<td>The name of the file, typically given an XML filename extension.</td>
</tr>
</tbody>
</table>

Usage Notes

Use this procedure to drop a cube materialized view that you created using the CREATE_MVIEW and DERIVE_FROM_MVIEW functions. If you make modifications to the cubes or dimensions, then DROP_MVIEW may not be able to drop the cube materialized view.

Some of the CUBEMVOPTION parameters used by the CREATE_MVIEW and DERIVE_FROM_MVIEW functions do not create a materialized view. Use Analytic Workspace Manager to drop the analytic workspace, cubes, and cube dimensions.

If you use the EXPORTXML parameter, then you can use the XML document to drop the cube materialized view, after you re-create it. Use the IMPORT_XML procedure.

See "Using SQL Aggregation Management".

Examples

The current schema has four materialized views. CB$CAL_MONTH_SALES is a cube materialized view for the SALES table. CB$TIMES_DIM_D1_CAL_ROLLUP is a cube dimension materialized view for the TIMES_DIM dimension on the TIMES dimension table. The others are relational materialized views.

```
SELECT mview_name FROM user_mviews;

MVIEW_NAME
-------------------
CB$CAL_MONTH_SALES
CB$TIMES_DIM_D1_CAL_ROLLUP
CAL_MONTH_SALES_MV
FWEAK_FSCAT_SALES_MV
```

The following command drops both CB$CAL_MONTH_SALES and CB$TIMES_DIM_D1_CAL_ROLLUP.

```
EXECUTE dbms_cube.drop_mview('SH', 'CB$CAL_MONTH_SALES');
```

Dropped cube organized materialized view "SH"."CAL_MONTH_SALES" including container analytic workspace "SH"."CAL_MONTH_SALES_AW" at 20130213 16:31:40.056.
This query against the data dictionary confirms that the materialized views have been dropped.

```
SELECT mview_name FROM user_mviews;
```

<table>
<thead>
<tr>
<th>MVIEW_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAL_MONTH_SALES_MV</td>
</tr>
<tr>
<td>FWEEK_PSCAT_SALES_MV</td>
</tr>
</tbody>
</table>

### 40.4.7 EXPORT_XML Procedure

This procedure writes OLAP metadata to a CLOB.

**Syntax**

```
DBMS_CUBE.EXPORT_XML
   (object_ids    IN      VARCHAR2,
    out_xml       IN/OUT  CLOB;

DBMS_CUBE.EXPORT_XML
   (object_ids    IN      VARCHAR2,
    options_xml   IN      CLOB,
    out_xml       IN/OUT  CLOB;

DBMS_CUBE.EXPORT_XML
   (object_ids    IN      VARCHAR2,
    options_dirname IN     VARCHAR2,
    options_filename IN     VARCHAR2,
    out_xml       IN/OUT  CLOB;
```

**Parameters**

**Table 40-10  EXPORT_XML Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_ids</td>
<td>Any of these identifiers.</td>
</tr>
<tr>
<td></td>
<td>· The name of a schema, such as GLOBAL.</td>
</tr>
<tr>
<td></td>
<td>· The fully qualified name of an analytic workspace in the form owner.aw_name.AW, such as GLOBAL.GLOBAL.AW.</td>
</tr>
<tr>
<td></td>
<td>· Cube</td>
</tr>
<tr>
<td></td>
<td>· Dimension</td>
</tr>
<tr>
<td></td>
<td>· Named build process</td>
</tr>
<tr>
<td></td>
<td>· Measure folder</td>
</tr>
<tr>
<td></td>
<td>You can specify multiple objects by separating the names with commas.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> When exporting an individual object, be sure to export any</td>
</tr>
<tr>
<td></td>
<td>objects required to reconstruct it. For example, when exporting a cube,</td>
</tr>
<tr>
<td></td>
<td>you must also export the dimensions of the cube.</td>
</tr>
<tr>
<td>options_dirname</td>
<td>The case-sensitive name of a database directory that contains options_filename.</td>
</tr>
<tr>
<td>options_filename</td>
<td>A file containing an XML document of export options.</td>
</tr>
<tr>
<td>options_xml</td>
<td>A CLOB variable that contains an XML document of export options. Use the CREATE_EXPORT_OPTIONS Procedure to generate this document.</td>
</tr>
</tbody>
</table>
Table 40-10  (Cont.) EXPORT_XML Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_xml</td>
<td>A CLOB variable that will store the XML document of OLAP metadata for the objects listed in object_ids.</td>
</tr>
</tbody>
</table>

Export Options

The default settings for the export options are appropriate in many cases, so you can omit the options_xml parameter or the options_dirname and options_filename parameters. However, when upgrading Oracle OLAP 10g metadata to OLAP 12c, you must specify an XML document that changes the default settings. This example changes all of the parameters from False to True; set them appropriately for your schema.

```xml
<?xml version="1.0"?>
<Export>
  <ExportOptions>
    <Option Name="SuppressNamespace" Value="True"/>
    <Option Name="SuppressOwner" Value="True"/>
    <Option Name="PreserveTableOwners" Value="True"/>
  </ExportOptions>
</Export>
```

You can create this XML document manually or by using the CREATE_EXPORT_OPTIONS Procedure.

Usage Notes

See "DBMS_CUBE - Upgrading 10g Analytic Workspaces".

Example

For an example of using EXPORT_XML in an upgrade to the same schema, see "DBMS_CUBE - Upgrading 10g Analytic Workspaces".

The following PL/SQL script copies an OLAP 12c analytic workspace named GLOBAL12 from the GLOBAL_AW schema to the GLOBAL schema. No upgrade is performed.

To upgrade into a different schema, change the example as follows:

- Call the INITIALIZE_CUBE_UPGRADE procedure.
- Call the CREATE_EXPORT_OPTIONS procedure with the additional parameter setting SUPPRESS_NAMESPACE=>TRUE.

The PL/SQL client must be connected to the database as GLOBAL. The GLOBAL user must have SELECT permissions on GLOBAL_AW.AW$GLOBAL and on all relational data sources.

```sql
BEGIN
  -- Create a CLOB for the export options
  dbms_lob.createtemporary(optionsClob, TRUE);
  dbms_cube.create_export_options(out_options_xml=>optionsClob, suppress_owner=>TRUE, preserve_table_owners=>TRUE);

  -- Create a CLOB for the XML template
  dbms_lob.createtemporary(exportClob, TRUE);

  -- Export metadata from an analytic workspace to a CLOB
```
dbms_cube.export_xml(object_ids=>'GLOBAL_AW.GLOBAL12.AW',
options_xml=>optionsClob, out_xml=>exportClob);

-- Import metadata from the CLOB
dbms_cube.import_xml(in_xml=>exportClob);

-- Load and aggregate the data
dbms_cube.build(script=>'GLOBAL.UNITS_CUBE, GLOBAL.PRICE_AND_COST_CUBE');
END;
/

40.4.8 EXPORT_XML_TO_FILE Procedure

This procedure exports OLAP metadata to a file. This file can be imported into a new
or existing analytic workspace using the IMPORT_XML procedure. In this way, you can
create a copy of the analytic workspace in another schema or database.

This procedure can also be used as part of the process for upgrading OLAP standard
form metadata contained in an Oracle OLAP 10g analytic workspace to OLAP 12c
format.

Syntax

DBMS_CUBE.EXPORT_XML_TO_FILE
(object_ids            IN      VARCHAR2,
output_dirname        IN      VARCHAR2,
output_filename       IN      VARCHAR2;)

DBMS_CUBE.EXPORT_XML_TO_FILE
(object_ids            IN      VARCHAR2,
options_dirname       IN      VARCHAR2,
options_filename      IN      VARCHAR2,
output_dirname        IN      VARCHAR2,
output_filename       IN      VARCHAR2;)

Parameters

Table 40-11 EXPORT_XML_TO_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_ids</td>
<td>Any of these identifiers.</td>
</tr>
<tr>
<td>options_dirname</td>
<td>The case-sensitive name of a database directory that contains options_filename. See &quot;Export Options&quot;.</td>
</tr>
</tbody>
</table>

Note: When exporting an individual object, be sure to export any objects required to reconstruct it. For example, when you export a cube, you must also export the dimensions of the cube.
Table 40-11  (Cont.) EXPORT_XML_TO_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options_filename</td>
<td>The name of a file containing an XML document of export options. See &quot;Export Options&quot;.</td>
</tr>
<tr>
<td>output_dirname</td>
<td>The case-sensitive name of a database directory where output_filename is created.</td>
</tr>
<tr>
<td>output_filename</td>
<td>The name of the template file created by the procedure.</td>
</tr>
</tbody>
</table>

Export Options

The default settings for the export options are appropriate in most cases, and you can omit the options_dirname and options_filename parameters. However, when upgrading Oracle OLAP 10g metadata to OLAP 12c, you must specify an XML document that changes the default settings, like the following:

```xml
<?xml version="2.0"?>
<Export>
  <ExportOptions>
    <Option Name="SuppressNamespace" Value="True"/>
    <Option Name="SuppressOwner" Value="True"/>
    <Option Name="PreserveTableOwners" Value="True"/>
  </ExportOptions>
</Export>
```

You can create this XML document manually or by using the CREATE_EXPORT_OPTIONS Procedure.

Usage Notes

See "DBMS_CUBE - Upgrading 10g Analytic Workspaces".

Examples

The following example generates an XML file named global.xml in OLAP 12c format using the default export settings. The metadata is derived from all analytic workspaces and CWM metadata in the GLOBAL_AW schema. The output file is generated in the WORK_DIR database directory.

execute dbms_cube.export_xml_to_file('GLOBAL_AW', 'WORK_DIR', 'global.xml');

The next example also generates an XML file named global.xml in OLAP 12c format using the export options set in options.xml. The metadata is derived from the GLOBAL analytic workspace in the GLOBAL_AW schema. Both the options file and the output file are in the WORK_DIR database directory.

execute dbms_cube.export_xml_to_file('GLOBAL_AW.GLOBAL.AW', 'WORK_DIR', 'options.xml', 'WORK_DIR', 'global.xml');
40.4.9 IMPORT_XML Procedure

This procedure creates, modifies, or drops an analytic workspace by using an XML template.

Syntax

\[
\text{DBMS_CUBE.IMPORT_XML} \\
\text{(dirname} \
\text{IN} \ 	ext{VARCHAR2}, \\
\text{filename} \
\text{IN} \ 	ext{VARCHAR2} \
\text{);}
\]

\[
\text{DBMS_CUBE.IMPORT_XML} \\
\text{(dirname} \
\text{IN} \ 	ext{VARCHAR2}, \\
\text{filename} \
\text{IN} \ 	ext{VARCHAR2}, \\
\text{out_xml} \
\text{IN/OUT} \ 	ext{CLOB} \
\text{);}
\]

\[
\text{DBMS_CUBE.IMPORT_XML} \\
\text{(input_dirname} \
\text{IN} \ 	ext{VARCHAR2}, \\
\text{input_filename} \
\text{IN} \ 	ext{VARCHAR2}, \\
\text{options_dirname} \
\text{IN} \ 	ext{VARCHAR2}, \\
\text{options_filename} \
\text{IN} \ 	ext{VARCHAR2}, \\
\text{out_xml} \
\text{IN/OUT} \ 	ext{CLOB} \
\text{);}
\]

\[
\text{DBMS_CUBE.IMPORT_XML} \\
\text{(in_xml} \
\text{IN} \ 	ext{CLOB} \
\text{);}
\]

\[
\text{DBMS_CUBE.IMPORT_XML} \\
\text{(in_xml} \
\text{IN} \ 	ext{CLOB}, \\
\text{out_xml} \
\text{IN/OUT} \ 	ext{CLOB} \
\text{);}
\]

\[
\text{DBMS_CUBE.IMPORT_XML} \\
\text{(in_xml} \
\text{IN} \ 	ext{CLOB}, \\
\text{options_xml} \
\text{IN} \ 	ext{CLOB}, \\
\text{out_xml} \
\text{IN/OUT} \ 	ext{CLOB} \
\text{);}
\]

Parameters

Table 40-12 IMPORT_XML Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirname</td>
<td>The case-sensitive name of a database directory containing the XML document describing an analytic workspace.</td>
</tr>
<tr>
<td>filename</td>
<td>A file containing an XML document describing an analytic workspace.</td>
</tr>
<tr>
<td>in_xml</td>
<td>A CLOB containing an XML document describing an analytic workspace.</td>
</tr>
<tr>
<td>input_dirname</td>
<td>The case-sensitive name of a database directory containing the XML document describing an analytic workspace.</td>
</tr>
<tr>
<td>input_filename</td>
<td>A file containing an XML document describing an analytic workspace.</td>
</tr>
<tr>
<td>options_dirname</td>
<td>The case-sensitive name of a database directory containing a file of import options.</td>
</tr>
<tr>
<td>options_filename</td>
<td>A file of import options.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>options_xml</td>
<td>An XML document describing the import options. Use the <code>CREATE_IMPORT_OPTIONS Procedure</code> to generate this document.</td>
</tr>
<tr>
<td>out_xml</td>
<td>An XML document that either describes the analytic workspace or, for validation only, describes any errors. It may contain changes that <code>DBMS_CUBE</code> made to the imported XML, such as setting default values or making minor corrections to the XML.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The XML can define, modify, or drop an entire analytic workspace, or one or more cubes or dimensions. When defining just cubes or dimensions, you must do so within an existing analytic workspace.

You can also use `IMPORT_XML` to drop an analytic workspace by using the XML document generated by the `DROP_MVIEW` procedure with the `EXPORTXML` parameter.

See "DBMS_CUBE - Upgrading 10g Analytic Workspaces".

**Example**

This example loads an XML template from a file named `GLOBAL.XML` and located in a database directory named `XML_DIR`.

```sql
EXECUTE dbms_cube.import_xml('XML_DIR', 'GLOBAL.XML');
```

The next example exports an OLAP 10g template and uses `IMPORT_XML` to validate it before an upgrade to 12c.

```sql
DECLARE
    exportOptClob clob;
    importOptClob clob;
    importClob    clob;
    exportClob    clob;
BEGIN
    -- Create a CLOB for the export options
    dbms_lob.createtemporary(exportOptClob, TRUE);
    dbms_cube.create_export_options(out_options_xml=>exportOptClob,
    suppress_namespace=>TRUE, preserve_table_owners=>TRUE);
    -- Create a CLOB for the XML template
    dbms_lob.createtemporary(exportClob, TRUE);
    -- Create a CLOB for import options
    dbms_lob.createtemporary(importOptClob, TRUE);
    dbms_cube.create_import_options(out_options_xml=>importOptClob, validate_only=>TRUE);
    -- Create a CLOB for the change log
    dbms_lob.createtemporary(importClob, TRUE);
    -- Enable Oracle Database 12c Release 1 (12.1) clients to access 10g metadata
```
dbms_cube.initialize_cube_upgrade;

-- Export metadata from an analytic workspace to a CLOB
dbms_cube.export_xml(object_ids=>'GLOBAL_AW', options_xml=>exportOptClob,
out_xml=>exportClob);

/* Import metadata from the CLOB. No objects are committed to the database
because the validate_only parameter of CREATE_IMPORT_OPTIONS is set to
TRUE.
*/

dbms_cube.import_xml(in_xml=>exportClob, options_xml=>importOptClob,
out_xml=>importClob);

-- Output the metadata changes
dbms_output.put_line('This is the validation log:');
dbms_output.put_line(importClob);

END;
/

The contents of importClob show that the XML is valid. Otherwise, error messages
appear in the <RootCommitResult> element.

This is the validation log:
<?xml version="1.0" encoding="UTF-16"?>
<RootCommitResult>
</RootCommitResult>

For an example of IMPORT_XML within the context of an upgrade from 10g to 12c
metadata, see the Custom Upgrade section of "DBMS_CUBE - Upgrading 10g
Analytic Workspaces".

40.4.10 INITIALIZE_CUBE_UPGRADE Procedure

This procedure processes analytic workspaces created in Oracle OLAP 10g so they
can be used by Oracle OLAP 12c clients. It processes all analytic workspaces in the
current schema. Run this procedure once for each schema in which there are 10g
analytic workspaces.

Without this processing step, 12c clients cannot connect to a database containing a
10g analytic workspace with subobjects of a dimension or cube having the same
name. Additionally, some DBMS_CUBE procedures and functions, such as EXPORT_XML
and EXPORT_XML_TO_FILE, do not work on the 10g metadata.

After processing, OLAP 12c clients can connect and use the alternate names provided
by INITIALIZE_CUBE_UPGRADE for the conflicting subobjects. OLAP 10g clients
continue to use the original names.

INITIALIZE_CUBE_UPGRADE does not upgrade any OLAP 10g objects to OLAP 12c
format.

See "DBMS_CUBE - Upgrading 10g Analytic Workspaces".

Syntax

DBMS_CUBE.INITIALIZE_CUBE_UPGRADE;
Usage Notes

This procedure creates and populates a table named CUBE_UPGRADE_INFO. If it already exists, the table is truncated and repopulated.

While the 10g namespace allowed subobjects with the same name in the same dimension or cube, the 12c namespace does not. When INITIALIZE_CUBE_UPGRADE detects a name conflict among subobjects such as levels, hierarchies, and dimension attributes, it creates a row in CUBE_UPGRADE_INFO providing a new, unique name for each one. Rows may also be created for objects that do not require renaming; these rows are distinguished by a value of 0 or null in the CONFLICT column. Top-level objects, such as dimensions and cubes, are not listed.

You can edit the table using SQL INSERT and UPDATE if you want to customize the names of OLAP 10g objects on OLAP 12c clients.

The UPGRADE_AW, EXPORT_XML and EXPORT_XML_TO_FILE procedures use the names specified in the NEW_NAME column of the table to identify objects in CWM or OLAP standard form (AWXML) analytic workspaces, rather than the original names.

The following table describes the columns of CUBE_UPGRADE_INFO.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER</td>
<td>VARCHAR2</td>
<td>NOT NULL</td>
<td>Owner of the analytic workspace.</td>
</tr>
<tr>
<td>AW</td>
<td>VARCHAR2</td>
<td>NOT NULL</td>
<td>Name of the analytic workspace.</td>
</tr>
<tr>
<td>AWXML_ID</td>
<td>VARCHAR2</td>
<td>NOT NULL</td>
<td>Full logical name of the object requiring modification, in the form simple_name.[subtype_name].object_type. For example, TIME.DIMENSION and PRODUCT.COLOR.ATTRIBUTE.</td>
</tr>
<tr>
<td>NEW_NAME</td>
<td>VARCHAR2</td>
<td>NOT NULL</td>
<td>The name the object will have in Oracle 12c after the upgrade.</td>
</tr>
<tr>
<td>OBJECT_CLASS</td>
<td>VARCHAR2</td>
<td>--</td>
<td>DerivedMeasure for calculated measures, or empty for all other object types.</td>
</tr>
<tr>
<td>CONFLICT</td>
<td>NUMBER</td>
<td>--</td>
<td>Indicates the reason that the row was added to CUBE_UPGRADE_INFO:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: The object does not have a naming conflict but appears in the table for other reasons.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: Two objects have the same name and would create a conflict in the OLAP 12c namespace. The object type (such as level or hierarchy) will be added to the names.</td>
</tr>
</tbody>
</table>

Examples

The following command creates and populates the CUBE_UPGRADE_INFO table:

EXECUTE dbms_cube.initialize_cube_upgrade;

The table shows that the OLAP 10g analytic workspace has a hierarchy and a level named MARKET_SEGMENT, which will be renamed. The table also contains rows for calculated measures, but these objects do not require renaming: The value of CONFLICT is 0.

SELECT awxml_id, new_name, conflict FROM cube_upgrade_info;

<table>
<thead>
<tr>
<th>AWXML_ID</th>
<th>NEW_NAME</th>
<th>CONFLICT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
40.4.11 REFRESH_MVIEW Procedure

This procedure refreshes the data in a cube materialized view.

Syntax

```sql
DBMS_CUBE.REFRESH_MVIEW (
    mvowner IN VARCHAR2,
    mvname IN VARCHAR2,
    method IN VARCHAR2 DEFAULT NULL,
    refresh_after_errors IN BOOLEAN DEFAULT FALSE,
    parallelism IN BINARY_INTEGER DEFAULT 0,
    atomic_refresh IN BOOLEAN DEFAULT FALSE,
    scheduler_job IN VARCHAR2 DEFAULT NULL,
    sam_parameters IN CLOB DEFAULT NULL,
    nested IN BOOLEAN DEFAULT FALSE );
```

Parameters

Table 40-13  REFRESH_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvowner</td>
<td>Owner of the cube materialized view.</td>
</tr>
<tr>
<td>mvname</td>
<td>Name of the cube materialized view.</td>
</tr>
</tbody>
</table>
Table 40-13 (Cont.) REFRESH_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| method        | A full or a fast (partial) refresh. In a fast refresh, only changed rows are inserted in the cube and the affected areas of the cube are re-aggregated. You can specify a method for each cube in sequential order, or a single method to apply to all cubes. If you list more cubes than methods, then the last method applies to the additional cubes.  
- C: Complete refresh clears all dimension values before loading. (Default)  
- F: Fast refresh of a cube materialized view, which performs an incremental refresh and re-aggregation of only changed rows in the source table.  
- ?: Fast refresh if possible, and otherwise a complete refresh.  
- P: Recomputes rows in a cube materialized view that are affected by changed partitions in the detail tables.  
- S: Fast solve of a compressed cube. A fast solve reloads all the detail data and re-aggregates only the changed values.  
See the "Usage Notes" for the BUILD procedure for additional details. |
| refresh_after_errors | TRUE to roll back just the cube or dimension with errors, and then continue building the other objects.  
FALSE to roll back all objects in the build. |
| parallelism   | Number of parallel processes to allocate to this job.  
See the "Usage Notes" for the BUILD procedure for additional details. |
| atomic_refresh | TRUE prevents users from accessing intermediate results during a build. It freezes the current state of an analytic workspace at the beginning of the build to provide current sessions with consistent data. This option thaws the analytic workspace at the end of the build to give new sessions access to the refreshed data. If an error occurs during the build, then all objects are rolled back to the frozen state.  
FALSE enables users to access intermediate results during an build. |
| scheduler_job | Any text identifier for the job, which will appear in the log table. The string does not need to be unique. |
| sam_parameters | None. |
| nested        | TRUE performs nested refresh operations for the specified set of cube materialized views. Nested refresh operations refresh all the depending materialized views and the specified set of materialized views based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.  
All objects must reside in a single analytic workspace. |

**Usage Notes**

REFRESH_MVIEW changes mvname to the name of the cube, then passes the cube name and all parameters to the BUILD procedure. Thus, you can use the BUILD procedure to refresh a cube...
materialized view. See the "BUILD Procedure" for additional information about the parameters.

Examples

The following example uses the default settings to refresh a cube materialized view named $CB\_F\_WEEK\_PSCAT\_SALES$.

```
SET serverout ON format wrapped
EXECUTE dbms_cube.refresh_mview('SH', 'CB\_F\_WEEK\_PSCAT\_SALES');
```

The next example changes the refresh method to use fast refresh if possible, continue refreshing after an error, and use two parallel processes.

```
EXECUTE dbms_cube.refresh_mview('SH', 'CB\_F\_WEEK\_PSCAT\_SALES', '?', TRUE, 2);
```

After successfully refreshing the cube materialized view, REFRESH_MVIEW returns a message like the following:

```
Completed refresh of cube mview "SH\_CB\_F\_WEEK\_PSCAT\_SALES" at 20130212 15:04:46.370.
```

40.4.12 UPGRADE_AW Procedure

This procedure creates an Oracle OLAP 12c analytic workspace from a copy of the metadata contained in an OLAP 10g analytic workspace. The original OLAP 10g analytic workspace is not affected and can exist at the same time and in the same schema as the OLAP 12c analytic workspace.

**UPGRADE_AW** automatically runs **INITIALIZE_CUBE_UPGRADE** if the **CUBE_UPGRADE_INFO** table does not exist. If it does exist, then **UPGRADE_AW** does not overwrite it, thus preserving any changes you made to the table.

See "DBMS_CUBE - Upgrading 10g Analytic Workspaces".

Syntax

```
DBMS_CUBE.UPGRADE_AW
(sourceaw              IN  VARCHAR2,
 destaw                IN  VARCHAR2,
 upgoptions            IN  CLOB DEFAULT NULL);
```

Parameters

**Table 40-14**  **UPGRADE_AW** Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceaw</td>
<td>The name of a 10g analytic workspace.</td>
</tr>
<tr>
<td>destaw</td>
<td>A new name for the generated 12c analytic workspace. It cannot be the same as sourceaw.</td>
</tr>
</tbody>
</table>
Table 40-14  (Cont.) UPGRADE_AW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>upgoptions</td>
<td>One or more of these upgrade options, as a string in the form 'OPTION=VALUE'. Separate multiple options with commas.</td>
</tr>
<tr>
<td></td>
<td>• PRESERVE_TABLE_OWNERS:</td>
</tr>
<tr>
<td></td>
<td>YES preserves the original source table mappings. Use this option when creating an OLAP 12c analytic workspace in a different schema from the 10g analytic workspace, and you want the new objects mapped to tables in the original schema. (Default)</td>
</tr>
<tr>
<td></td>
<td>NO removes the schema owner from the source table mappings. Use this option when creating an OLAP 12c analytic workspace in a different schema from the 10g analytic workspace, and you want the new objects mapped to tables in the destination schema.</td>
</tr>
<tr>
<td></td>
<td>• RENAME_TABLE: The name of a table that specifies new names for objects as they are created in OLAP 12c format. These changes are in addition to those specified by the INITIALIZE_CUBE_UPGRADE procedure. See &quot;CREATE_IMPORT_OPTIONS Procedure&quot; for information about creating a rename table.</td>
</tr>
<tr>
<td></td>
<td>• TARGET_VERSION: The version of the upgrade, specified by a 2- to 5-part number, such as 11.2 or 11.2.0.2.0. If you enter an unsupported version number, then the closest version below it is used.</td>
</tr>
</tbody>
</table>

Examples

This example upgrades an OLAP 10g analytic workspace named GLOBAL10 to an OLAP 12c analytic workspace named GLOBAL12, using a rename table named MY_OBJECT_MAP:

```
BEGIN
  -- Upgrade the analytic workspace
  dbms_cube.upgrade_aw(sourceaw =>'GLOBAL10', destaw => 'GLOBAL12', upgoptions => 'RENAME_TABLE=MY_OBJECT_MAP');
  -- Load and aggregate the data
  dbms_cube.build(script=>'UNITS_CUBE, PRICE_AND_COST_CUBE');
  END;
  /
```

40.4.13 VALIDATE_XML Procedure

This procedure checks the XML to assure that it is valid without committing the results to the database. It does not create an analytic workspace.

Syntax

```
DBMS_CUBE.VALIDATE_XML
  (dirname IN VARCHAR2,
   filename IN VARCHAR2 );
```
DBMS_CUBE.VALIDATE_XML
(in_xml IN CLOB);

Parameters

Table 40-15 VALIDATE_XML Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirname</td>
<td>The case-sensitive name of a database directory.</td>
</tr>
<tr>
<td>filename</td>
<td>The name of a file containing an XML template.</td>
</tr>
<tr>
<td>IN_XML</td>
<td>The name of a CLOB containing an XML template.</td>
</tr>
</tbody>
</table>

Usage Notes

You should always load a template into the same version and release of Oracle Database as the one used to generate the template. The XML may not be valid if it was generated by a different release of the software.

Example

This example reports a problem in the schema:

```plsql
EXECUTE dbms_cube.validate_xml('UPGRADE_DIR', 'MYGLOBAL.XML');
BEGIN dbms_cube.validate_xml('UPGRADE_DIR', 'MYGLOBAL.XML'); END;
```

* ERROR at line 1:
  ORA-37162: OLAP error
  'GLOBAL.PRICE_CUBE.$AW_ORGANIZATION': XOQ-01950: The AWCubeOrganization for cube "GLOBAL.PRICE_CUBE" contains multiple BuildSpecifications with the same name.
  'GLOBAL.UNITS_CUBE.$AW_ORGANIZATION': XOQ-01950: The AWCubeOrganization for cube "GLOBAL.UNITS_CUBE" contains multiple BuildSpecifications with the same name.
  XOQ-01400: invalid metadata objects
  ORA-06512: at "SYS.DBMS_CUBE", line 411
  ORA-06512: at "SYS.DBMS_CUBE", line 441
  ORA-06512: at "SYS.DBMS_CUBE", line 501
  ORA-06512: at "SYS.DBMS_CUBE", line 520
  ORA-06512: at line 1

After the problems are corrected, the procedure reports no errors:

```plsql
EXECUTE dbms_cube.validate_xml('UPGRADE_DIR', 'MYGLOBAL.XML');
```

PL/SQL procedure successfully completed.

This example loads an XML template into a temporary CLOB, then validates it. The script is named GLOBAL.XML, and it is located in a database directory named XML_DIR.

```plsql
DEFINE xml_file = 'GLOBAL.XML';
SET ECHO ON;
SET SERVEROUT ON;

DECLARE
  xml_file BFILE := bfilename('XML_DIR', '&xml_file');
```
BEGIN
-- Setup the clob from a file
DBMS_LOB.CREATETEMPORARY(in_xml, TRUE);
DBMS_LOB.OPEN(in_xml, DBMS_LOB.LOB_READWRITE);
DBMS_LOB.OPEN(xml_file, DBMS_LOB.FILE_READONLY);
DBMS_LOB.LOADCLOBFROMFILE(in_xml, xml_file, DBMS_LOB.LOBMAXSIZE,
  dest_offset, src_offset, 0, lang_context, warning);

-- Validate the xml
DBMS_CUBE.VALIDATE_XML(in_xml);
END;
/*
41

DBMS_CUBE_ADVISE

DBMS_CUBE_ADVISE contains subprograms for evaluating cube materialized views to support log-based fast refresh and query rewrite.

This chapter contains the following topics:

• DBMS_CUBE_ADVISE Security Model
• Summary of DBMS_CUBE_ADVISE Subprograms

See Also:

Oracle OLAP User's Guide for information about cube materialized views

41.1 DBMS_CUBE_ADVISE Security Model

The MV_CUBE_ADVICE function requires the ADVISOR privilege.

41.2 Summary of DBMS_CUBE_ADVISE Subprograms

This table lists and describes the DBMS_CUBE_ADVISE subprograms.

Table 41-1 Summary of DBMS_CUBE_ADVISE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV_CUBE_ADVICE Function</td>
<td>Evaluates the metadata of a cube materialized view and generates recommendations for constraints, SQL dimension objects, and materialized view logs to support a broad range of query rewrite and fast refresh opportunities.</td>
</tr>
<tr>
<td>SET_CNS_EXCEPTION_LOG Procedure</td>
<td>Identifies the name of an exception log used in validated constraints generated by MV_CUBE_ADVICE.</td>
</tr>
<tr>
<td>TRACE Procedure</td>
<td>Displays or suppresses diagnostic messages for MV_CUBE_ADVICE.</td>
</tr>
</tbody>
</table>

41.2.1 MV_CUBE_ADVICE Function

This table function evaluates the metadata for a specified cube materialized view. It generates recommendations and returns them as a SQL result set. These SQL statements can be used to create constraints, SQL dimension objects, and materialized view logs that
allow the broadest range of query rewrite transformations and log-based fast refresh of
the cube materialized view.

**Syntax**

```
DBMS_CUBE_ADVISE.MV_CUBE_ADVICE (  
    owner        IN  VARCHAR2  DEFAULT USER,  
    mvname       IN  VARCHAR2,  
    reqtype      IN  VARCHAR2  DEFAULT '0',  
    validate     IN  NUMBER    DEFAULT 0)  
RETURN COAD_ADVICE_T  PIPELINED;
```

**Parameters**

**Table 41-2  **MV_CUBE_ADVICE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the cube materialized view</td>
</tr>
<tr>
<td>mvname</td>
<td>Name of the cube, such as <strong>UNITS_CUBE</strong>, or the cube materialized view,</td>
</tr>
<tr>
<td></td>
<td>such as <strong>CB$UNITS_CUBE</strong></td>
</tr>
<tr>
<td>reqtype</td>
<td>Type of advice to generate:</td>
</tr>
<tr>
<td></td>
<td>• 0: All applicable advice types</td>
</tr>
<tr>
<td></td>
<td>• 1: Column NOT NULL constraints</td>
</tr>
<tr>
<td></td>
<td>• 2: Primary key constraints</td>
</tr>
<tr>
<td></td>
<td>• 3: Foreign key constraints</td>
</tr>
<tr>
<td></td>
<td>• 4: Relational dimension objects</td>
</tr>
<tr>
<td></td>
<td>• 5: Cube materialized view logs with primary key</td>
</tr>
<tr>
<td>validate</td>
<td>Validation option:</td>
</tr>
<tr>
<td></td>
<td>• 0: Validate the constraints</td>
</tr>
<tr>
<td></td>
<td>• 1: Do not validate the constraints</td>
</tr>
</tbody>
</table>

**Returns**

A table of type **COAD_ADVICE_T**, consisting of a set of rows of type **COAD_ADVICE_REC**.
**Table 41-3** describes the columns.

**Table 41-3  **MV_CUBE_ADVICE Return Values

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER</td>
<td>VARCHAR2(30)</td>
<td>Owner of the dimensional object identified in <strong>APIOBJECT</strong>.</td>
</tr>
<tr>
<td>APIOBJECT</td>
<td>VARCHAR2(30)</td>
<td>Name of a cube enhanced with materialized view capabilities, such as <strong>UNITS_CUBE</strong>.</td>
</tr>
<tr>
<td>SQLOBJOWN</td>
<td>VARCHAR2(30)</td>
<td>Owner of the relational object identified in <strong>SQLOBJECT</strong>.</td>
</tr>
<tr>
<td>SQLOBJECT</td>
<td>VARCHAR2(65)</td>
<td>Name of the master table, such as <strong>UNITS_FACT</strong>, or the cube materialized view, such as <strong>CB$UNITS_CUBE</strong>.</td>
</tr>
</tbody>
</table>
Table 41-3 (Cont.) MV_CUBE_ADVICE Return Values

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVICETYPE</td>
<td>NUMBER(38,0)</td>
<td>Type of recommendation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 1: Create <strong>NOT NULL</strong> constraints on the foreign key columns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 2: Create primary key constraints on the master table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 3: Create primary key constraints on the master view</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 4: Create foreign key constraints on the master table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 5: Create foreign key constraints on the master view</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 6: Create relational dimensions on the master dimension tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 7: Create a materialized view log</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 8: Compile the materialized view</td>
</tr>
<tr>
<td>DISPOSITION</td>
<td>CLOB</td>
<td>Pre-existing conditions that conflict with the recommendations and should be resolved before SQLTEXT can be executed.</td>
</tr>
<tr>
<td>SQLTEXT</td>
<td>CLOB</td>
<td>SQL statement that implements the recommendation.</td>
</tr>
<tr>
<td>DROPTTEXT</td>
<td>CLOB</td>
<td>SQL statement that reverses SQLTEXT. Pre-existing conditions may prevent these statements from restoring the schema to its previous state.</td>
</tr>
</tbody>
</table>

Usage Notes

This function is available in Analytic Workspace Manager as the Materialized View Advisor, which will generate a SQL script with the recommendations.

You can query the returned rows the same as any other table, as shown in the example.

`MV_CUBE_ADVICE` generates unique object names each time it is called. You should execute the function once, capture the results, and work with those SQL statements.

Take care when dropping database objects. If a table already has a materialized view log, it will have the same name used in the SQL `DROP MATERIALIZED VIEW LOG` statement in the DROPTTEXT column. You should avoid inadvertently dropping materialized view logs, especially when they may be used for remote data replication.

Examples

The following query displays the SQL statements recommended by `MV_CUBE_ADVICE`. `UNITS_FACT` is the master table for `UNITS_CUBE`, and `MV_CUBE_ADVICE` generates an ALTER TABLE command to add primary key constraints.

It also generates an ALTER MATERIALIZED VIEW command to compile the CB$UNITS_CUBE cube materialized view.

```
SQL> SELECT apiobject, sqlobject, sqltext
FROM TABLE(dbms_cube_advise.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE'));
```
41.2.2 SET_CNS_EXCEPTION_LOG Procedure

This procedure identifies the name of an exception log used in validated constraints generated by MV_CUBE_ADVICE.

Syntax

```sql
DBMS_CUBE_ADVISE.SET_CNS_EXCEPTION_LOG (exceptlogtab IN VARCHAR2 DEFAULT user.EXCEPTIONS);
```

Parameters

**Table 41-4  SET_CNS_EXCEPTION_LOG Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exceptlogtab</td>
<td>The name of an existing exception log.</td>
</tr>
</tbody>
</table>

Usage Notes

To create an exception log, use the utlexcpt.sql or the utlexptl.sql script before executing SET_CNS_EXCEPTION_LOG.

The validate parameter of MV_CUBE_ADVICE must be set to 1.

Examples

The utlexcpt.sql script creates a table named EXCEPTIONS, and the SET_CNS_EXCEPTION_LOG procedure identifies it as the exception log for MV_CUBE_ADVICE. The ALTER TABLE statement now includes the clause VALIDATE EXCEPTIONS INTO "GLOBAL"."EXCEPTIONS".

SQL> @utlexcpt
Table created.

SQL> EXECUTE dbms_cube_advise.set_cns_exception_log;
PL/SQL procedure successfully completed.

SQL> SELECT apiobject, sqlobject, advicetype type, sqltext
FROM TABLE(
dbms_cube_advise.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE', '2', 1));

<table>
<thead>
<tr>
<th>APIOBJECT</th>
<th>SQLOBJECT</th>
<th>TYPE</th>
<th>SQLTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNITS_CUBE</td>
<td>UNITS_FACT</td>
<td>2</td>
<td>alter table &quot;GLOBAL&quot;.&quot;UNITS_FACT&quot; add constraint &quot;COAD_PK000219&quot; PRIMARY KEY (&quot;CHANNEL_ID&quot;, &quot;ITEM_ID&quot;, &quot;SHIP_TO_ID&quot;, &quot;MONTH_ID&quot;) norely enable validate exceptions into &quot;GLOBAL&quot;.&quot;EXCEPTIONS&quot;</td>
</tr>
</tbody>
</table>
41.2.3 TRACE Procedure

This procedure turns on and off diagnostic messages to server output for the MV_CUBE_ADVICE function.

Syntax

```
DBMS_CUBE_ADVISE.TRACE (  
    diaglevel       IN  BINARY_INTEGER DEFAULT 0);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diaglevel</td>
<td>0 to turn tracing off, or 1 to turn tracing on.</td>
</tr>
</tbody>
</table>

Examples

The following example directs the diagnostic messages to server output. The SQL*Plus SERVEROUTPUT setting displays the messages.

```
SQL> SET SERVEROUT ON FORMAT WRAPPED
SQL> EXECUTE dbms_cube_advise.trace(1);
```

PL/SQL procedure successfully completed.

```
SQL> SELECT sqlobject, sqltext, droptext
FROM TABLE(dbms_cube_advise.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE'))  
WHERE apiobject='UNITS_CUBE';
```

```
SQLOBJECT       SQLTEXT                                  DROPTEXT
--------------- ---------------------------------------- ----------------------------------------
UNITS_FACT      alter table "GLOBAL"."UNITS_FACT" add co alter table "GLOBAL"."UNITS_FACT" drop c  
nstraint "COAD_PK000222" PRIMARY KEY ("C onstraint "COAD_PK000222" cascade  
HANNEL_ID", "ITEM_ID", "SHIP_TO_ID", "MO  
NTH_ID") rely disable novalidate

CB$UNITS_CUBE   alter materialized view "GLOBAL"."CB$UNI alter materialized view "GLOBAL"."CB$UNI  
TS_CUBE" compile                         TS_CUBE" compile
```

20070706 07:25:27.462368000 DBMS_COAD_DIAG NOTE: Parameter moveOwner : GLOBAL
20070706 07:25:27.4623922000 DBMS_COAD_DIAG NOTE: Parameter moveName : CB$UNITS_CUBE
20070706 07:25:27.4623301100 DBMS_COAD_DIAG NOTE: Parameter cubeName : UNITS_CUBE
20070706 07:25:27.462463053000 DBMS_COAD_DIAG NOTE: Parameter cnsState : rely disable novalidate
20070706 07:25:27.462463094000 DBMS_COAD_DIAG NOTE: Parameter NNSstate : disable novalidate
20070706 07:25:27.462462038000 DBMS_COAD_DIAG NOTE: Begin NN:
20070706 07:25:27.462463053000 DBMS_COAD_DIAG NOTE: End     NN:
20070706 07:25:27.833620000 DBMS_COAD_DIAG NOTE: Begin PK:
20070706 07:25:27.833620000 DBMS_COAD_DIAG NOTE: End      PK:
SQL> EXECUTE dbms_cube_advise.trace(0);
DBMS_COAD_DIAG: Changing diagLevel from [1] to [0]

PL/SQL procedure successfully completed.
DBMS_CUBE_LOG contains subprograms for creating and managing logs for cubes and cube dimensions.

See Also:
OLAP Technology in the Oracle Database in *Oracle OLAP User's Guide* regarding use of the OLAP option to support business intelligence and analytical applications.

This chapter contains the following topics:

- Using DBMS_CUBE_LOG
- Summary of DBMS_CUBE_LOG Subprograms

42.1 Using DBMS_CUBE_LOG

DBMS_CUBE_LOG manages several logs that enable you to track the progress of long running processes, then use the results to profile performance characteristics.

They provide information to help you diagnose and remedy problems that may occur during development and maintenance of a cube, such as hierarchies that are improperly structured in the relational source tables, records that fail to load, or data refreshes that take too long to complete. They also help diagnose performance problems in querying cubes.

Analytic Workspace Manager creates the logs automatically using the default names and types. It also disables the logs when Analytic Workspace Manager is closed. To use the same logs outside of Analytic Workspace Manager, you must first enable them. Alternatively, you can create and manage different logs for use outside of Analytic Workspace Manager.

This section contains the following topics:

- Logging Types
- Logging Targets
- Verbosity Levels
- Security Model
- Creating Cube Logs
- Cube Build Log
- Cube Dimension Compile Log
- Cube Operations Log
- Cube Rejected Records Log
42.1.1 DBMS_CUBE_LOG — Logging Types

Several logs are available, each one dedicated to storing messages of a particular type. You may use all of them or only those that you find particularly valuable.

The logs and their contents are described later in this topic.

- Cube Build Log
- Cube Dimension Compile Log
- Cube Operations Log
- Cube Rejected Records Log

DBMS_CUBE_LOG provides functions that return the binary integer for each log type. You can produce more readable code by using these functions instead of integers for the argument values of other DBMS_CUBE_LOG procedures and functions. Refer to these descriptions:

- TYPE_BUILD Function
- TYPE_DIMENSION_COMPILE Function
- TYPE_OPERATIONS Function
- TYPE_REJECTED_RECORDS Function

42.1.2 DBMS_CUBE_LOG — Logging Targets

The TABLE_CREATE procedure creates database tables for storing the logs. Using the ENABLE procedure, you can create additional targets with changes in the destination or logging level. For example, you might target the Cube Operations log to both a table and a disk file.

These are the available targets:

- Disk file
- LOB
- Database table
- Trace file

See ENABLE Procedure for more information about creating multiple targets.

DBMS_CUBE_LOG provides functions that return the binary integer for each target type. You can produce more readable code by using these functions instead of integers for the argument values of other DBMS_CUBE_LOG procedures and functions. Refer to these descriptions:

- TARGET_FILE Function
- TARGET_LOB Function
- TARGET_TABLE Function
- TARGET_TRACE Function
42.1.3 DBMS_CUBE_LOG — Verbosity Levels

You can decide how much information is recorded in a log. You may want fewer details when leaving a job to run overnight than when you are monitoring the success of a new build. You can choose from these verbosity levels. Each level adds to the preceding level.

- **LOWEST**: Logs the status of each command used to build the cube dimensions and cubes, the use of slave processes, and summary records. This is the basic logging level.
- **LOW**: Logs messages from the OLAP engine, such as start and finish records for SQL Import, Aggregate, and Update.
- **MEDIUM**: Logs messages at the level used by Analytic Workspace Manager.
- **HIGH**: Logs messages that provide tuning information, such as composite lengths, partitioning details, object sizes, and aggregation work lists. This level is intended for use by Oracle Field Services.
- **HIGHEST**: Logs debugging messages and other information typically sent to a trace file. This level is intended for use by Oracle Support Services.

DBMS_CUBE_LOG provides functions that return the binary integer for each verbosity level. You can produce more readable code by using these functions instead of integers for the argument values of other DBMS_CUBE_LOG procedures and functions. Refer to these descriptions:

- LEVEL_LOWEST Function
- LEVEL_LOW Function
- LEVEL_MEDIUM Function
- LEVEL_HIGH Function
- LEVEL_HIGHEST Function

42.1.4 DBMS_CUBE_LOG Security Model

The TABLE_CREATE procedure requires the CREATE TABLE privilege.

42.1.5 DBMS_CUBE_LOG — Creating Cube Logs

To store logging information in a database table, you must create that table using the TABLE_CREATE procedure. Cube Build logs are always stored in tables. The ENABLE procedure creates the other target types for the other logs.

To create a Cube Build log:

- Execute the TABLE_CREATE procedure.

The following command creates a Cube Build log with the default name of CUBE_BUILD_LOG:

```sql
EXECUTE dbms_cube_log.table_create(dbms_cube_log.type_build);
```

To create a Cube Dimension Compile log, Cube Operations log, or Cube Rejected Records log with a database table target:

1. Execute the TABLE_CREATE procedure to create the table.
2. Execute the `ENABLE` procedure to begin logging.

These commands create and enable a Cube Operations table with the default name of `CUBE_OPERATIONS_LOG` and the default verbosity level:

```sql
EXECUTE dbms_cube_log.table_create(dbms_cube_log.type_operations);
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations);
```

To create a Cube Dimension Compile log, Cube Operations log, or Cube Rejected Records log with a trace file, disk file, or LOB target:

- Execute the `ENABLE` procedure.

This command enables the Cube Rejected Records log, sets verbosity to the lowest level, and directs the output to a disk file named `rejects.log` in the `WORK_DIR` database directory:

```sql
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_rejected_records, -
    dbms_cube_log.target_file, dbms_cube_log.level_lowest, -
    'WORK_DIR/rejects.log');
```

### 42.1.6 DBMS_CUBE_LOG — Cube Build Log

The Cube Build log provides information about what happened during a build. Use this log to determine whether the build produced the results you were expecting, and if not, why not.

The contents of the Cube Build log are refreshed continuously during a build. You can query the log at any time to evaluate the progress of the build and to estimate the time to completion.

The default name of the Cube Build log is `CUBE_BUILD_LOG`. The following table describes its contents.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_ID</td>
<td>NUMBER</td>
<td>--</td>
<td>A unique sequence number for the build. The same number is used for slave processes in a parallel build.</td>
</tr>
<tr>
<td>SLAVE_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>A counter for slave processes in a parallel build: 0 is the master process, 1 is the first slave, 2 is the second slave, and so forth.</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR2(10)</td>
<td>--</td>
<td>The current status of the command: STARTED, COMPLETED, FAILED, or WORKING.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>VARCHAR2(25)</td>
<td>--</td>
<td>The name of the command being executed, such as BUILD, LOAD, and SOLVE.</td>
</tr>
<tr>
<td>BUILD_OBJECT</td>
<td>VARCHAR2(500)</td>
<td>--</td>
<td>The name of the cube or cube dimension being processed.</td>
</tr>
</tbody>
</table>

**Note:**

To manage a Cube Build log, use only the `TABLE_CREATE` and `VERSION` procedures.
<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_OBJECT_TYPE</td>
<td>VARCHAR2(10)</td>
<td>--</td>
<td>The type of object: CUBE, DIMENSION, or BUILD.</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>CLOB</td>
<td>--</td>
<td>Information structured like an XML document about the command, or NULL when there is no additional information, such as for a STARTED row.</td>
</tr>
<tr>
<td>AW</td>
<td>VARCHAR2(30)</td>
<td>--</td>
<td>The name of the analytic workspace that contains the objects of the build.</td>
</tr>
<tr>
<td>OWNER</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>The owner of the analytic workspace and all the objects of the build.</td>
</tr>
<tr>
<td>PARTITION</td>
<td>VARCHAR2(10)</td>
<td>--</td>
<td>The name of the partition being processed, or NULL when the current operation does not correspond to a partition.</td>
</tr>
<tr>
<td>SCHEDULER_JOB</td>
<td>VARCHAR2(100)</td>
<td>--</td>
<td>A user-specified string to identify the build.</td>
</tr>
<tr>
<td>TIME</td>
<td>TIMESTAMP(6)</td>
<td>--</td>
<td>The time the row is added to the table.</td>
</tr>
<tr>
<td>BUILD_SCRIPT</td>
<td>CLOB</td>
<td>--</td>
<td>The cube build script. Populated only in rows where COMMAND is BUILD.</td>
</tr>
<tr>
<td>BUILD_TYPE</td>
<td>VARCHAR2(22)</td>
<td>--</td>
<td>The origin of the build: DBMS_CUBE, DBMS_MVIEW, JAVA, or SLAVE.</td>
</tr>
<tr>
<td>COMMAND_DEPTH</td>
<td>NUMBER(2)</td>
<td>--</td>
<td>The nesting depth of the command. For example, COMPILE HIERARCHIES is a component step of COMPILE, so if COMPILE has a depth of 1, then COMPILE HIERARCHIES has a depth of 2.</td>
</tr>
<tr>
<td>BUILD_SUB_OBJECT</td>
<td>VARCHAR2(30)</td>
<td>--</td>
<td>The name of a subobject being processed, such as a measure that does not inherit the aggregation rules of the cube.</td>
</tr>
<tr>
<td>REFRESH_METHOD</td>
<td>VARCHAR2(1)</td>
<td>--</td>
<td>The refresh method, such as C or F, that is associated with the current command. The refresh method is important only for the CLEAR step.</td>
</tr>
<tr>
<td>SEQ_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>Not currently used.</td>
</tr>
<tr>
<td>COMMAND_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>The sequence number of the command in the current process, which can be used to distinguish the same command on different objects. For example, a LOAD on PRODUCT and a LOAD on TIME.</td>
</tr>
<tr>
<td>IN_BRANCH</td>
<td>NUMBER(1)</td>
<td>--</td>
<td>Not currently used.</td>
</tr>
<tr>
<td>COMMAND_STATUS_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>Identifies the sequence number of all rows for a particular command. For example, a particular command might be represented by four rows: The first row has a status of STARTED and the last row has a status of COMPLETED. This column is used for sorting.</td>
</tr>
</tbody>
</table>
42.1.7 DBMS_CUBE_LOG — Cube Dimension Compile Log

When solving a cube, OLAP checks the dimension hierarchies to make sure they are valid. Errors that occur during this validation are written to the Cube Dimension Compile log.

The checks include:

- **Circularity:** Hierarchies are defined by parent-child relations among dimension members. Circularity occurs when a dimension member is specified as its own ancestor or descendant.
- **Hierarchy type:** Hierarchies can be level based or value based. You can define a cube so that only level-based hierarchies are valid, such as a cube materialized view.
- **Level options:** Level-based hierarchies can be regular, ragged, or skip level. You can define a dimension so that only regular hierarchies are valid, such as a Time dimension.

The default name of the Cube Dimension Compile log is `CUBE_DIMENSION_COMPILE`. The following table describes its contents.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>Current operation identifier</td>
</tr>
<tr>
<td>SEQ_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>Sequence number in the Cube Build log</td>
</tr>
<tr>
<td>ERROR#</td>
<td>NUMBER(8)</td>
<td>NOT NULL</td>
<td>Number of the error being reported</td>
</tr>
<tr>
<td>ERROR_MESSAGE</td>
<td>VARCHAR2(2000)</td>
<td>--</td>
<td>Error message associated with the error</td>
</tr>
<tr>
<td>DIMENSION</td>
<td>VARCHAR2(100)</td>
<td>--</td>
<td>Name of the dimension being compiled</td>
</tr>
<tr>
<td>DIMENSION_MEMBER</td>
<td>VARCHAR2(100)</td>
<td>--</td>
<td>Faulty dimension member</td>
</tr>
<tr>
<td>MEMBER_ANCESTOR</td>
<td>VARCHAR2(100)</td>
<td>--</td>
<td>Parent of DIMENSION_MEMBER</td>
</tr>
<tr>
<td>HIERARCHY1</td>
<td>VARCHAR2(100)</td>
<td>--</td>
<td>First hierarchy involved in the error</td>
</tr>
<tr>
<td>HIERARCHY2</td>
<td>VARCHAR2(100)</td>
<td>--</td>
<td>Second hierarchy involved in the error</td>
</tr>
<tr>
<td>ERROR_CONTEXT</td>
<td>CLOB</td>
<td>--</td>
<td>Additional information about the error</td>
</tr>
</tbody>
</table>

42.1.8 DBMS_CUBE_LOG — Cube Operations Log

The Cube Operations log contains messages and debugging information for all OLAP engine events. You can track current operations at a very detailed level. Using the SQL_ID column, you can join the Cube Operations log to dynamic performance views such as VS$SQL, VS$SESSION, and VS$SESSION_LONGOPS to see cube operations in the context of other database operations such as I/O Wait and CPU.

The default name of the Cube Operations log is `CUBEOPERATIONS_LOG`. The following table describes its contents.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Instance identifier</td>
</tr>
<tr>
<td>Column</td>
<td>Datatype</td>
<td>NULL</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>--------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>SID</td>
<td>NUMBER</td>
<td>NOT</td>
<td>Session identifier</td>
</tr>
<tr>
<td>SERIAL#</td>
<td>NUMBER</td>
<td>NOT</td>
<td>Session serial number</td>
</tr>
<tr>
<td>USER#</td>
<td>NUMBER</td>
<td>NOT</td>
<td>User identifier</td>
</tr>
<tr>
<td>SQL_ID</td>
<td>VARCHAR2(13)</td>
<td>--</td>
<td>Executing SQL statement identifier</td>
</tr>
<tr>
<td>JOB</td>
<td>NUMBER</td>
<td>--</td>
<td>Job identifier</td>
</tr>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>Current operation identifier</td>
</tr>
<tr>
<td>PARENT_ID</td>
<td>NUMBER</td>
<td>--</td>
<td>Parent operation identifier</td>
</tr>
<tr>
<td>SEQ_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>Sequence number in the Cube Build log</td>
</tr>
<tr>
<td>TIME</td>
<td>TIMESTAMP(6)</td>
<td>NOT</td>
<td>Time the record was added to the Cube Operations log</td>
</tr>
<tr>
<td>LOG_LEVEL</td>
<td>NUMBER(4)</td>
<td>NOT</td>
<td>Verbosity level of the record, as specified by the DBMS_CUBE_LOG.ENABLE procedure.</td>
</tr>
<tr>
<td>DEPTH</td>
<td>NUMBER(4)</td>
<td>--</td>
<td>Nesting depth of the record. For example, a level of 0 indicates that the operation and suboperation are not nested within other operations and suboperations.</td>
</tr>
<tr>
<td>OPERATION</td>
<td>VARCHAR2(15)</td>
<td>NOT</td>
<td>Current operation, such as AGGREGATE, ROWSOURCE, or SQLIMPORT.</td>
</tr>
<tr>
<td>SUBOPERATION</td>
<td>VARCHAR2(20)</td>
<td>--</td>
<td>Current suboperation, such as Loading or Import</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR2(10)</td>
<td>NOT</td>
<td>Current status of the operation, such as START, TRACE, COMPLETED, or Failed.</td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR2(20)</td>
<td>NOT</td>
<td>Name of the record, such as ROWS LOADED, AVE_ROW_LEN, and PAGEPOOLSIZE</td>
</tr>
<tr>
<td>VALUE</td>
<td>VARCHAR2(4000)</td>
<td>--</td>
<td>Value of NAME</td>
</tr>
<tr>
<td>DETAILS</td>
<td>CLOB</td>
<td>--</td>
<td>Additional information about NAME.</td>
</tr>
</tbody>
</table>

### 42.1.9 DBMS_CUBE_LOG — Cube Rejected Records Log

The Cube Rejected Records log contains a summary of the loader job and any records that were rejected because they did not meet the expected format.

A single row in the source table may have errors in more than one field. Each field generates an error in the log, resulting in multiple rows with the same rowid4 in the SOURCE_ROW column.

The default name of the Cube Rejected Records log is CUBE_REJECTED_RECORDS. The following table describes its contents.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>Current operation identifier</td>
</tr>
<tr>
<td>SEQ_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>Sequence number in the Cube Build log</td>
</tr>
</tbody>
</table>
42.2 Summary of DBMS_CUBE_LOG Subprograms

This table lists and describes the DBMS_CUBE_LOG subprograms.

Table 42-1  DBMS_CUBE_LOG Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_NAME Function</td>
<td>Returns the default table names of the various log types.</td>
</tr>
<tr>
<td>DISABLE Procedure</td>
<td>Turns logging off for the duration of a session.</td>
</tr>
<tr>
<td>ENABLE Procedure</td>
<td>Turns on logging for the duration of a session, redirects logging to additional output types, and changes the verbosity level in the logs.</td>
</tr>
<tr>
<td>FLUSH Procedure</td>
<td>Forces all buffered messages to be written to the logs.</td>
</tr>
<tr>
<td>GET_LOG Procedure</td>
<td>Returns the current settings for the level and location of a particular log.</td>
</tr>
<tr>
<td>GET_LOG_SPEC Function</td>
<td>Retrieves a description of all active logs.</td>
</tr>
<tr>
<td>GET_PARAMETER Function</td>
<td>Returns the current values of the options that control various aspects of logging.</td>
</tr>
<tr>
<td>LEVEL_HIGH Function</td>
<td>Returns the integer value of the high verbosity level.</td>
</tr>
<tr>
<td>LEVEL_HIGHEST Function</td>
<td>Returns the integer value of the highest verbosity level.</td>
</tr>
<tr>
<td>LEVEL_LOW Function</td>
<td>Returns the integer value of the low verbosity level.</td>
</tr>
<tr>
<td>LEVEL_LOWEST Function</td>
<td>Returns the integer value of the lowest verbosity level.</td>
</tr>
<tr>
<td>LEVEL_MEDIUM Function</td>
<td>Returns the integer value of the medium verbosity level.</td>
</tr>
<tr>
<td>SET_LOG_SPEC Procedure</td>
<td>Sets all logging to the values specified in the input string.</td>
</tr>
<tr>
<td>SET_PARAMETER Procedure</td>
<td>Sets options that control various aspects of logging.</td>
</tr>
<tr>
<td>TABLE_CREATE Procedure</td>
<td>Creates the table targets for the OLAP logs.</td>
</tr>
<tr>
<td>TARGET_FILE Function</td>
<td>Returns the integer value of a disk file target.</td>
</tr>
<tr>
<td>TARGET_LOB Function</td>
<td>Returns the integer value of a LOB target.</td>
</tr>
<tr>
<td>TARGET_TABLE Function</td>
<td>Returns the integer value of a database table target.</td>
</tr>
<tr>
<td>TARGET_TRACE Function</td>
<td>Returns the integer value of a trace file target.</td>
</tr>
<tr>
<td>TYPE_BUILD Function</td>
<td>Returns the integer value of the Cube Build log.</td>
</tr>
<tr>
<td>TYPE_DIMENSION_COMPILE Function</td>
<td>Returns the integer value of the Cube Dimension Compile log.</td>
</tr>
</tbody>
</table>
### Table 42-1  (Cont.) DBMS_CUBE_LOG Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_OPERATIONS Function</td>
<td>Returns the integer value of the Cube Operations log.</td>
</tr>
<tr>
<td>TYPE_REJECTED_RECORDS Function</td>
<td>Returns the integer value of the Cube Rejected Records log.</td>
</tr>
<tr>
<td>VERSION Function</td>
<td>Returns the version number of a specific log table or the current version number of a specific log type.</td>
</tr>
</tbody>
</table>

#### 42.2.1 DEFAULT_NAME Function

This function returns the default table names of the various log types.

**Syntax**

```sql
DBMS_CUBE_LOG.DEFAULT_NAME (LOG_TYPE IN BINARY_INTEGER DEFAULT TYPE_OPERATIONS) RETURN VARCHAR2;
```

**Parameters**

**Table 42-2  DEFAULT_NAME Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_type</td>
<td>One of the following log types:</td>
</tr>
<tr>
<td></td>
<td>• 1: TYPE_OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>• 2: TYPE_REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>• 3: TYPE_DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td>• 4: TYPE_BUILD</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Types&quot;.</td>
</tr>
</tbody>
</table>

**Returns**

The default table name of the specified log type.

**Examples**

This example returns the default name of the Cube Operations log:

```sql
SELECT dbms_cube_log.default_name FROM dual;
```

```
DEFAULT_NAME
--------------------
CUBE_OPERATIONS_LOG
```

The next example returns the default name of the Cube Rejected Records log:

```sql
SELECT dbms_cube_log.default_name(dbms_cube_log.type_rejected_records) - "Default Name" FROM dual;
```

```
Default Name
---------------------
CUBE_REJECTED_RECORDS
```
42.2.2 DISABLE Procedure

This procedure turns logging off for the duration of a session, unless logging is explicitly turned on again with the ENABLE procedure.

Syntax

```sql
DBMS_CUBE_LOG.DISABLE (  
    LOG_TYPE       IN   BINARY_INTEGER  DEFAULT,  
    LOG_TARGET     IN   BINARY_INTEGER  DEFAULT);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_type</td>
<td>One of the following log types:</td>
</tr>
<tr>
<td></td>
<td>1: TYPE_OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>2: TYPE_REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>3: TYPE_DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You cannot disable the Cube Build log with this procedure.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Types&quot;.</td>
</tr>
<tr>
<td>log_target</td>
<td>One of the following destinations for the logging records. The logs are sent to a table unless you previously specified a different target using the ENABLE procedure.</td>
</tr>
<tr>
<td></td>
<td>1: TARGET_TABLE</td>
</tr>
<tr>
<td></td>
<td>2: TARGET_TRACE</td>
</tr>
<tr>
<td></td>
<td>3: TARGET_FILE</td>
</tr>
<tr>
<td></td>
<td>4: TARGET_LOB</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Targets&quot;.</td>
</tr>
</tbody>
</table>

Example

This command disables the dimension compilation error log table:

```sql
EXECUTE dbms_cube_log.disable(dbms_cube_log.type_dimension_compile);  
```

42.2.3 ENABLE Procedure

This procedure turns on logging for the duration of a session or until it is turned off using the DISABLE procedure.

The ENABLE procedure also allows you to direct logging to additional output types and to change the amount of detail in the logs. You can enable a log type to each of the log targets. For example, you can enable the Cube Operations log to the trace file, a table, and a file at different verbosity levels, but you cannot enable the Cube Operations log to two files at the same time.

This procedure also checks the format of the logs and updates them if necessary.
Syntax

```sql
DBMS_CUBE_LOG.ENABLE (  
    LOG_TYPE       IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_TARGET     IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_LEVEL      IN     BINARY_INTEGER  DEFAULT NULL);  
```

```sql
DBMS_CUBE_LOG.ENABLE (  
    LOG_TYPE       IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_TARGET     IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_LEVEL      IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_LOCATION   IN     VARCHAR2        DEFAULT NULL);  
```

```sql
DBMS_CUBE_LOG.ENABLE (  
    LOG_TYPE       IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_TARGET     IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_LEVEL      IN     BINARY_INTEGER  DEFAULT NULL,  
    LOG_LOCATION   IN/OUT CLOB );  
```

Parameters

Table 42-4  ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_type</td>
<td>One of the following log types:</td>
</tr>
<tr>
<td></td>
<td>• 1: TYPE_OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>• 2: TYPE_REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>• 3: TYPE_DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You cannot disable the Cube Build log with this procedure.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Types&quot;</td>
</tr>
<tr>
<td>log_target</td>
<td>One of the following destinations for the logging records. The logs are</td>
</tr>
<tr>
<td></td>
<td>sent to a table unless you previously specified a different target using</td>
</tr>
<tr>
<td></td>
<td>the ENABLE procedure.</td>
</tr>
<tr>
<td></td>
<td>• 1: TARGET_TABLE</td>
</tr>
<tr>
<td></td>
<td>• 2: TARGET_TRACE</td>
</tr>
<tr>
<td></td>
<td>• 3: TARGET_FILE</td>
</tr>
<tr>
<td></td>
<td>• 4: TARGET_LOB</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Targets&quot;</td>
</tr>
<tr>
<td>log_level</td>
<td>One of the following log verbosity levels. Each level adds new types of</td>
</tr>
<tr>
<td></td>
<td>messages to the previous level.</td>
</tr>
<tr>
<td></td>
<td>• 1: LEVEL_LOWEST</td>
</tr>
<tr>
<td></td>
<td>• 2: LEVEL_LOW</td>
</tr>
<tr>
<td></td>
<td>• 3: LEVEL_MEDIUM</td>
</tr>
<tr>
<td></td>
<td>• 4: LEVEL_HIGH</td>
</tr>
<tr>
<td></td>
<td>• 5: LEVEL_HIGHEST</td>
</tr>
<tr>
<td></td>
<td>See &quot;Verbosity Levels&quot;</td>
</tr>
<tr>
<td>log_location</td>
<td>The full identity of the log, such as owner.table_name when</td>
</tr>
<tr>
<td></td>
<td>log_target is a table.</td>
</tr>
</tbody>
</table>

Examples

The following command enables all cube logs:
EXECUTE dbms_cube_log.enable;

The following PL/SQL procedure sets the log level to `LEVEL_LOWEST`:

```
BEGIN
    dbms_cube_log.disable(dbms_cube_log.type_rejected_records);
    dbms_cube_log.enable(dbms_cube_log.type_rejected_records,
                          dbms_cube_log.target_table, dbms_cube_log.level_lowest);
END;
/
```

### 42.2.4 FLUSH Procedure

This procedure forces all buffered messages to be written to the logs.

The buffers are flushed automatically throughout a session, but manually flushing them before viewing the logs assures that you can view all of the messages.

**Syntax**

```
DBMS_CUBE_LOG.FLUSH ( );
```

**Example**

The following example flushes the buffers for all of the logs:

EXECUTE dbms_cube_log.flush;

### 42.2.5 GET_LOG Procedure

This procedure returns the current settings for the level and location of a particular log.

**Syntax**

```
DBMS_CUBE_LOG.GET_LOG (  
                          LOG_TYPE       IN   BINARY_INTEGER  DEFAULT TYPE_OPERATIONS,  
                          LOG_TARGET     IN   BINARY_INTEGER  DEFAULT TARGET_TABLE,  
                          LOG_LEVEL      OUT  BINARY_INTEGER,  
                          LOG_LOCATION   OUT  VARCHAR2 );
```

**Parameters**

**Table 42-5**  GET_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_type</td>
<td>One of the following log types:</td>
</tr>
<tr>
<td></td>
<td>• 1:TYPE_OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>• 2:TYPE_REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>• 3:TYPE_DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Types&quot;.</td>
</tr>
</tbody>
</table>
Table 42-5  (Cont.) GET_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_target</td>
<td>One of the following destinations for the logging records. The logs are sent to a table unless you previously specified a different target using the ENABLE procedure.</td>
</tr>
<tr>
<td></td>
<td>- 1: TARGET_TABLE</td>
</tr>
<tr>
<td></td>
<td>- 2: TARGET_TRACE</td>
</tr>
<tr>
<td></td>
<td>- 3: TARGET_FILE</td>
</tr>
<tr>
<td></td>
<td>- 4: TARGET_LOB</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Targets&quot;</td>
</tr>
<tr>
<td>log_level</td>
<td>One of the following log verbosity levels. Each level adds new types of messages to the previous level.</td>
</tr>
<tr>
<td></td>
<td>- 1: LEVEL_LOWEST</td>
</tr>
<tr>
<td></td>
<td>- 2: LEVEL_LOW</td>
</tr>
<tr>
<td></td>
<td>- 3: LEVEL_MEDIUM</td>
</tr>
<tr>
<td></td>
<td>- 4: LEVEL_HIGH</td>
</tr>
<tr>
<td></td>
<td>- 5: LEVEL_HIGHEST</td>
</tr>
<tr>
<td></td>
<td>See &quot;Verbosity Levels&quot;</td>
</tr>
<tr>
<td>log_location</td>
<td>The full identity of the log, such as owner.table_name when log_target is a table.</td>
</tr>
</tbody>
</table>

Usage Notes

If log_type is not active, then log_level and log_location are null. Use DBMS_CUBE_LOG.ENABLE to activate a log.

Examples

This PL/SQL procedure provides information about the Cube Rejected Records log:

```sql
SET serverout ON format wrapped

DECLARE
    myloglevel  binary_integer;
    mylogtarget varchar2(128);
BEGIN
    dbms_cube_log.get_log(dbms_cube_log.type_rejected_records, dbms_cube_log.target_table, myloglevel, mylogtarget);
    dbms_output.put_line('Log Level: ' || myloglevel);
    dbms_output.put_line('Log Target: ' || mylogtarget);
END;
/
```

The procedure generates results like the following:

Log Level: 5

Log Target: GLOBAL.CUBE_REJECTED_RECORDS
42.2.6 GET_LOG_SPEC Function

This function retrieves a description of all active Cube Operations logs, Cube Rejected Records logs, and Cube Dimension Compile logs.

Syntax

```sql
DBMS_CUBE_LOG.GET_LOG_SPEC ( )
    RETURN VARCHAR2;
```

Returns

The type and target of all active logs.

Usage Notes

You can use the output from this function as the input to SET_LOG_SPEC.

Examples

The following example shows that the Cube Operations log, Cube Rejected Records log, and Cube Dimension Compile log are active. The Cube Operations log is stored in the session trace file and the other logs are stored in tables.

```sql
SELECT dbms_cube_log.get_log_spec FROM dual;
```

```
GET_LOG_SPEC
--------------------------------------------------------------------------------
OPERATIONS(TABLE, TRACE) REJECTED_RECORDS(TABLE[DEBUG])
```

42.2.7 GET_PARAMETER Function

This function returns the current values of the options that control various aspects of logging. To set these options, use the SET_PARAMETER function.

Syntax

```sql
DBMS_CUBE_LOG.GET_PARAMETER ( 
    LOG_TYPE       IN   BINARY_INTEGER, 
    LOG_PARAMETER  IN   BINARY_INTEGER )
    RETURN BINARY_INTEGER;
```

Parameters

Table 42-6   GET_PARAMETER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_type</td>
<td>One of the following log types:</td>
</tr>
<tr>
<td></td>
<td>• 1: TYPE_OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>• 2: TYPE_REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>• 3: TYPE_DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td>See “Logging Types”.</td>
</tr>
</tbody>
</table>


Table 42-6  (Cont.) GET_PARAMETER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_parameter</td>
<td>One of the following options:</td>
</tr>
<tr>
<td></td>
<td>• 1: MAX_ERRORS</td>
</tr>
<tr>
<td></td>
<td>• 2: FLUSH_INTERVAL</td>
</tr>
<tr>
<td></td>
<td>• 3: LOG_FULL_RECORD</td>
</tr>
<tr>
<td></td>
<td>• 4: LOG_EVERY_N</td>
</tr>
<tr>
<td></td>
<td>• 5: ALLOW_ERRORS</td>
</tr>
</tbody>
</table>

Returns
The value of the specified log_parameter.

Examples
This example shows the current maximum number of errors in the Cube Rejected Records log before logging stops. This parameter was previously set with the SET_PARAMETER procedure.

```
SELECT dbms_cube_log.get_parameter(dbms_cube_log.type_rejected_records, 1) - "Maximum Records" FROM dual;
```

Maximum Records
---------------
100

42.2.8 LEVEL_HIGH Function

This function returns the integer value of the high verbosity level.

Syntax

```
DBMS_CUBE_LOG.LEVEL_HIGH ()
RETURN BINARY_INTEGER;
```

Returns
4

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_LEVEL parameter in DBMS_CUBE_LOG subprograms. See "Verbosity Levels".

Example

This command sets the verbosity level of the cube operations table to high:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -
                           dbms_cube_log.target_table, dbms_cube_log.level_high);
```
42.2.9 LEVEL_HIGHEST Function

This function returns the integer value of the highest verbosity level.

Syntax

```sql
DBMS_CUBE_LOG.LEVEL_HIGHEST ()
    RETURN BINARY_INTEGER;
```

Returns

5

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_LEVEL parameter in DBMS_CUBE_LOG subprograms. See "Verbosity Levels".

Example

This command sets the verbosity level of the cube operations table to highest:

```sql
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -
    dbms_cube_log.target_table, dbms_cube_log.level_highest);
```

42.2.10 LEVEL_LOW Function

This function returns the integer value of the low verbosity level.

Syntax

```sql
DBMS_CUBE_LOG.LEVEL_LOW ()
    RETURN BINARY_INTEGER;
```

Returns

2

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_LEVEL parameter in DBMS_CUBE_LOG subprograms. See "Verbosity Levels".

Example

This command sets the verbosity level of the cube operations table to low:

```sql
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -
    dbms_cube_log.target_table, dbms_cube_log.level_low);
```
42.2.11 LEVEL_LOWEST Function

This function returns the integer value of the lowest verbosity level. This level logs the status of each command used to build the cube dimensions and cubes, the use of slave processes, and summary records.

**Syntax**

```
DBMS_CUBE_LOG.LEVEL_LOWEST ()
    RETURN BINARY_INTEGER;
```

**Returns**

1

**Usage Notes**

Use this function instead of its binary integer equivalent for the LOG_LEVEL parameter in DBMS_CUBE_LOG subprograms. See "Verbosity Levels".

**Example**

This command sets the verbosity level of the cube operations table to lowest:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -
    dbms_cube_log.target_table, dbms_cube_log.level_lowest);
```

---

42.2.12 LEVEL_MEDIUM Function

This function returns the integer value of the medium verbosity level.

**Syntax**

```
DBMS_CUBE_LOG.LEVEL_MEDIUM ()
    RETURN BINARY_INTEGER;
```

**Returns**

3

**Usage Notes**

Use this function instead of its binary integer equivalent for the LOG_LEVEL parameter in DBMS_CUBE_LOG subprograms. See "Verbosity Levels".

**Example**

This command sets the verbosity level of the cube operations table to medium:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -
    dbms_cube_log.target_table, dbms_cube_log.level_medium);
```
42.2.13 SET_LOG_SPEC Procedure

This procedure sets all logging to the values specified in the input string.

Syntax

```
DBMS_CUBE_LOG.SET_LOG_SPEC (
    LOG_SPEC       IN   VARCHAR2);
```

Parameters

Table 42-7  SET_LOG_SPEC Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_spec</td>
<td>A string consisting of type(target) pairs.</td>
</tr>
<tr>
<td></td>
<td>Type can be:</td>
</tr>
<tr>
<td></td>
<td>• OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>• REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>• DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td>Target can be:</td>
</tr>
<tr>
<td></td>
<td>• TABLE</td>
</tr>
<tr>
<td></td>
<td>• TRACE</td>
</tr>
<tr>
<td></td>
<td>• FILE</td>
</tr>
<tr>
<td></td>
<td>• LOB</td>
</tr>
</tbody>
</table>

Usage Notes

The GET_LOG_SPEC function returns a properly formatted string for SET_LOG_SPEC.

Examples

This PL/SQL procedure disables all logs, verifies that they are disabled, then activates the Cube Operations log and the Cube Rejected Records log.

```
BEGIN
    dbms_cube_log.disable;
    dbms_output.put_line('Cube Logs: ' || dbms_cube_log.get_log_spec);

    dbms_cube_log.set_log_spec('OPERATIONS(TRACE) REJECTED_RECORDS(TABLE)');
    dbms_output.put_line('Cube Logs: ' || dbms_cube_log.get_log_spec);
END;
/
```

The output from the procedure verifies that the DISABLE function de-activated all logs, and the SET_LOG_SPEC function activated two logs:

Cube Logs:

Cube Logs: OPERATIONS(TRACE) REJECTED_RECORDS(TABLE)
42.2.14 SET_PARAMETER Procedure

This procedure sets options that control various aspects of logging.

To obtain the current value of these options, use the GET_PARAMETER function.

Syntax

```
DBMS_CUBE_LOG.SET_PARAMETER ( 
    LOG_TYPE IN BINARY_INTEGER,
    LOG_PARAMETER IN BINARY_INTEGER,
    VALUE IN BINARY_INTEGER );
```

Parameters

**Table 42-8**  SET_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_type</td>
<td>One of the following log types:</td>
</tr>
<tr>
<td></td>
<td>• 1: TYPE_OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>• 2: TYPE_REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>• 3: TYPE_DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td>• 4: TYPE_BUILD</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Types&quot;.</td>
</tr>
<tr>
<td>log_parameter</td>
<td>One of the following parameters:</td>
</tr>
<tr>
<td></td>
<td>• 1: MAX$errors</td>
</tr>
<tr>
<td></td>
<td>Maximum number of records before signalling an end to logging,</td>
</tr>
<tr>
<td></td>
<td>such as the number of rejected records in the Cube Rejected Records log or</td>
</tr>
<tr>
<td></td>
<td>the number of compilation errors in the dimension compilation error log.</td>
</tr>
<tr>
<td></td>
<td>• 2: FLUSH_INTERVAL</td>
</tr>
<tr>
<td></td>
<td>The number of seconds to buffer the records before writing them to a log.</td>
</tr>
<tr>
<td></td>
<td>When this parameter is 0, the records are written directly to the logs</td>
</tr>
<tr>
<td></td>
<td>without buffering.</td>
</tr>
<tr>
<td></td>
<td>• 3: LOG_FULL_RECORD</td>
</tr>
<tr>
<td></td>
<td>Controls logging of rejected records. Set this parameter to one of the</td>
</tr>
<tr>
<td></td>
<td>following constants:</td>
</tr>
<tr>
<td></td>
<td>0: FULL_RECORD_AUTO: Log the full record when no row ID is available.</td>
</tr>
<tr>
<td></td>
<td>1: FULL_RECORD_ALWAYS: Always log the full record.</td>
</tr>
<tr>
<td></td>
<td>2: FULL_RECORD_NEVER: Never log the full record.</td>
</tr>
<tr>
<td></td>
<td>• 4: LOG_EVERY_N</td>
</tr>
<tr>
<td></td>
<td>Enters a progress message every n rows during data maintenance.</td>
</tr>
<tr>
<td></td>
<td>• 5: ALLOW_ERRORS: Displays logging errors, which are initially turned off</td>
</tr>
<tr>
<td></td>
<td>to allow processing to proceed.</td>
</tr>
<tr>
<td>value</td>
<td>The new value of log_parameter.</td>
</tr>
</tbody>
</table>

Examples

This PL/SQL procedure sets the two parameters, then uses the GET_PARAMETER function to show the settings:
BEGIN
    dbms_cube_log.set_parameter(dbms_cube_log.type_rejected_records, 1, 150);
    dbms_cube_log.set_parameter(dbms_cube_log.type_rejected_records, 2, 5);

    dbms_output.put_line('Max rejected records: ' ||
                         dbms_cube_log.get_parameter(dbms_cube_log.type_rejected_records, 1));

    dbms_output.put_line('Buffer time: ' ||
                         dbms_cube_log.get_parameter(dbms_cube_log.type_rejected_records, 2) ||
                         ' seconds');
END;
/

The procedure displays this information:

Max rejected records: 150
Buffer time: 5 seconds

42.2.15 TABLE_CREATE Procedure

This procedure creates the table targets for the OLAP logs. You must have the CREATE TABLE privilege to use this procedure.

TABLE_CREATE also upgrades existing log tables to the current version while preserving the data.

Syntax

DBMS_CUBE_LOG.TABLE_CREATE (  
    LOG_TYPE       IN   BINARY_INTEGER  DEFAULT,  
    TBLNAME        IN   VARCHAR2        DEFAULT );

Parameters

Table 42-9  TABLE_CREATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| log_type  | One of the following log types:  
            | • 1: TYPE_OPERATIONS  
            | • 2: TYPE_REJECTED_RECORDS  
            | • 3: TYPE_DIMENSION_COMPILE  
            | • 4: TYPE_BUILD  
            | See "Logging Types".  
| tblname   | A table name for the log. These are the default names:  
            | • CUBE_OPERATIONS_LOG  
            | • CUBE_REJECTED_RECORDS  
            | • CUBE_DIMENSION_COMPILE  
            | • CUBE_BUILD_LOG  

Examples

This example creates a Cube Operations log table named CUBE_OPERATIONS_LOG:

EXECUTE dbms_cube_log.table_create;
This example creates a Cube Rejected Records log table named CUBE_REJECTED_RECORDS:
EXECUTE dbms_cube_log.table_create(dbms_cube_log.type_rejected_records);

The next example creates a Cube Build log table named MY_BUILD_LOG:
EXECUTE dbms_cube_log.table_create -
   (dbms_cube_log.type_build, 'MY_BUILD_LOG');

### 42.2.16 TARGET_FILE Function

This function returns the integer value of a file target in DBMS_CUBE_LOG subprograms.

**Syntax**

```
DBMS_CUBE_LOG.TARGET_FILE ()
```

**Returns**

3

**Usage Notes**

Use this function instead of its binary integer equivalent for the LOG_LEVEL parameter in DBMS_CUBE_LOG subprograms. See "Logging Targets".

**Example**

This command disables the Cube Operations log file:
EXECUTE dbms_cube_log.disable -
   (dbms_cube_log.type_operations, dbms_cube_log.target_file);

### 42.2.17 TARGET_LOB Function

This function returns the integer value of a LOB target.

**Syntax**

```
DBMS_CUBE_LOG.TARGET_LOB ()
```

**Returns**

4

**Usage Notes**

Use this function instead of its binary integer equivalent for the LOG_LEVEL parameter in DBMS_CUBE_LOG subprograms. See "Logging Targets".

**Example**

This command disables the Cube Operations log LOB:
EXECUTE dbms_cube_log.disable -
   (dbms_cube_log.type_operations, dbms_cube_log.target_lob);
42.2.18 TARGET_TABLE Function

This function returns the integer value of a table target.

Syntax

```
DBMS_CUBE_LOG.TARGET_TABLE ()
  RETURN BINARY_INTEGER;
```

Returns

1

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_TARGET parameter in DBMS_CUBE_LOG subprograms. See "Logging Targets".

Example

This command disables the Cube Operations log table:

```
EXECUTE dbms_cube_log.disable -
  (dbms_cube_log.type_operations, dbms_cube_log.target_table);
```

42.2.19 TARGET_TRACE Function

This function returns the integer value of a trace file target.

Syntax

```
DBMS_CUBE_LOG.TARGET_TRACE ()
  RETURN BINARY_INTEGER;
```

Returns

2

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_TARGET parameter in DBMS_CUBE_LOG subprograms. See "Logging Targets".

Example

This command disables the Cube Operations log trace file:

```
EXECUTE dbms_cube_log.disable -
  (dbms_cube_log.type_operations, dbms_cube_log.target_trace);
```
42.2.20 TYPE_BUILD Function

This function returns the integer value of the Cube Build log.

Syntax

```sql
DBMS_CUBE_LOG.TYPE_BUILD ()
   RETURN BINARY_INTEGER;
```

Returns

4

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_TYPE parameter in DBMS_CUBE_LOG subprograms. See "Logging Types".

Example

This query returns the default name of a Cube Build log:

```sql
SELECT dbms_cube_log.default_name(dbms_cube_log.type_build) "Log Name" -
   FROM dual;
```

Log Name

-------------------------
CUBE_BUILD_LOG

42.2.21 TYPE_DIMENSION_COMPILE Function

This function returns the integer value of the Cube Dimension Compile log.

Syntax

```sql
DBMS_CUBE_LOG.TYPE_DIMENSION_COMPILE ()
   RETURN BINARY_INTEGER;
```

Returns

3

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_TYPE parameter in DBMS_CUBE_LOG subprograms. See "Logging Types".

Example

This query returns the default name of a Cube Dimension Compile log:

```sql
SELECT dbms_cube_log.default_name(dbms_cube_log.type_dimension_compile) -
   "Log Name" FROM dual;
```

Log Name

-------------------------
CUBE_DIMENSION_COMPILE
42.2.22 TYPE_OPERATIONS Function

This function returns the integer value of the Cube Operations log.

Syntax

```sql
DBMS_CUBE_LOG.TYPE_OPERATIONS ()
RETURN BINARY_INTEGER;
```

Returns

1

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_TYPE parameter in DBMS_CUBE_LOG subprograms. See "Logging Types".

Example

This query returns the default name of a Cube Dimension Compile log:

```sql
SELECT dbms_cube_log.default_name(dbms_cube_log.type_operations) "Log Name" -
     FROM dual;
```

Log Name

```
-------------------------
CUBE_OPERATIONS_LOG
```

42.2.23 TYPE_REJECTED_RECORDS Function

This function returns the integer value of the cube Cube Rejected Records log.

Syntax

```sql
DBMS_CUBE_LOG.TYPE_REJECTED_RECORDS ()
RETURN BINARY_INTEGER;
```

Returns

2

Usage Notes

Use this function instead of its binary integer equivalent for the LOG_TYPE parameter in DBMS_CUBE_LOG subprograms. See "Logging Types".

Example

This query returns the default name of a Cube Rejected Records log:

```sql
SELECT dbms_cube_log.default_name(dbms_cube_log.type_rejected_records) "Log Name" -
     FROM dual;
```

Log Name

```
-------------------------
CUBE_REJECTED_RECORDS
```
42.2.24 VERSION Function

This function returns the version number of a specific log table or the current version number of a specific log type.

Syntax

```
DBMS_CUBE_LOG.VERSION (LOG_TYPE IN BINARY_INTEGER DEFAULT 1,
                        TBLNAME IN VARCHAR2 DEFAULT NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 42-10 VERSION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_type</td>
<td>One of the following log types:</td>
</tr>
<tr>
<td></td>
<td>1: TYPE_OPERATIONS</td>
</tr>
<tr>
<td></td>
<td>2: TYPE_REJECTED_RECORDS</td>
</tr>
<tr>
<td></td>
<td>3: TYPE_DIMENSION_COMPILE</td>
</tr>
<tr>
<td></td>
<td>4: TYPE_BUILD</td>
</tr>
<tr>
<td></td>
<td>See &quot;Logging Types&quot;.</td>
</tr>
<tr>
<td>tblname</td>
<td>The name of the log table whose version is returned.</td>
</tr>
</tbody>
</table>

Returns

A version number

Examples

This example returns the current version of the Cube Operations log:

```
SELECT dbms_cube_log.version FROM dual;
```

```
VERSION
----------
2
```

This example returns the version number of an existing Cube Rejected Records log named CUBE_REJECTED_RECORDS.

```
SELECT dbms_cube_log.version(dbms_cube_log.type_rejected_records, -
'CUBE_REJECTED_RECORDS') version FROM dual;
```

```
VERSION
----------
2
```
The `DBMS_DATA_MINING` package is the application programming interface for creating, evaluating, and querying data mining models.

This chapter contains the following topics:

- Overview
- Security Model
- Mining Functions
- Model Settings
- Solver Settings
- Datatypes
- Summary of DBMS_DATA_MINING Subprograms

See Also:

- *Oracle Data Mining Concepts*
- *Oracle Data Mining User’s Guide*
- `DBMS_DATA_MINING_TRANSFORM`
- `DBMS_PREDICTIVE_ANALYTICS`

### 43.1 DBMS_DATA_MINING Overview

Oracle Data Mining supports both supervised and unsupervised data mining. Supervised data mining predicts a target value based on historical data. Unsupervised data mining discovers natural groupings and does not use a target. You can use Oracle Data Mining to mine structured data and unstructured text.

Supervised data mining functions include:

- Classification
- Regression
- Feature Selection (Attribute Importance)

Unsupervised data mining functions include:

- Clustering
- Association
- Feature Extraction
- Anomaly Detection
The steps you use to build and apply a mining model depend on the data mining function and the algorithm being used. The algorithms supported by Oracle Data Mining are listed in Table 43-1.

**Table 43-1 Oracle Data Mining Algorithms**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Abbreviation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apriori</td>
<td>AR</td>
<td>Association</td>
</tr>
<tr>
<td>CUR Matrix Decomposition</td>
<td>CUR</td>
<td>Attribute Importance</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>DT</td>
<td>Classification</td>
</tr>
<tr>
<td>Expectation Maximization</td>
<td>EM</td>
<td>Clustering</td>
</tr>
<tr>
<td>Explicit Semantic Analysis</td>
<td>ESA</td>
<td>Feature Extraction, Classification</td>
</tr>
<tr>
<td>Exponential Smoothing</td>
<td>ESM</td>
<td>Time Series</td>
</tr>
<tr>
<td>Generalized Linear Model</td>
<td>GLM</td>
<td>Classification, Regression</td>
</tr>
<tr>
<td>k-Means</td>
<td>KM</td>
<td>Clustering</td>
</tr>
<tr>
<td>Minimum Descriptor Length</td>
<td>MDL</td>
<td>Attribute Importance</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>NB</td>
<td>Classification</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>NN</td>
<td>Classification</td>
</tr>
<tr>
<td>Non-Negative Matrix Factorization</td>
<td>NMF</td>
<td>Feature Extraction</td>
</tr>
<tr>
<td>Orthogonal Partitioning Clustering</td>
<td>O-Cluster</td>
<td>Clustering</td>
</tr>
<tr>
<td>Random Forest</td>
<td>RF</td>
<td>Classification</td>
</tr>
<tr>
<td>Singular Value Decomposition and Principal Component Analysis</td>
<td>SVD and PCA</td>
<td>Feature Extraction</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>SVM</td>
<td>Classification, Regression, Anomaly Detection</td>
</tr>
</tbody>
</table>

Oracle Data Mining supports more than one algorithm for the classification, regression, clustering, and feature extraction mining functions. Each of these mining functions has a default algorithm, as shown in Table 43-2.

**Table 43-2 Oracle Data Mining Default Algorithms**

<table>
<thead>
<tr>
<th>Mining Function</th>
<th>Default Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>Naive Bayes</td>
</tr>
<tr>
<td>Clustering</td>
<td>k-Means</td>
</tr>
<tr>
<td>Feature Extraction</td>
<td>Non-Negative Matrix Factorization</td>
</tr>
<tr>
<td>Feature Selection</td>
<td>Minimum Descriptor Length</td>
</tr>
<tr>
<td>Regression</td>
<td>Support Vector Machine</td>
</tr>
</tbody>
</table>

**43.2 DBMS_DATA_MINING Security Model**

The DBMS_DATA_MINING package is owned by user SYS and is installed as part of database installation. Execution privilege on the package is granted to public.
routines in the package are run with invokers' rights (run with the privileges of the current user).

The DBMS_DATA_MINING package exposes APIs that are leveraged by the Oracle Data Mining component of the Advanced Analytics Option. Users who wish to create mining models in their own schema require the CREATE_MINING_MODEL system privilege. Users who wish to create mining models in other schemas require the CREATE_ANY_MINING_MODEL system privilege.

Users have full control over managing models that exist within their own schema. Additional system privileges necessary for managing data mining models in other schemas include ALTER_ANY_MINING_MODEL, DROP_ANY_MINING_MODEL, SELECT_ANY_MINING_MODEL, COMMENT_ANY_MINING_MODEL, and AUDIT_ANY.

Individual object privileges on mining models, ALTER_MINING_MODEL and SELECT_MINING_MODEL, can be used to selectively grant privileges on a model to a different user.

See Also:

Oracle Data Mining User's Guide for more information about the security features of Oracle Data Mining

43.3 DBMS_DATA_MINING — Mining Functions

A data mining function refers to the methods for solving a given class of data mining problems.

The mining function must be specified when a model is created. (See CREATE_MODEL Procedure.)

Table 43-3  Mining Functions

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSOCIATION</td>
<td>Association is a descriptive mining function. An association model identifies relationships and the probability of their occurrence within a data set. Association models use the Apriori algorithm.</td>
</tr>
<tr>
<td>ATTRIBUTE_IMPORTANCE</td>
<td>Attribute importance is a predictive mining function, also known as feature selection. An attribute importance model identifies the relative importance of an attribute in predicting a given outcome. Attribute importance models can use Minimum Description Length, or CUR Matrix Decomposition. Minimum Description Length is the default.</td>
</tr>
<tr>
<td>CLASSIFICATION</td>
<td>Classification is a predictive mining function. A classification model uses historical data to predict a categorical target. Classification models can use: Naive Bayes, Decision Tree, Logistic Regression, or Support Vector Machine. The default is Naive Bayes. The classification function can also be used for anomaly detection. In this case, the SVM algorithm with a null target is used (One-Class SVM).</td>
</tr>
</tbody>
</table>
Table 43-3 (Cont.) Mining Functions

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTERING</td>
<td>Clustering is a descriptive mining function. A clustering model identifies natural groupings within a data set. Clustering models can use ( k )-Means, O-Cluster, or Expectation Maximization. The default is ( k )-Means.</td>
</tr>
<tr>
<td>FEATURE_EXTRACTION</td>
<td>Feature Extraction is a descriptive mining function. A feature extraction model creates an optimized data set on which to base a model. Feature extraction models can use Explicit Semantic Analysis, Non-Negative Matrix Factorization, Singular Value Decomposition, or Principal Component Analysis. Non-Negative Matrix Factorization is the default.</td>
</tr>
<tr>
<td>REGRESSION</td>
<td>Regression is a predictive mining function. A regression model uses historical data to predict a numerical target. Regression models can use Support Vector Machine or Linear Regression. The default is Support Vector Machine.</td>
</tr>
<tr>
<td>TIME_SERIES</td>
<td>Time series is a predictive mining function. A time series model forecasts the future values of a time-ordered series of historical numeric data over a user-specified time window. Time series models use the Exponential Smoothing algorithm.</td>
</tr>
</tbody>
</table>

See Also:
Oracle Data Mining Concepts for more information about mining functions

43.4 DBMS_DATA_MINING — Model Settings

Oracle Data Mining uses settings to specify the algorithm and other characteristics of a model. Some settings are general, some are specific to a mining function, and some are specific to an algorithm.

All settings have default values. If you want to override one or more of the settings for a model, you must create a settings table. The settings table must have the column names and datatypes shown in the following table.

Table 43-4 Required Columns in the Model Settings Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETTING_NAME</td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>SETTING_VALUE</td>
<td>VARCHAR2 (4000)</td>
</tr>
</tbody>
</table>

The information you provide in the settings table is used by the model at build time. The name of the settings table is an optional argument to the CREATE_MODEL Procedure.

You can find the settings used by a model by querying the data dictionary view ALL_MINING_MODEL_SETTINGS. This view lists the model settings used by the mining
models to which you have access. All the setting values are included in the view, whether
default or user-specified.

See Also:

- ALL_MINING_MODEL_SETTINGS in Oracle Database Reference
- Oracle Data Mining User's Guide for information about specifying model settings

43.4.1 DBMS_DATA_MINING — Algorithm Names

The ALGO_NAME setting specifies the model algorithm.

The values for the ALGO_NAME setting are listed in the following table.

Table 43-5 Algorithm Names

<table>
<thead>
<tr>
<th>ALGO_NAME Value</th>
<th>Description</th>
<th>Mining Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_AI_MDL</td>
<td>Minimum Description Length</td>
<td>Attribute Importance</td>
</tr>
<tr>
<td>ALGO_APRIORI_ASSOCIATION_RULES</td>
<td>Apriori</td>
<td>Association Rules</td>
</tr>
<tr>
<td>ALGO_CUR_DECOMPOSITION</td>
<td>CUR Decomposition</td>
<td>Attribute Importance</td>
</tr>
<tr>
<td>ALGO_DECISION_TREE</td>
<td>Decision Tree</td>
<td>Classification</td>
</tr>
<tr>
<td>ALGO_EXPECTATION_MAXIMIZATION</td>
<td>Expectation Maximization</td>
<td>Clustering</td>
</tr>
<tr>
<td>ALGO_EXPLICIT_SEMANTIC_ANALYSIS</td>
<td>Explicit Semantic Analysis</td>
<td>Feature Extraction</td>
</tr>
<tr>
<td>ALGO_EXPLICIT_SEMANTIC_ANALYSIS</td>
<td>Exponential Smoothing</td>
<td>Classification</td>
</tr>
<tr>
<td>ALGO_GENERALIZED_LINEAR_MODEL</td>
<td>Language used for extensible</td>
<td>All mining functions supported</td>
</tr>
<tr>
<td></td>
<td>algorithm</td>
<td></td>
</tr>
<tr>
<td>ALGOGENERALIZED_LINEAR_MODEL</td>
<td>Generalized Linear Model</td>
<td>Classification, Regression; also Feature Selection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and Generation</td>
</tr>
<tr>
<td>ALGO_KMEANS</td>
<td>Enhanced k_Means</td>
<td>Clustering</td>
</tr>
<tr>
<td>ALGO_NAIVE_BAYES</td>
<td>Naive Bayes</td>
<td>Classification</td>
</tr>
<tr>
<td>ALGO_NEURAL_NETWORK</td>
<td>Neural Network</td>
<td>Classification</td>
</tr>
<tr>
<td>ALGO_NONNEGATIVE_MATRIX_FACTOR</td>
<td>Non-Negative Matrix Factorization</td>
<td>Feature Extraction</td>
</tr>
<tr>
<td>ALGO_O_CLUSTER</td>
<td>O-Cluster</td>
<td>Clustering</td>
</tr>
<tr>
<td>ALGO_RANDOM_FOREST</td>
<td>Random Forest</td>
<td>Classification</td>
</tr>
<tr>
<td>ALGO_SINGULAR_VALUE_DECOMP</td>
<td>Singular Value Decomposition</td>
<td>Feature Extraction</td>
</tr>
<tr>
<td>ALGO_SUPPORT_VECTOR_MACHINES</td>
<td>Support Vector Machine</td>
<td>Classification and Regression</td>
</tr>
</tbody>
</table>
Oracle Data Mining supports fully Automatic Data Preparation (ADP), user-directed general data preparation, and user-specified embedded data preparation. The `PREP_*` settings enable the user to request fully automated or user-directed general data preparation. By default, fully Automatic Data Preparation (`PREP_AUTO_ON`) is enabled.

When you enable Automatic Data Preparation, the model uses heuristics to transform the build data according to the requirements of the algorithm. Instead of fully Automatic Data Preparation, the user can request that the data be shifted and/or scaled with the `PREP_SCALE*` and `PREP_SHIFT*` settings. The transformation instructions are stored with the model and reused whenever the model is applied. Refer to Model Detail Views, Oracle Data Mining User's Guide.

You can choose to supplement Automatic Data Preparations by specifying additional transformations in the `xform_list` parameter when you build the model. (See "CREATE_MODEL Procedure".)

If you do not use Automatic Data Preparation and do not specify transformations in the `xform_list` parameter to `CREATE_MODEL`, you must implement your own transformations separately in the build, test, and scoring data. You must take special care to implement the exact same transformations in each data set.

If you do not use Automatic Data Preparation, but you do specify transformations in the `xform_list` parameter to `CREATE_MODEL`, Oracle Data Mining embeds the transformation definitions in the model and prepares the test and scoring data to match the build data.

The values for the `PREP_*` setting are described in the following table.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREP_AUTO</td>
<td>· PREP_AUTO_ON</td>
<td>This setting enables fully automated data preparation. The default is PREP_AUTO_ON.</td>
</tr>
<tr>
<td></td>
<td>· PREP_AUTO_OFF</td>
<td></td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for information about algorithms
<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| PREP_SCALE_2DNU M    | · PREP_SCALE_STDDEV    | This setting enables scaling data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values:  
  · PREP_SCALE_STDDEV: A request to divide the column values by the standard deviation of the column and is often provided together with PREP_SHIFT_MEAN to yield z-score normalization.  
  · PREP_SCALE_RANGE: A request to divide the column values by the range of values and is often provided together with PREP_SHIFT_MIN to yield a range of \((0,1]\). |
| PREP_SCALE_NNUM      | · PREP_SCALE_MAXABS    | This setting enables scaling data preparation for nested numeric columns. PREP_AUTO must be OFF for this setting to take effect. If specified, then the valid value for this setting is PREP_SCALE_MAXABS, which yields data in the range of \([-1,1]\). |
| PREP_SHIFT_2DNU M    | · PREP_SHIFT_MEAN      | This setting enables centering data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values:  
  · PREP_SHIFT_MEAN: Results in subtracting the average of the column from each value.  
  · PREP_SHIFT_MIN: Results in subtracting the minimum of the column from each value. |

See Also:  
Oracle Data Mining User’s Guide for information about data transformations

43.4.3 DBMS_DATA_MINING — Mining Function Settings

The settings described in this table apply to a mining function.
### Table 43-7  Mining Function Settings

<table>
<thead>
<tr>
<th>Mining Function</th>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association</td>
<td>ASSO_MAX_RULE_LENGTH</td>
<td>```TO_CHAR( 2 &lt;= numeric_exp r &lt;= 20)`</td>
<td>Maximum rule length for Association Rules. Default is 4.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_MIN_CONFIDENCE</td>
<td>```TO_CHAR( 0 &lt;= numeric_exp r &lt;= 1)`</td>
<td>Minimum confidence for Association Rules. Default is 0.1.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_MIN_SUPPORT</td>
<td>```TO_CHAR( 0 &lt;= numeric_exp r &lt;= 1)`</td>
<td>Minimum support for Association Rules. Default is 0.1.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_MIN_SUPPORT_INT</td>
<td>```TO_CHAR( 0 &lt;= numeric_exp r &lt;= 1)`</td>
<td>Minimum absolute support that each rule must satisfy. The value must be an integer. Default is 1.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_MIN_REV_CONFIDENCE</td>
<td>```TO_CHAR( 0 &lt;= numeric_exp r &lt;= 1)`</td>
<td>Sets the Minimum Reverse Confidence that each rule should satisfy. The Reverse Confidence of a rule is defined as the number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs. The value is a real number between 0 and 1. The default is 0.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_IN_RULES</td>
<td>NULL</td>
<td>Sets Including Rules applied for each association rule: it specifies the list of items that at least one of them must appear in each reported association rule, either as antecedent or as consequent. It is a comma separated string containing the list of including items. If not set, the default behavior is, the filtering is not applied.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_EX_RULES</td>
<td>NULL</td>
<td>Sets Excluding Rules applied for each association rule: it specifies the list of items that none of them can appear in each reported Association Rules. It is a comma separated string containing the list of excluding items. No rule can contain any item in the list. The default is NULL.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_ANT_IN_RULES</td>
<td>NULL</td>
<td>Sets Including Rules for the antecedent: it specifies the list of items that at least one of them must appear in the antecedent part of each reported association rule. It is a comma separated string containing the list of including items. The antecedent part of each rule must contain at least one item in the list. The default is NULL.</td>
</tr>
</tbody>
</table>
### Table 43-7 (Cont.) Mining Function Settings

<table>
<thead>
<tr>
<th>Mining Function</th>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association</td>
<td>ASSO_ANT_EX_RULES</td>
<td>NULL</td>
<td>Sets Excluding Rules for the antecedent: it specifies the list of items that none of them can appear in the antecedent part of each reported rule. The default is NULL.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_CONS_IN_RULES</td>
<td>NULL</td>
<td>Sets Including Rules for the consequent: it specifies the list of items that at least one of them must appear in the consequent part of each reported rule. The default is NULL.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_CONS_EX_RULES</td>
<td>NULL</td>
<td>Sets Excluding Rules for the consequent: it specifies the list of items that none of them can appear in the consequent part of each reported rule. The default is NULL.</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_AGGREGATES</td>
<td>NULL</td>
<td>Specifies the columns to be aggregated. It is a comma separated string containing the names of the columns for aggregation. Number of columns in the list must be &lt;= 10. You can set ASSO_AGGREGATES if ODMS_ITEM_ID_COLUMN_NAME is set indicating transactional input data. See DBMS_DATA_MINING - Global Settings. The data table must have valid column names such as ITEM_ID and CASE_ID which are derived from ODMS_ITEM_ID_COLUMN_NAME and case_id_column_name respectively. ITEM_VALUE is not a mandatory value. The default is NULL. For each item, the user may supply several columns to aggregate. It requires more memory to buffer the extra data. Also, the performance impact can be seen because of the larger input data set and more operation.</td>
</tr>
</tbody>
</table>
### Table 43-7  (Cont.) Mining Function Settings

<table>
<thead>
<tr>
<th>Mining Function</th>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association</td>
<td>ASSO_ABS_ERROR</td>
<td>(0 &lt; \text{ASSO_ABS_ERROR} \leq \text{MAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE)}).</td>
<td>Specifies the absolute error for the Association Rules sampling. A smaller value of \text{ASSO_ABS_ERROR} obtains a larger sample size which gives accurate results but takes longer computational time. Set a reasonable value for \text{ASSO_ABS_ERROR}, such as its default value, to avoid large sample size. The default value is (0.5 \times \text{MAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE)}).</td>
</tr>
<tr>
<td>Association</td>
<td>ASSO_CONF_LEVEL</td>
<td>0</td>
<td>Specifies the confidence level for an Association Rules sample. A larger value of \text{ASSO_CONF_LEVEL} obtains a larger sample size. Any value between 0.9 and 1 is suitable. The default value is 0.95.</td>
</tr>
</tbody>
</table>
### Table 43-7  (Cont.) Mining Function Settings

<table>
<thead>
<tr>
<th>Mining Function</th>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>ON, OFF</td>
<td>This setting indicates that the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.</td>
</tr>
<tr>
<td>Clustering</td>
<td>CLUS_NUM_CLUSTERS</td>
<td>TO_CHAR( numeric_expr</td>
<td>Maximum number of leaf clusters generated by a clustering algorithm. The algorithm may return fewer clusters, depending on the data. Enhanced k-Means usually produces the exact number of clusters specified by CLUS_NUM_CLUSTERS, unless there are fewer distinct data points. Expectation Maximization (EM) may return fewer clusters than the number specified by CLUS_NUM_CLUSTERS depending on the data. The number of clusters returned by EM cannot be greater than the number of components, which is governed by algorithm-specific settings. (See Expectation Maximization Settings for Learning table) Depending on these settings, there may be fewer clusters than components. If component clustering is disabled, the number of clusters equals the number of components. For EM, the default value of CLUS_NUM_CLUSTERS is system-determined. For k-Means and O-Cluster, the default is 10.</td>
</tr>
<tr>
<td>Feature Extraction</td>
<td>FEAT_NUM_FEATURES</td>
<td>TO_CHAR( numeric_expr</td>
<td>Number of features to be extracted by a feature extraction model. The default is estimated from the data by the algorithm. If the matrix rank is smaller than this number, fewer features will be returned. For CUR Matrix Decomposition, the FEAT_NUM_FEATURES value is same as the CURS_SVD_RANK value.</td>
</tr>
</tbody>
</table>

**See Also:**

*Oracle Data Mining Concepts* for information about mining functions

### 43.4.4 DBMS_DATA_MINING — Global Settings

The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.
### Table 43-8  Global Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| ODMS_ITEM_ID_COLUMN_NAME     | column_name                                        | (Association rules only) Name of a column that contains the items in a transaction. When this setting is specified, the algorithm expects the data to be presented in a native transactional format, consisting of two columns:  
  - Case ID, either categorical or numeric  
  - Item ID, either categorical or numeric  
  A typical example of transactional data is market basket data, wherein a case represents a basket that may contain many items. Each item is stored in a separate row, and many rows may be needed to represent a case. The case ID values do not uniquely identify each row. Transactional data is also called multi-record case data.  
  Association rules function is normally used with transactional data, but it can also be applied to single-record case data (similar to other algorithms).  
  For more information about single-record and multi-record case data, see Oracle SQL Developer Data Modeler User's Guide. |
| ODMS_ITEM_VALUE_COLUMN_NAME  | column_name                                        | (Association rules only) Name of a column that contains a value associated with each item in a transaction. This setting is only used when a value has been specified for ODMS_ITEM_ID_COLUMN_NAME indicating that the data is presented in native transactional format.  
  If ASSO_AGGREGATES is used, then the build data must include the following three columns and the columns specified in the AGGREGATES setting.  
  - Case ID, either categorical or numeric  
  - Item ID, either categorical or numeric, specified by ODMS_ITEM_ID_COLUMN_NAME  
  - Item value, either categorical or numeric, specified by ODMS_ITEM_VALUE_COLUMN_NAME  
  If ASSO_AGGREGATES, Case ID, and Item ID column are present, then the Item Value column may or may not appear.  
  The Item Value column may specify information such as the number of items (for example, three apples) or the type of the item (for example, macintosh apples).  
  For details on ASSO_AGGREGATES, see DBMS_DATA_MINING - Mining Function Settings. |
### Table 43-8  (Cont.) Global Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_MEAN_MODE</td>
<td>Indicates how to treat missing values in the training data. This setting does not affect the scoring data. The default value is ODMS_MISSING_VALUE_AUTO.</td>
</tr>
<tr>
<td></td>
<td>ODMS_MISSING_VALUE_DELETE_ROW</td>
<td>Indicates how to treat missing values in the training data. This setting does not affect the scoring data. The default value is ODMS_MISSING_VALUE_AUTO.</td>
</tr>
<tr>
<td></td>
<td>ODMS_MISSING_VALUE_AUTO</td>
<td>Indicates how to treat missing values in the training data. This setting does not affect the scoring data. The default value is ODMS_MISSING_VALUE_AUTO.</td>
</tr>
<tr>
<td>ODMS_ROW_WEIGHT_COLUMN_NAME</td>
<td>column_name</td>
<td>(GLM only) Name of a column in the training data that contains a weighting factor for the rows. The column data type must be NUMBER. Row weights can be used as a compact representation of repeated rows, as in the design of experiments where a specific configuration is repeated several times. Row weights can also be used to emphasize certain rows during model construction. For example, to bias the model towards rows that are more recent and away from potentially obsolete data.</td>
</tr>
<tr>
<td>ODMS_TEXT_POLICY_NAME</td>
<td>The name of an Oracle Text POLICY created using CTX_DDL.CREATE_POLICY</td>
<td>Affects how individual tokens are extracted from unstructured text. For details about CTX_DDL.CREATE_POLICY, see Oracle Text Reference.</td>
</tr>
<tr>
<td>ODMS_TEXT_MAX_FEATURES</td>
<td>1 &lt;= value</td>
<td>The maximum number of distinct features, across all text attributes, to use from a document set passed to CREATE_MODEL. The default is 3000. ESA has the default value of 300000.</td>
</tr>
<tr>
<td>ODMS_TEXT_MIN_DOCUMENTS</td>
<td>Non-negative value</td>
<td>This is a text processing setting the controls how in many documents a token needs to appear to be used as a feature. The default is 1. ESA has a default of 3.</td>
</tr>
<tr>
<td>ODMS_PARTITION_COLUMNS</td>
<td>Comma separated list of machine learning attributes</td>
<td>This setting indicates a request to build a partitioned model. The setting value is a comma-separated list of the machine learning attributes to be used to determine the in-list partition key values. These machine learning attributes are taken from the input columns unless an XFORM_LIST parameter is passed to CREATE_MODEL. If the XFORM_LIST parameter is passed to CREATE_MODEL, then the machine learning attributes are taken from the attributes produced by these transformations.</td>
</tr>
<tr>
<td>Setting Name</td>
<td>Setting Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ODMS_MAX_PARTITIONS</td>
<td>1&lt; value &lt;= 1000000</td>
<td>This setting indicates the maximum number of partitions allowed for the model. The default is 1000.</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_ENABLE</td>
<td>This setting allows the user to request a sampling of the build data. The default is ODMS_SAMPLING_DISABLE.</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
<td></td>
</tr>
<tr>
<td>ODMS_SAMPLE_SIZE</td>
<td>0 &lt; Value</td>
<td>This setting determines how many rows will be sampled (approximately). It can be set only if ODMS_SAMPLING is enabled. The default value is the system determined.</td>
</tr>
</tbody>
</table>
| ODMS_PARTITION_BUILD_TYPE | ODMS_PARTITION_BUILD_INTRA              | This setting controls the parallel build of partitioned models.  
ODMS_PARTITION_BUILD_INTER | Each partition is built entirely in a single slave, but multiple partitions may be built at the same time since multiple slaves are active.  
ODMS_PARTITION_BUILD_HYBRID | It is a combination of the other two types and is recommended for most situations to adapt to dynamic environments.  
The default mode is ODMS_PARTITION_BUILD_HYBRID |
| ODMS_TABLESPACE_NAME    | tablespace_name                         | This setting controls the storage specifications. If you explicitly sets this to the name of a tablespace (for which you have sufficient quota), then the specified tablespace storage creates the resulting model content. If you do not provide this setting, then the default tablespace of the user creates the resulting model content. |
| ODMS_RANDOM_SEED        | The value must be a non-negative integer | The hash function with a random number seed generates a random number with uniform distribution. Users can control the random number seed by this setting. The default is 0.  
This setting is used by Random Forest, Neural Network, and CUR Matrix Decomposition. |
| ODMSDETAILS             | • ODMS_ENABLE                           | This setting reduces the space that is used while creating a model, especially a partitioned model. The default value is ODMS_ENABLE.  
When the setting is ODMS_ENABLE, it creates model tables and views when the model is created. You can query the model with SQL. When the setting is ODMS_DISABLE, model views are not created and tables relevant to model details are not created either.  
The reduction in space depends on the model. Reduction on the order of 10x can be achieved. |
43.4.5 DBMS_DATA_MINING — Algorithm Settings:
ALGO_EXTENSIBLE_LANG

The settings listed in the following table configure the behavior of the mining model with an Extensible algorithm. The mining model is built in R language.

The `RALG_*_FUNCTION` specifies the R script that is used to build, score, and view R model and must be registered in ORACLR Script repository. The R scripts are registered through Oracle Enterprise R product with special privilege. When `ALGO_EXTENSIBLE_LANG` is set to R in the `MINING_MODEL_SETTING` table, the mining model is built in the R language. After the R model is built, the names of the R scripts are recorded in `MINING_MODEL_SETTING` table in the `SYS` schema. The scripts must exist in the repository for the R model to function. The amount of R memory used to build, score, and view the R model through these R scripts can be controlled by Oracle Enterprise R.

All algorithm-independent `DBMS_DATA_MINING` subprograms can operate on the R model for mining functions such as Association, Attribute Importance, Classification, Clustering, Feature Extraction, and Regression.

The supported `DBMS_DATA_MINING` subprograms include, but are not limited, to the following:

- ADD_COST_MATRIX Procedure
- COMPUTE_CONFUSION_MATRIX Procedure
- COMPUTE_LIFT Procedure
- COMPUTE_ROC Procedure
- CREATE_MODEL Procedure
- DROP_MODEL Procedure
- EXPORT_MODEL Procedure
- GET_MODEL_COST_MATRIX Function
- IMPORT_MODEL Procedure
- REMOVE_COST_MATRIX Procedure
- RENAME_MODEL Procedure
### Table 43-9  ALGO_EXTENSIBLE_LANG Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RALG_BUILD_FUNCTION</td>
<td>R_BUILD_FUNCTION_SCRIPT_NAME</td>
<td>Specifies the name of an existing registered R script for R algorithm mining model build function. The R script defines an R function for the first input argument for training data and returns an R model object. For Clustering and Feature Extraction mining function model build, the R attributes <code>dm$nclus</code> and <code>dm$nfeat</code> must be set on the R model to indicate the number of clusters and features respectively. The <code>RALG_BUILD_FUNCTION</code> must be set along with <code>ALGO_EXTENSIBLE_LANG</code> in the <code>model_setting_table</code>.</td>
</tr>
<tr>
<td>RALG_BUILD_PARAMETER</td>
<td>SELECT <code>value</code> <code>param_name</code>, ... FROM DUAL</td>
<td>Specifies a list of numeric and string scalar for optional input parameters of the model build function.</td>
</tr>
<tr>
<td>RALG_SCORE_FUNCTION</td>
<td>R_SCORE_FUNCTION_SCRIPT_NAME</td>
<td>Specifies the name of an existing registered R script to score data. The script returns a <code>data.frame</code> containing the corresponding prediction results. The setting is used to score data for mining functions such as Regression, Classification, Clustering, and Feature Extraction. This setting does not apply to Association and Attribute Importance functions.</td>
</tr>
<tr>
<td>RALG_WEIGHT_FUNCTION</td>
<td>R_WEIGHT_FUNCTION_SCRIPT_NAME</td>
<td>Specifies the name of an existing registered R script for R algorithm that computes the weight (contribution) for each attribute in scoring. The script returns a <code>data.frame</code> containing the contributing weight for each attribute in a row. This function setting is needed for <code>PREDICTION_DETAILS</code> SQL function.</td>
</tr>
<tr>
<td>RALG_DETAILS_FUNCTION</td>
<td>R_DETAILS_FUNCTION_SCRIPT_NAME</td>
<td>Specifies the name of an existing registered R script for R algorithm that produces the model information. This setting is required to generate a model view.</td>
</tr>
<tr>
<td>RALG_DETAILS_FORMAT</td>
<td>SELECT <code>type_value</code> <code>column_name</code>, ... FROM DUAL</td>
<td>Specifies the <code>SELECT</code> query for the list of numeric and string scalars for the output column type and the column name of the generated model view. This setting is required to generate a model view.</td>
</tr>
</tbody>
</table>

**See Also:**

*Oracle Data Mining User’s Guide*
43.4.6 DBMS_DATA_MINING — Algorithm Settings: CUR Matrix Decomposition

The following settings affects the behavior of the CUR Matrix Decomposition algorithm.

The following settings configure the behavior of the CUR Matrix Decomposition algorithm.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURS_APPROX_ATTR_NUM</td>
<td>The value must be a positive integer</td>
<td>Defines the approximate number of attributes to be selected. The default value is the number of attributes.</td>
</tr>
<tr>
<td>CURS_ROW_IMPORTANCE</td>
<td>CURS_ROW_IMP_ENABLE</td>
<td>Defines the flag indicating whether or not to perform row selection. The default value is CURS_ROW_IMP_DISABLE.</td>
</tr>
<tr>
<td>CURS_ROW_IMP_DISABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CURS_APPROX_ROW_NUM</td>
<td>The value must be a positive integer</td>
<td>Defines the approximate number of rows to be selected. This parameter is only used when users decide to perform row selection (CURS_ROW_IMP_ENABLE). The default value is the total number of rows.</td>
</tr>
<tr>
<td>CURS_SVD_RANK</td>
<td>The value must be a positive integer</td>
<td>Defines the rank parameter used in the column/row leverage score calculation. If users do not provide an input value, the value is determined by the system.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts

43.4.7 DBMS_DATA_MINING — Algorithm Settings: Decision Tree

These settings configure the behavior of the Decision Tree algorithm. Note that the Decision Tree settings are also used to configure the behavior of Random Forest as it constructs each individual Decision Tree.
### Table 43-11  Decision Tree Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE_IMPURITY_METRIC</td>
<td>TREE_IMPURITY_ENTROPY</td>
<td>Tree impurity metric for Decision Tree. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with a metric. Decision trees can use either gini (TREE_IMPURITY_GINI) or entropy (TREE_IMPURITY_ENTROPY) as the purity metric. By default, the algorithm uses TREE_IMPURITY_GINI.</td>
</tr>
<tr>
<td>TREE_IMPURITY_GINI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TREE_TERM_MAX_DEPTH</td>
<td>For Decision Tree: 2&lt;= a number &lt;=20 For Random Forest: 2&lt;= a number &lt;=100</td>
<td>Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). For Decision Tree the default is 7. For Random Forest the default is 16.</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_NODE</td>
<td>0&lt;= a number&lt;=10</td>
<td>The minimum number of training rows in a node expressed as a percentage of the rows in the training data. Default is 0.05, indicating 0.05%.</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_SPLIT</td>
<td>0 &lt; a number &lt;=20</td>
<td>Minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. Default is 0.1, indicating 0.1%.</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_NODE</td>
<td>a number&gt;=0</td>
<td>Minimum number of rows in a node. Default is 10.</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_SPLIT</td>
<td>a number &gt; 1</td>
<td>Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if number of records is below this value. Default is 20.</td>
</tr>
<tr>
<td>CLAS_MAX_SUP_BINS</td>
<td>For Decision Tree: 2 &lt;= a number &lt;=2147483647 For Random Forest: 2 &lt;= a number &lt;=254</td>
<td>This parameter specifies the maximum number of bins for each attribute. Default value is 32.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for information about Decision Tree
43.4.8 DBMS_DATA_MINING — Algorithm Settings: Expectation Maximization

These algorithm settings configure the behavior of the Expectation Maximization algorithm.

- Table 43-12
- Table 43-13
- Table 43-14
- Table 43-15

See Also:
Oracle Data Mining Concepts for information about Expectation Maximization

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_ATTRIBUTE_FILTER</td>
<td>EMCS_ATTR_FILTER_ENABLE</td>
<td>Whether or not to include uncorrelated attributes in the model. When EMCS_ATTRIBUTE_FILTER is enabled, uncorrelated attributes are not included.</td>
</tr>
<tr>
<td></td>
<td>EMCS_ATTR_FILTER_DISABLE</td>
<td></td>
</tr>
<tr>
<td>EMCS_MAX_NUM_ATTR_2D</td>
<td>TO_CHAR(numeric_expr) &gt;=1</td>
<td>Maximum number of correlated attributes to include in the model. Note: This setting applies only to attributes that are not nested (2D). Default is 50.</td>
</tr>
<tr>
<td>EMCS_NUM_DISTRIBUTION</td>
<td>EMCS_NUM_DISTR_BERNOULLI</td>
<td>The distribution for modeling numeric attributes. Applies to the input table or view as a whole and does not allow per-attribute specifications. The options include Bernoulli, Gaussian, or system-determined distribution. When Bernoulli or Gaussian distribution is chosen, all numeric attributes are modeled using the same type of distribution. When the distribution is system-determined, individual attributes may use different distributions (either Bernoulli or Gaussian), depending on the data. Default is EMCS_NUM_DISTR_SYSTEM.</td>
</tr>
<tr>
<td></td>
<td>EMCS_NUM_DISTR_GAUSSIAN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EMCS_NUM_DISTR_SYSTEM</td>
<td></td>
</tr>
</tbody>
</table>
### Table 43-12  (Cont.) Expectation Maximization Settings for Data Preparation and Analysis

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_NUM_EQUIWIDTH_BINS</td>
<td>TO_CHAR(1 &lt; numeric_expr &lt;= 255)</td>
<td>Number of equi-width bins that will be used for gathering cluster statistics for numeric columns. Default is 11.</td>
</tr>
<tr>
<td>EMCS_NUM_PROJECTIONS</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>Specifies the number of projections that will be used for each nested column. If a column has fewer distinct attributes than the specified number of projections, the data will not be projected. The setting applies to all nested columns. Default is 50.</td>
</tr>
<tr>
<td>EMCS_NUM_QUANTILE_BINS</td>
<td>TO_CHAR(1 &lt; numeric_expr &lt;= 255)</td>
<td>Specifies the number of quantile bins that will be used for modeling numeric columns with multivalued Bernoulli distributions. Default is system-determined.</td>
</tr>
<tr>
<td>EMCS_NUM_TOPN_BINS</td>
<td>TO_CHAR(1 &lt; numeric_expr &lt;= 255)</td>
<td>Specifies the number of top-N bins that will be used for modeling categorical columns with multivalued Bernoulli distributions. Default is system-determined.</td>
</tr>
</tbody>
</table>

### Table 43-13  Expectation Maximization Settings for Learning

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_CONVERGENCE_CRITERION</td>
<td>EMCS_CONV_CRIT_HELDASIDE</td>
<td>The convergence criterion for EM. The convergence criterion may be based on a held-aside data set, or it may be Bayesian Information Criterion. Default is system determined.</td>
</tr>
<tr>
<td>EMCS_CONV_CRIT_BIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMCS_LOGLIKE_IMPROVEMENT</td>
<td>TO_CHAR(0 &lt; numeric_expr &lt; 1)</td>
<td>When the convergence criterion is based on a held-aside data set (EMCS_CONVERGENCE_CRITERION = EMCS_CONV_CRIT_HELDASIDE), this setting specifies the percentage improvement in the value of the log likelihood function that is required for adding a new component to the model. Default value is 0.001.</td>
</tr>
<tr>
<td>EMCS_NUM_COMPONENTS</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>Maximum number of components in the model. If model search is enabled, the algorithm automatically determines the number of components based on improvements in the likelihood function or based on regularization, up to the specified maximum. The number of components must be greater than or equal to the number of clusters. Default is 20.</td>
</tr>
<tr>
<td>EMCS_NUM_ITERATIONS</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>Specifies the maximum number of iterations in the EM algorithm. Default is 100.</td>
</tr>
</tbody>
</table>
### Table 43-13  (Cont.) Expectation Maximization Settings for Learning

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_MODEL_SEARCH</td>
<td>EMCS_MODEL_SEARCH_ENA BLE</td>
<td>This setting enables model search in EM where different model sizes are explored and a best size is selected. The default is EMCS_MODEL_SEARCH_DISABLE.</td>
</tr>
<tr>
<td></td>
<td>EMCS_MODEL_SEARCH_DIS ABLE (default)</td>
<td></td>
</tr>
<tr>
<td>EMCS_REMOVE_COMPONENTS</td>
<td>EMCS_REMOVE_COMPS_E NAB LE (default)</td>
<td>This setting allows the EM algorithm to remove a small component from the solution. The default is EMCS_REMOVE_COMPS_ENABLE.</td>
</tr>
<tr>
<td></td>
<td>EMCS_REMOVE_COMPS_DIS ABLE</td>
<td></td>
</tr>
<tr>
<td>EMCS_RANDOM_SEED</td>
<td>Non-negative integer</td>
<td>This setting controls the seed of the random generator used in EM. The default is 0.</td>
</tr>
</tbody>
</table>

### Table 43-14  Expectation Maximization Settings for Component Clustering

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_CLUSTER_COMPONENTS</td>
<td>EMCS_CLUSTER_COMP_E NABLE</td>
<td>Enables or disables the grouping of EM components into high-level clusters. When disabled, the components themselves are treated as clusters. When component clustering is enabled, model scoring through the SQL CLUSTER function will produce assignments to the higher level clusters. When clustering is disabled, the CLUSTER function will produce assignments to the original components. Default is EMCS_CLUSTER_COMP_ENABLE.</td>
</tr>
<tr>
<td></td>
<td>EMCS_CLUSTER_COMP_D ISABLE</td>
<td></td>
</tr>
<tr>
<td>EMCS_CLUSTER_THRESH</td>
<td>TO_CHAR(numeric_expression&gt;=1)</td>
<td>Dissimilarity threshold that controls the clustering of EM components. When the dissimilarity measure is less than the threshold, the components are combined into a single cluster. A lower threshold may produce more clusters that are more compact. A higher threshold may produce fewer clusters that are more spread out. Default is 2.</td>
</tr>
<tr>
<td>EMCS_LINKAGE_FUNCTION</td>
<td>EMCS_LINKAGE_SINGLE</td>
<td>Allows the specification of a linkage function for the agglomerative clustering step. EMCS_LINKAGE_SINGLE uses the nearest distance within the branch. The clusters tend to be larger and have arbitrary shapes. EMCS_LINKAGE_AVERAGE uses the average distance within the branch. There is less chaining effect and the clusters are more compact. EMCS_LINKAGE_COMPLETE uses the maximum distance within the branch. The clusters are smaller and require strong component overlap. Default is EMCS_LINKAGE_SINGLE.</td>
</tr>
<tr>
<td></td>
<td>EMCS_LINKAGE_AVER AGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EMCS_LINKAGE_COMPLE TE</td>
<td></td>
</tr>
</tbody>
</table>


### Table 43-15  Expectation Maximization Settings for Cluster Statistics

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_CLUSTER_STATISTICS</td>
<td>EMCS_CLUS_STATS_ENABLE</td>
<td>Enables or disables the gathering of descriptive statistics for clusters (centroids, histograms, and rules). When statistics are disabled, model size is reduced, and GET_MODELDETAILS_EM only returns taxonomy (hierarchy) and cluster counts. Default is EMCS_CLUS_STATS_ENABLE.</td>
</tr>
<tr>
<td></td>
<td>EMCS_CLUS_STATS_DISABLE</td>
<td></td>
</tr>
<tr>
<td>EMCS_MIN_PCT_ATTR_SUPPORT</td>
<td>TO_CHAR( 0 &lt; numeric_exp &lt; 1)</td>
<td>Minimum support required for including an attribute in the cluster rule. The support is the percentage of the data rows assigned to a cluster that must have non-null values for the attribute. Default is 0.1.</td>
</tr>
</tbody>
</table>

### 43.4.9 DBMS_DATA_MINING — Algorithm Settings: Explicit Semantic Analysis

Explicit Semantic Analysis (ESA) is a useful technique for extracting meaningful and interpretable features.

The settings listed in the following table configure the ESA values.

#### Table 43-16  Explicit Semantic Analysis Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESAS_VALUE_THRESHOLD</td>
<td>Non-negative number</td>
<td>This setting thresholds a small value for attribute weights in the transformed build data. The default is 1e-8.</td>
</tr>
<tr>
<td>ESAS_MIN_ITEMS</td>
<td>Text input 100</td>
<td>This setting determines the minimum number of non-zero entries that need to be present in an input row. The default is 100 for text input and 0 for non-text input.</td>
</tr>
<tr>
<td></td>
<td>Non-text input is 0</td>
<td></td>
</tr>
<tr>
<td>ESAS_TOPN_FEATURES</td>
<td>A positive integer</td>
<td>This setting controls the maximum number of features per attribute. The default is 1000.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for information about Explicit Semantic Analysis.
43.4.10 DBMS_DATA_MINING — Algorithm Settings: Exponential Smoothing Models

Exponential Smoothing Models (ESM) is a useful technique for extracting meaningful and interpretable features.

The settings listed in the following table configure ESM values.
### Table 43-17 Exponential Smoothing Models Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSM_MODEL</td>
<td>It can take value in set {EXSM_SIMPLE, EXSM_SIMPLE_MULT, EXSM_HOLT, EXSM_HOLT_DMP, EXSM_MUL_TRND, EXSM_MULTRD_DMP, EXSM_SEAS_ADD, EXSM_SEAS_MUL, EXSM_HW, EXSM_HW_DMP, EXSM_HW_ADDSEA, EXSM_DHW_ADDSEA, EXSM_HWMT, EXSM_HWMT_DMP}</td>
<td>This setting specifies the model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_SIMPLE: Simple exponential smoothing model is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_SIMPLE_MULT: Simple exponential smoothing model with multiplicative error is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_HOLT: Holt linear exponential smoothing model is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_HOLT_DMP: Holt linear exponential smoothing model with damped trend is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_MUL_TRND: Exponential smoothing model with multiplicative trend is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_MULTRD_DMP: Exponential smoothing model with multiplicative damped trend is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_SEAS_ADD: Exponential smoothing with additive seasonality, but no trend, is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_SEAS_MUL: Exponential smoothing with multiplicative seasonality, but no trend, is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_HW: Holt-Winters triple exponential smoothing model, additive trend, multiplicative seasonality is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_HW_DMP: Holt-Winters multiplicative exponential smoothing model with damped trend, additive trend, multiplicative seasonality is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_HW_ADDSEA: Holt-Winters additive exponential smoothing model, additive trend, additive seasonality is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_DHW_ADDSEA: Holt-Winters additive exponential smoothing model with damped trend, additive trend, additive seasonality is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_HWMT: Holt-Winters multiplicative exponential smoothing model with multiplicative trend, multiplicative trend, multiplicative seasonality is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXSM_HWMT_DMP: Holt-Winters multiplicative exponential smoothing model with damped multiplicative trend, multiplicative trend, multiplicative seasonality is applied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is EXSM_SIMPLE.</td>
</tr>
</tbody>
</table>
### Table 43-17 (Cont.) Exponential Smoothing Models Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSM_SEASONALITY</td>
<td>positive integer &gt; 1</td>
<td>This setting specifies a positive integer value as the length of seasonal cycle. The value it takes must be larger than 1. For example, setting value 4 means that every group of four observations forms a seasonal cycle. This setting is only applicable and must be provided for models with seasonality, otherwise the model throws an error. When EXSM_INTERVAL is not set, this setting applies to the original input time series. When EXSM_INTERVAL is set, this setting applies to the accumulated time series.</td>
</tr>
<tr>
<td>EXSM_INTERVAL</td>
<td>It can take value in set {EXSM_INTERVAL_YEAR, EXSM_INTERVAL_QTR, EXSM_INTERVAL_MONTH, EXSM_INTERVAL_WEEK, EXSM_INTERVAL_DAY, EXSM_INTERVAL_HOUR, EXSM_INTERVAL_MIN, EXSM_INTERVAL_SEC}</td>
<td>This setting only applies and must be provided when the time column (case_id column) has datetime type. It specifies the spacing interval of the accumulated equally spaced time series. If the time column of input table is of datetime type and setting EXSM_INTERVAL is not provided, then the model throws an error. If the time column of input table is of oracle number type and setting EXSM_INTERVAL is provided, then the model throws an error.</td>
</tr>
<tr>
<td>EXSM_ACCUMULATE</td>
<td>It can take value in set {EXSM_ACCU_TOTAL, EXSM_ACCU_STD, EXSM_ACCU_MAX, EXSM_ACCU_MIN, EXSM_ACCU_AVG, EXSM_ACCU_MEDIAN, EXSM_ACCU_COUNT}</td>
<td>This setting only applies and must be provided when the time column has datetime type. It specifies how to generate the value of the accumulated time series from the input time series.</td>
</tr>
</tbody>
</table>
### Table 43-17 (Cont.) Exponential Smoothing Models Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSM_SETMISSING</td>
<td>It can also specify an option taking value in set (EXSM_MISS_MIN, EXSM_MISS_MAX, EXSM_MISS_AVG, EXSM_MISS_MEDIAN, EXSM_MISS_LAST, EXSM_MISS_FIRST, EXSM_MISS_PREV, EXSM_MISS_NEXT, EXSM_MISS_AUTO).</td>
<td>This setting specifies how to handle missing values, which may come from input data and/or the accumulation process of input time series. It can specify either a number or an option. If a number is specified, all the missing values are set to that number. EXSM_MISS_MIN: Replaces missing value with minimum of the accumulated time series. EXSM_MISS_MAX: Replaces missing value with maximum of the accumulated time series. EXSM_MISS_AVG: Replaces missing value with average of the accumulated time series. EXSM_MISS_MEDIAN: Replaces missing value with median of the accumulated time series. EXSM_MISS_LAST: Replaces missing value with last non-missing value of the accumulated time series. EXSM_MISS_FIRST: Replaces missing value with first non-missing value of the accumulated time series. EXSM_MISS_PREV: Replaces missing value with the previous non-missing value of the accumulated time series. EXSM_MISS_NEXT: Replaces missing value with the next non-missing value of the accumulated time series. EXSM_MISS_AUTO: EXSM model treats the input data as an irregular (non-uniformly spaced) time series. If this setting is not provided, EXSM_MISS_AUTO is the default value. In such a case, the model treats the input time series as irregular time series, viewing missing values as gaps.</td>
</tr>
<tr>
<td>EXSM_PREDICTION_STEP</td>
<td>It must be set to a number between 1-30.</td>
<td>This setting is used to specify how many steps ahead the predictions are to be made. If it is not set, the default value is 1: the model gives one-step-ahead prediction. A value greater than 30 results in an error.</td>
</tr>
<tr>
<td>EXSM_CONFIDENCE_LEVEL</td>
<td>It must be a number between 0 and 1, exclusive.</td>
<td>This setting is used to specify the desired confidence level for prediction. The lower and upper bounds of the specified confidence interval is reported. If not specified, the default confidence level is 95%.</td>
</tr>
</tbody>
</table>
Table 43-17  (Cont.) Exponential Smoothing Models Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXSM_OPT_CRITERION</td>
<td>It takes value in set ( { \text{EXSM_OPT_CRIT_LIKE,} )( \text{XSM_OPT_CRIT_MSE,} )( \text{EXSM_OPT_CRIT_AMSE,} )( \text{EXSM_OPT_CRIT_SIG,} )( \text{EXSM_OPT_CRIT_MAE} ) )}.</td>
<td>This setting is used to specify the desired optimization criterion.</td>
</tr>
<tr>
<td>EXSM_NMSE</td>
<td>positive integer</td>
<td>This setting specifies the length of the window used in computing the error metric average mean square error (AMSE).</td>
</tr>
</tbody>
</table>

See Also:

*Oracle Data Mining Concepts* for information about Exponential Smoothing Models.

43.4.11 DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Models

The settings listed in the following table configure the behavior of Generalized Linear Models.

Table 43-18  DBMS_DATA_MINING GLM Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLMS_CONF_LEVEL</td>
<td><code>TO_CHAR(0&lt;numeric_expr&lt;1)</code></td>
<td>The confidence level for coefficient confidence intervals. The default confidence level is 0.95.</td>
</tr>
<tr>
<td>GLMS_FTR_GEN_METHOD</td>
<td><code>GLMS_FTR_GEN_QUADRATIC</code> (\text{GLMS_FTR_GEN_CUBIC})</td>
<td>Whether feature generation is quadratic or cubic. When feature generation is enabled, the algorithm automatically chooses the most appropriate feature generation method based on the data.</td>
</tr>
<tr>
<td>GLMS_FTR_GENERATION</td>
<td><code>GLMS_FTR_GENERATION_Enable</code> (\text{GLMS_FTR_GENERATION_DISABLE})</td>
<td>Whether or not feature generation is enabled for GLM. By default, feature generation is not enabled. <em>Note</em>: Feature generation can only be enabled when feature selection is also enabled.</td>
</tr>
<tr>
<td>GLMS_FTR_SEL_CRIT</td>
<td><code>GLMS_FTR_SEL_AIC</code> (\text{GLMS_FTR_SEL_SBIC}) (\text{GLMS_FTR_SEL_RIC}) (\text{GLMS_FTR_SEL_ALPHA_INV})</td>
<td>Feature selection penalty criterion for adding a feature to the model. When feature selection is enabled, the algorithm automatically chooses the penalty criterion based on the data.</td>
</tr>
<tr>
<td>GLMS_FTR_SELECTION</td>
<td><code>GLMS_FTR_SELECTION_Enable</code> (\text{GLMS_FTR_SELECTION_DISABLE})</td>
<td>Whether or not feature selection is enabled for GLM. By default, feature selection is not enabled.</td>
</tr>
</tbody>
</table>
### Table 43-18  (Cont.) DBMS_DATA_MINING GLM Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLMS_MAX_FEATURES</td>
<td>TO_CHAR(0 &lt; numeric_expr &lt;= 2000)</td>
<td>When feature selection is enabled, this setting specifies the maximum number of features that can be selected for the final model. By default, the algorithm limits the number of features to ensure sufficient memory.</td>
</tr>
<tr>
<td>GLMS_PRUNE_MODEL</td>
<td>GLMS_PRUNE_MODEL_ENABLE</td>
<td>Prune enable or disable for features in the final model. Pruning is based on T-Test statistics for linear regression, or Wald Test statistics for logistic regression. Features are pruned in a loop until all features are statistically significant with respect to the full data.</td>
</tr>
<tr>
<td></td>
<td>GLMS_PRUNE_MODEL_DISABLE</td>
<td>When feature selection is enabled, the algorithm automatically performs pruning based on the data.</td>
</tr>
<tr>
<td>GLMS_REFERENCE_CLASS_NAME</td>
<td>target_value</td>
<td>The target value used as the reference class in a binary logistic regression model. Probabilities are produced for the other class. By default, the algorithm chooses the value with the highest prevalence (the most cases) for the reference class.</td>
</tr>
<tr>
<td>GLMS_RIDGE_REGRESSION</td>
<td>GLMS_RIDGE_REG_ENABLE</td>
<td>Enable or disable Ridge Regression. Ridge applies to both regression and Classification mining functions. When ridge is enabled, prediction bounds are not produced by the PREDICTION_BOUNDS SQL function.</td>
</tr>
<tr>
<td></td>
<td>GLMS_RIDGE_REG_DISABLE</td>
<td>Note: Ridge may only be enabled when feature selection is not specified, or has been explicitly disabled. If Ridge Regression and feature selection are both explicitly enabled, then an exception is raised.</td>
</tr>
<tr>
<td>GLMS_RIDGE_VALUE</td>
<td>TO_CHAR(numeric_expr &gt; 0)</td>
<td>The value of the ridge parameter. This setting is only used when the algorithm is configured to use Ridge Regression. If Ridge Regression is enabled internally by the algorithm, then the ridge parameter is determined by the algorithm.</td>
</tr>
<tr>
<td>GLMS_ROW_DIAGNOSTICS</td>
<td>GLMS_ROW_DIAG_ENABLE</td>
<td>Enable or disable row diagnostics.</td>
</tr>
<tr>
<td>GLMS_ROW_DIAG_DISABLE</td>
<td>(default).</td>
<td></td>
</tr>
<tr>
<td>GLMS_CONV_TOLERANCE</td>
<td>The range is (0, 1) non-inclusive.</td>
<td>Convergence Tolerance setting of the GLM algorithm. The default value is system-determined.</td>
</tr>
<tr>
<td>GLMS_NUM_ITERATIONS</td>
<td>Positive integer</td>
<td>Maximum number of iterations for the GLM algorithm. The default value is system-determined.</td>
</tr>
<tr>
<td>GLMS_BATCH_ROWS</td>
<td>0 or Positive integer</td>
<td>Number of rows in a batch used by the SGD solver. The value of this parameter sets the size of the batch for the SGD solver. An input of 0 triggers a data driven batch size estimate. The default is 2000.</td>
</tr>
</tbody>
</table>
Table 43-18  (Cont.) DBMS_DATA_MINING GLM Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLMS_SOLVER</td>
<td>GLMS_SOLVER_SGD (Stochastic Gradient Descent)</td>
<td>This setting allows the user to choose the GLM solver. The solver cannot be selected if GLMS_FTR_SELECTION setting is enabled. The default value is system determined.</td>
</tr>
<tr>
<td></td>
<td>GLMS_SOLVER_CHOL (Cholesky)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GLMS_SOLVER_QR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GLMS_SOLVER_LBFGS_ADM M</td>
<td></td>
</tr>
<tr>
<td>GLMS_SPARSE_SOLVER</td>
<td>GLMS_SPARSE_SOLVER_ENABLE</td>
<td>This setting allows the user to use sparse solver if it is available. The default value is GLMS_SPARSE_SOLVER_DISABLE.</td>
</tr>
<tr>
<td></td>
<td>GLMS_SPARSE_SOLVER_DISABLE (default)</td>
<td></td>
</tr>
</tbody>
</table>

Related Topics

- **DBMS_DATA_MINING — Algorithm Settings: Neural Network**
  The settings listed in the following table configure the behavior of Neural Network.

- **DBMS_DATA_MINING — Solver Settings: LBFGS**
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Models (GLM) use these settings.

- **DBMS_DATA_MINING — Solver Settings: ADMM**
  The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). Generalized Linear Models (GLM) use these settings.

- **Oracle Data Mining Concepts**

  See Also:

  *Oracle Data Mining Concepts* for information about GLM.

43.4.12 DBMS_DATA_MINING — Algorithm Settings: *k*-Means

The settings listed in the following table configure the behavior of the *k*-Means algorithm.
## Table 43-19  k-Means Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMNS_CONV_TOLERANCE</td>
<td>TO_CHAR(0&lt;numeric_expr&lt;1)</td>
<td>Minimum Convergence Tolerance for k-Means. The algorithm iterates until the minimum Convergence Tolerance is satisfied or until the maximum number of iterations, specified in KMNS_ITERATIONS, is reached. Decreasing the Convergence Tolerance produces a more accurate solution but may result in longer run times. The default Convergence Tolerance is 0.001.</td>
</tr>
<tr>
<td>KMNS_DISTANCE</td>
<td>KMNS_COSINE, KMNS_EUCLIDEAN</td>
<td>Distance function for k-Means. The default distance function is KMNS_EUCLIDEAN.</td>
</tr>
<tr>
<td>KMNS_ITERATIONS</td>
<td>TO_CHAR(positive_numeric_expr)</td>
<td>Maximum number of iterations for k-Means. The algorithm iterates until either the maximum number of iterations is reached or the minimum Convergence Tolerance, specified in KMNS_CONV_TOLERANCE, is satisfied. The default number of iterations is 20.</td>
</tr>
<tr>
<td>KMNS_MIN_PCT_ATTR_SUP</td>
<td>TO_CHAR(0&lt;=numeric_expr&lt;=1)</td>
<td>Minimum percentage of attribute values that must be non-null in order for the attribute to be included in the rule description for the cluster. If the data is sparse or includes many missing values, a minimum support that is too high can cause very short rules or even empty rules. The default minimum support is 0.1.</td>
</tr>
<tr>
<td>KMNS_NUM_BINS</td>
<td>TO_CHAR(numeric_expr&gt;0)</td>
<td>Number of bins in the attribute histogram produced by k-Means. The bin boundaries for each attribute are computed globally on the entire training data set. The binning method is equi-width. All attributes have the same number of bins with the exception of attributes with a single value that have only one bin. The default number of histogram bins is 11.</td>
</tr>
<tr>
<td>KMNS_SPLIT_CRITERION</td>
<td>KMNS_SIZE, KMNS_VARIANCE</td>
<td>Split criterion for k-Means. The split criterion controls the initialization of new k-Means clusters. The algorithm builds a binary tree and adds one new cluster at a time. When the split criterion is based on size, the new cluster is placed in the area where the largest current cluster is located. When the split criterion is based on the variance, the new cluster is placed in the area of the most spread-out cluster. The default split criterion is the KMNS_VARIANCE.</td>
</tr>
<tr>
<td>KMNS_RANDOM_SEED</td>
<td>Non-negative integer</td>
<td>This setting controls the seed of the random generator used during the k-Means initialization. It must be a non-negative integer value. The default is 0.</td>
</tr>
</tbody>
</table>
Table 43-19  (Cont.) k-Means Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMNS_DETAILS</td>
<td>KMNS_DETAILS_NONE</td>
<td>This setting determines the level of cluster detail that are computed during the build.</td>
</tr>
<tr>
<td></td>
<td>KMNS_DETAILS_HIERARCHY</td>
<td>KMNS_DETAILS_NONE: No cluster details are computed. Only the scoring information is persisted.</td>
</tr>
<tr>
<td></td>
<td>KMNS_DETAILS_ALL</td>
<td>KMNS_DETAILS_HIERARCHY: Cluster hierarchy and cluster record counts are computed. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>KMNS_DETAILS_ALL</td>
<td>KMNS_DETAILS_ALL: Cluster hierarchy, record counts, descriptive statistics (means, variances, modes, histograms, and rules) are computed.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for information about k-Means

43.4.13 DBMS_DATA_MINING — Algorithm Settings: Naive Bayes

The settings listed in the following table configure the behavior of the Naive Bayes Algorithm.

Table 43-20  Naive Bayes Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NABS_PAIRWISE_THRESHLD</td>
<td>TO_CHAR(0&lt;= numeric_expr &lt;=1)</td>
<td>Value of pairwise threshold for NB algorithm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default is 0.</td>
</tr>
<tr>
<td>NABS_SINGLETON_THRESHLD</td>
<td>TO_CHAR(0&lt;= numeric_expr &lt;=1)</td>
<td>Value of singleton threshold for NB algorithm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is 0.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for information about Naive Bayes

43.4.14 DBMS_DATA_MINING — Algorithm Settings: Neural Network

The settings listed in the following table configure the behavior of Neural Network.
### Table 43-21  DBMS_DATA_MINING Neural Network Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNET_HIDDEN_LAYERS</td>
<td>Non-negative integer</td>
<td>Defines the topology by number of hidden layers. The default value is 1.</td>
</tr>
<tr>
<td>NNET_NODES_PER_LAYER</td>
<td>A list of positive integers</td>
<td>Defines the topology by number of nodes per layer. Different layers can have different number of nodes. The value should be non-negative integers and comma separated. For example, ‘10, 20, 5’. The setting values must be consistent with NNET_HIDDEN_LAYERS. The default number of nodes per layer is the number of attributes or 50 (if the number of attributes &gt; 50).</td>
</tr>
<tr>
<td>NNET_ACTIVATIONS</td>
<td>A list of the following strings:</td>
<td>Defines the activation function for the hidden layers. For example, &quot;NNET_ACTIVATIONS_BIPOLAR_SIG&quot;, &quot;NNET_ACTIVATIONS_TANH&quot;. Different layers can have different activation functions. The default value is &quot;NNET_ACTIVATIONS_LOG_SIG&quot;. The number of activation functions must be consistent with NNET_HIDDEN_LAYERS and NNET_NODES_PER_LAYER.</td>
</tr>
<tr>
<td>NNET_WEIGHT_LOWER_BOUN</td>
<td>A real number</td>
<td>The setting specifies the lower bound of the region where weights are randomly initialized. NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND must be set together. Setting one and not setting the other raises an error. NNET_WEIGHT_LOWER_BOUND must not be greater than NNET_WEIGHT_UPPER_BOUND. The default value is (-\sqrt{6/(l_nodes+r_nodes)}). The value of (l_nodes) for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· input layer dense attributes is (l_nodes) + number of dense attributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· input layer sparse attributes is (l_nodes) - number of sparse attributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· each hidden layer is (l_nodes) + number of nodes in that hidden layer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value of (r_nodes) is the number of nodes in the layer that the weight is connecting to.</td>
</tr>
</tbody>
</table>

**Note:**

All quotes are single and two single quotes are used to escape a single quote in SQL statements.
Table 43-21  (Cont.) DBMS_DATA_MINING Neural Network Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNET_WEIGHT_UPPER_BOUND</td>
<td>A real number</td>
<td>This setting specifies the upper bound of the region where weights are initialized. It should be set in pairs with NNET_WEIGHT_LOWER_BOUND and its value must not be smaller than the value of NNET_WEIGHT_LOWER_BOUND. If not specified, the values of NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND are system determined. The default value is ( \sqrt{\frac{6}{l_{\text{nodes}}+r_{\text{nodes}}}} ). See NNET_WEIGHT_LOWER_BOUND.</td>
</tr>
<tr>
<td>NNET_ITERATIONS</td>
<td>Positive integer</td>
<td>This setting specifies the maximum number of iterations in the Neural Network algorithm. The default value is 200.</td>
</tr>
<tr>
<td>NNET_TOLERANCE</td>
<td>TO_CHAR(0&lt; numeric_expr &lt;1)</td>
<td>Defines the convergence tolerance setting of the Neural Network algorithm. The default value is 0.000001.</td>
</tr>
<tr>
<td>NNET_REGULARIZER</td>
<td>NNET_REGULARIZER_NONE</td>
<td>Regularization setting for Neural Network algorithm. If the total number of training rows is greater than 50000, the default is NNET_REGULARIZER_HELDASIDE. If the total number of training rows is less than or equal to 50000, the default is NNET_REGULARIZER_NONE.</td>
</tr>
<tr>
<td></td>
<td>NNET_REGULARIZER_L2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NNET_REGULARIZER_HELDASIDE</td>
<td></td>
</tr>
<tr>
<td>NNET_HELDASIDE_RATIO</td>
<td>0 &lt;= numeric_expr &lt;=1</td>
<td>Define the held ratio for the held-aside method. The default value is 0.25.</td>
</tr>
<tr>
<td>NNET_HELDASIDE_MAX_FAIL</td>
<td>The value must be a positive integer.</td>
<td>With NNET_REGULARIZER_HELDASIDE, the training process is stopped early if the network performance on the validation data fails to improve or remains the same for NNET_HELDASIDE_MAX_FAIL epochs in a row. The default value is 6.</td>
</tr>
<tr>
<td>NNET_REG_LAMBDA</td>
<td>TO_CHAR(numeric_expr &gt;=0)</td>
<td>Defines the L2 regularization parameter lambda. This can not be set together with NNET_REGULARIZER_HELDASIDE. The default value is 1.</td>
</tr>
</tbody>
</table>

Related Topics

- DBMS_DATA_MINING — Solver Settings: LBFGS
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Models (GLM) use these settings.

See Also:

Oracle Data Mining Concepts for information about Neural Network.
43.4.15 DBMS_DATA_MINING — Algorithm Settings: Non-Negative Matrix Factorization

The settings listed in the following table configure the behavior of the Non-Negative Matrix Factorization algorithm.

You can query the data dictionary view * MINING_MODEL_SETTINGS (using the ALL, USER, or DBA prefix) to find the setting values for a model. See Oracle Database Reference for information about * MINING_MODEL_SETTINGS.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMFS_CONV_TOLERANCE</td>
<td>TO_CHAR(0&lt; numeric_expr &lt;=0.5)</td>
<td>Convergence tolerance for NMF algorithm. Default is 0.05.</td>
</tr>
<tr>
<td>NMFS_NONNEGATIVE_SCORING</td>
<td>NMFS_NONNEG_SCORING_ENABLE</td>
<td>Whether negative numbers should be allowed in scoring results. When set to NMFS_NONNEG_SCORING_ENABLE, negative feature values will be replaced with zeros. When set to NMFS_NONNEG_SCORING_DISABLE, negative feature values will be allowed. Default is NMFS_NONNEG_SCORING_ENABLE.</td>
</tr>
<tr>
<td></td>
<td>NMFS_NONNEG_SCORING_DISABLE</td>
<td></td>
</tr>
<tr>
<td>NMFS_NUM_ITERATIONS</td>
<td>TO_CHAR(1 &lt;= numeric_expr &lt;=500)</td>
<td>Number of iterations for NMF algorithm. Default is 50.</td>
</tr>
<tr>
<td>NMFS_RANDOM_SEED</td>
<td>TO_CHAR(numeric_expr)</td>
<td>Random seed for NMF algorithm. Default is -1.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for information about NMF

43.4.16 DBMS_DATA_MINING — Algorithm Settings: O-Cluster

The settings in the table configure the behavior of the O-Cluster algorithm.
Table 43-23  O-CLuster Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCLT_SENSITIVITY</td>
<td>TO_CHAR(0 &lt;= numeric_expr &lt;= 1)</td>
<td>A fraction that specifies the peak density required for separating a new cluster. The fraction is related to the global uniform density. Default is 0.5.</td>
</tr>
</tbody>
</table>

See Also:
Oracle Data Mining Concepts for information about O-Cluster

43.4.17 DBMS_DATA_MINING — Algorithm Settings: Random Forest

These settings configure the behavior of the Random Forest algorithm. Random forest makes use of the Decision Tree settings to configure the construction of individual trees.

Table 43-24  Random Forest Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFOR_MTRY</td>
<td>a number &gt;= 0</td>
<td>Size of the random subset of columns to be considered when choosing a split at a node. For each node, the size of the pool remains the same, but the specific candidate columns change. The default is half of the columns in the model signature. The special value 0 indicates that the candidate pool includes all columns.</td>
</tr>
<tr>
<td>RFOR_NUM_TREES</td>
<td>1&lt;= a number &lt;=65535</td>
<td>Number of trees in the forest Default is 20.</td>
</tr>
<tr>
<td>RFOR_SAMPLING_RATIO</td>
<td>0&lt; a fraction&lt;=1</td>
<td>Fraction of the training data to be randomly sampled for use in the construction of an individual tree. The default is half of the number of rows in the training data.</td>
</tr>
</tbody>
</table>

Related Topics
- DBMS_DATA_MINING — Algorithm Settings: Decision Tree
  These settings configure the behavior of the Decision Tree algorithm. Note that the Decision Tree settings are also used to configure the behavior of Random Forest as it constructs each individual Decision Tree.

See Also:
Oracle Data Mining Concepts for information about Random Forest
43.4.18 DBMS_DATA_MINING — Algorithm Constants and Settings: Singular Value Decomposition

The following constant affects the behavior of the Singular Value Decomposition algorithm.

Table 43-25  Singular Value Decomposition Constant

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Constant Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVDS_MAX_NUM_FEATURES</td>
<td>2500</td>
<td>The maximum number of features supported by SVD.</td>
</tr>
</tbody>
</table>

The following settings configure the behavior of the Singular Value Decomposition algorithm.

Table 43-26  Singular Value Decomposition Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVDS_U_MATRIX_OUTPUT</td>
<td>SVDS_U_MATRIX_ENABLE</td>
<td>Indicates whether or not to persist the U Matrix produced by SVD. The U matrix in SVD has as many rows as the number of rows in the build data. To avoid creating a large model, the U matrix is persisted only when SVDS_U_MATRIX_OUTPUT is enabled. When SVDS_U_MATRIX_OUTPUT is enabled, the build data must include a case ID. If no case ID is present and the U matrix is requested, then an exception is raised. Default is SVDS_U_MATRIX_DISABLE.</td>
</tr>
<tr>
<td></td>
<td>SVDS_U_MATRIX_DISABLE</td>
<td></td>
</tr>
<tr>
<td>SVDS_SCORING_MODE</td>
<td>SVDS_SCORING_SVD</td>
<td>Whether to use SVD or PCA scoring for the model. When the build data is scored with SVD, the projections will be the same as the U matrix. When the build data is scored with PCA, the projections will be the product of the U and S matrices. Default is SVDS_SCORING_SVD.</td>
</tr>
<tr>
<td></td>
<td>SVDS_SCORING_PCA</td>
<td></td>
</tr>
</tbody>
</table>
### Table 43-26  (Cont.) Singular Value Decomposition Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| SVDS_SOLVER        | SVDS_SOLVER_TSSVD                       | This setting indicates the solver to be used for computing SVD of the data. In the case of PCA, the solver setting indicates the type of SVD solver used to compute the PCA for the data. When this setting is not specified the solver type selection is data driven. If the number of attributes is greater than 3240, then the default wide solver is used. Otherwise, the default narrow solver is selected. The following are the group of solvers:  
|                   | SVDS_SOLVER_TSEIGEN                     | - Narrow data solvers: for matrices with up to 11500 attributes (TSEIGEN) or up to 8100 attributes (TSSVD).                                                                                                      |
|                   | SVDS_SOLVER_SSVD                        | - Wide data solvers: for matrices up to 1 million attributes.                                                                                                                                               |
|                   | SVDS_SOLVER_STEIGEN                     | - Tall-Skinny SVD uses QR computation TSVD (SVDS_SOLVER_TSSVD)                                                                                                                                             |
|                   |                                        | - Tall-Skinny SVD uses eigenvalue computation, TSEIGEN (SVDS_SOLVER_TSEIGEN), is the default solver for narrow data.                                                                                          |
|                   |                                        | For wide data solvers:                                                                                                                                                                                      |
|                   |                                        | - Stochastic SVD uses QR computation SSVD (SVDS_SOLVER_SSVD), is the default solver for wide data solvers.                                                                                                                                                                  |
|                   |                                        | - Stochastic SVD uses eigenvalue computations, STEIGEN (SVDS_SOLVER_STEIGEN).                                                                                                                              |
| SVDS_TOLERANCE     | Range [0, 1]                            | This setting is used to prune features. Define the minimum value the eigenvalue of a feature as a share of the first eigenvalue to not to prune. Default value is data driven.                                           |
| SVDS_RANDOM_SEED   | Range [0 - 4,294,967,296]               | The random seed value is used for initializing the sampling matrix used by the Stochastic SVD solver. The default is 0. The SVD Solver must be set to SSVD or STEIGEN.                                                      |
| SVDS_OVER_SAMPLING | Range [1, 5000].                        | This setting is configures the number of columns in the sampling matrix used by the Stochastic SVD solver. The number of columns in this matrix is equal to the requested number of features plus the oversampling setting. The SVD Solver must be set to SSVD or STEIGEN. |
| SVDS_POWER_ITERATI ONS | Range [0, 20].                | The power iteration setting improves the accuracy of the SSVD solver. The default is 2. The SVD Solver must be set to SSVD or STEIGEN.                                                                             |

**See Also:**

*Oracle Data Mining Concepts*
43.4.19 DBMS_DATA_MINING — Algorithm Settings: Support Vector Machine

The settings listed in the following table configure the behavior of the Support Vector Machine algorithm.

Table 43-27  SVM Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVMS_COMPLEXITY_FACTOR</td>
<td>TO_CHAR(numeric_expr &gt; 0)</td>
<td>Regularization setting that balances the complexity of the model against model robustness to achieve good generalization on new data. SVM uses a data-driven approach to finding the complexity factor. Value of complexity factor for SVM algorithm (both Classification and Regression). Default value estimated from the data by the algorithm.</td>
</tr>
<tr>
<td>SVMS_CONV_TOLERANCE</td>
<td>TO_CHAR(numeric_expr &gt; 0)</td>
<td>Convergence tolerance for SVM algorithm. Default is 0.0001.</td>
</tr>
<tr>
<td>SVMS_EPSILON</td>
<td>TO_CHAR(numeric_expr &gt; 0)</td>
<td>Regularization setting for regression, similar to complexity factor. Epsilon specifies the allowable residuals, or noise, in the data. Value of epsilon factor for SVM regression. Default is 0.1.</td>
</tr>
<tr>
<td>SVMS_KERNEL_FUNCTION</td>
<td>SVMS_GAUSSIAN, SVMS_LINEAR</td>
<td>Kernel for Support Vector Machine. Linear or Gaussian. The default value is SVMS_LINEAR.</td>
</tr>
<tr>
<td>SVMS_OUTLIER_RATE</td>
<td>TO_CHAR(0 &lt; numeric_expr &lt; 1)</td>
<td>The desired rate of outliers in the training data. Valid for One-Class SVM models only (Anomaly Detection). Default is 0.01.</td>
</tr>
<tr>
<td>SVMS_STD_DEV</td>
<td>TO_CHAR(numeric_expr &gt; 0)</td>
<td>Controls the spread of the Gaussian kernel function. SVM uses a data-driven approach to find a standard deviation value that is on the same scale as distances between typical cases. Value of standard deviation for SVM algorithm. This is applicable only for Gaussian kernel. Default value estimated from the data by the algorithm.</td>
</tr>
<tr>
<td>SVMS_NUM_ITERATIONS</td>
<td>Positive integer</td>
<td>This setting sets an upper limit on the number of SVM iterations. The default is system determined because it depends on the SVM solver.</td>
</tr>
<tr>
<td>SVMS_NUM_PIVOTS</td>
<td>Range [1; 10000]</td>
<td>This setting sets an upper limit on the number of pivots used in the Incomplete Cholesky decomposition. It can be set only for non-linear kernels. The default value is 200.</td>
</tr>
<tr>
<td>SVMS_BATCH_ROWS</td>
<td>Positive integer</td>
<td>This setting applies to SVM models with linear kernel. This setting sets the size of the batch for the SGD solver. An input of 0 triggers a data driven batch size estimate. The default is 20000.</td>
</tr>
</tbody>
</table>
### Table 43-27  (Cont.) SVM Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVMS_REGULARIZER</td>
<td>SVMS_REGULARIZER_L 1</td>
<td>This setting controls the type of regularization that the SGD SVM solver uses. The setting can be used only for linear SVM models. The default is system determined because it depends on the potential model size.</td>
</tr>
<tr>
<td></td>
<td>SVMS_REGULARIZER_L 2</td>
<td></td>
</tr>
<tr>
<td>SVMS_SOLVER</td>
<td>SVMS_SOLVER_SGD (Sub-Gradient Descend)</td>
<td>This setting allows the user to choose the SVM solver. The SGD solver cannot be selected if the kernel is non-linear. The default value is system determined.</td>
</tr>
<tr>
<td></td>
<td>SVMS_SOLVER_IPM (Interior Point Method)</td>
<td></td>
</tr>
</tbody>
</table>

See Also:

*Oracle Data Mining Concepts* for information about SVM

### 43.5 DBMS_DATA_MINING — Solver Settings

Oracle Data Mining algorithms can use different solvers. Solver settings can be provided at build time in the setting table.

**Related Topics**

- **DBMS_DATA_MINING — Solver Settings: ADMM**
  The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). Generalized Linear Models (GLM) use these settings.

- **DBMS_DATA_MINING — Solver Settings: LBFGS**
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Models (GLM) use these settings.

### 43.5.1 DBMS_DATA_MINING — Solver Settings: ADMM

The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). Generalized Linear Models (GLM) use these settings.

#### Table 43-28  DBMS_DATA_MINING ADMM Settings

<table>
<thead>
<tr>
<th>Settings Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMM_CONSENSUS</td>
<td>A positive integer</td>
<td>It is a ADMM's consensus parameter. The value must be a positive number. The default value is 0.1.</td>
</tr>
<tr>
<td>ADMM ITERATIONS</td>
<td>A positive integer</td>
<td>The number of ADMM iterations. The value must be a positive integer. The default value is 50.</td>
</tr>
</tbody>
</table>
Table 43-28  (Cont.) DBMS_DATA_MINING ADMM Settings

<table>
<thead>
<tr>
<th>Settings Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMM_TOLERANCE</td>
<td>A positive integer</td>
<td>It is a tolerance parameter. The value must be a positive number. The default value is 0.0001</td>
</tr>
</tbody>
</table>

Related Topics

- *Oracle Data Mining Concepts*

See Also:

*Oracle Data Mining Concepts* for information about Neural Network

43.5.2 DBMS_DATA_MINING — Solver Settings: LBFGS

The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Models (GLM) use these settings.

Table 43-29  DBMS_DATA_MINING L-BFGS Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBFGS_GRADIENT_TOLERANCE</td>
<td>TO_CHAR (numeric_expr &gt;0)</td>
<td>Defines gradient infinity norm tolerance for L-BFGS. Default value is 1E-9.</td>
</tr>
<tr>
<td>LBFGS_HISTORY_DEPTH</td>
<td>The value must be a positive integer.</td>
<td>Defines the number of historical copies kept in L-BFGS solver. The default value is 20.</td>
</tr>
<tr>
<td>LBFGS_SCALE_HESSIAN</td>
<td>LBFGS_SCALE_HESSIAN_ENABLE</td>
<td>Defines whether to scale Hessian in L-BFGS or not. Default value is LBFGS_SCALE_HESSIAN_ENABLE.</td>
</tr>
</tbody>
</table>

See Also:

*Oracle Data Mining Concepts* for information about Neural Network

43.6 DBMS_DATA_MINING Datatypes

The *DBMS_DATA_MINING* package defines object datatypes for mining transactional data. The package also defines a type for user-specified transformations. These types are called *DM_NESTED_n*, where *n* identifies the Oracle datatype of the nested attributes.

The Data Mining object datatypes are described in the following table:
Table 43-30   DBMS_DATA_MINING Summary of Datatypes

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_NESTED_BINARY_DOUBLE</td>
<td>The name and value of a numerical attribute of type BINARY_DOUBLE.</td>
</tr>
<tr>
<td>DM_NESTED_BINARY_DOUBLES</td>
<td>A collection of DM_NESTED_BINARY_DOUBLE.</td>
</tr>
<tr>
<td>DM_NESTED_BINARY_FLOAT</td>
<td>The name and value of a numerical attribute of type BINARY_FLOAT.</td>
</tr>
<tr>
<td>DM_NESTED_BINARY_FLOATS</td>
<td>A collection of DM_NESTED_BINARY_FLOAT.</td>
</tr>
<tr>
<td>DM_NESTED_CATEGORICAL</td>
<td>The name and value of a categorical attribute of type CHAR, VARCHAR, or VARCHAR2.</td>
</tr>
<tr>
<td>DM_NESTED_CATEGORICALS</td>
<td>A collection of DM_NESTED_CATEGORICAL.</td>
</tr>
<tr>
<td>DM_NESTED_NUMERICAL</td>
<td>The name and value of a numerical attribute of type NUMBER or FLOAT.</td>
</tr>
<tr>
<td>DM_NESTED_NUMERICALS</td>
<td>A collection of DM_NESTED_NUMERICAL.</td>
</tr>
<tr>
<td>ORA_MINING_VARCHAR2_NT</td>
<td>A table of VARCHAR2(4000).</td>
</tr>
<tr>
<td>TRANSFORM_LIST</td>
<td>A list of user-specified transformations for a model. Accepted as a parameter by the CREATE_MODEL Procedure. This collection type is defined in the DBMS_DATA_MINING_TRANSFORM package.</td>
</tr>
</tbody>
</table>

For more information about mining nested data, see Oracle Data Mining User's Guide.

Note:

Starting from Oracle Database 12c Release 2, *GET_MODELDETAILS are deprecated and are replaced with Model Detail Views. See Oracle Data Mining User’s Guide.

43.6.1 Deprecated Types

This topic contains tables listing deprecated types.

The DBMS_DATA_MINING package defines object datatypes for storing information about model attributes. Most of these types are returned by the table functions GET_n, where n identifies the type of information to return. These functions take a model name as input and return the requested information as a collection of rows.

For a list of the GET functions, see "Summary of DBMS_DATA_MINING Subprograms".

All the table functions use pipelining, which causes each row of output to be materialized as it is read from model storage, without waiting for the generation of the complete table object. For more information on pipelined, parallel table functions, consult the Oracle Database PL/SQL Language Reference.
<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_CENTROID</td>
<td>The centroid of a cluster.</td>
</tr>
<tr>
<td>DM_CENTROIDS</td>
<td>A collection of DM_CENTROID. A member of DM_CLUSTER.</td>
</tr>
<tr>
<td>DM_CHILD</td>
<td>A child node of a cluster.</td>
</tr>
<tr>
<td>DM_CHILDREN</td>
<td>A collection of DM_CHILD. A member of DM_CLUSTER.</td>
</tr>
<tr>
<td>DM_CLUSTER</td>
<td>A cluster. A cluster includes DM_PREDICATES, DM_CHILDREN, DM_CENTROIDS, and DM_HISTOGRAMS. It also includes a DM_RULE.                                                                                                                                                                                                                   See also, DM_CLUSTER Fields.</td>
</tr>
<tr>
<td>DM_CONDITIONAL</td>
<td>The conditional probability of an attribute in a Naive Bayes model.</td>
</tr>
<tr>
<td>DM_CONDITIONALS</td>
<td>A collection of DM_CONDITIONAL. Returned by GET_MODEL_DETAILS_NB Function.</td>
</tr>
<tr>
<td>DM_COST_ELEMENT</td>
<td>The actual and predicted values in a cost matrix.</td>
</tr>
<tr>
<td>DM_COST_MATRIX</td>
<td>A collection of DM_COST_ELEMENT. Returned by GET_MODEL_COST_MATRIX Function.</td>
</tr>
<tr>
<td>DM_EM_COMPONENT</td>
<td>A component of an Expectation Maximization model.</td>
</tr>
<tr>
<td>DM_EM_PROJECTION</td>
<td>A projection of an Expectation Maximization model.</td>
</tr>
<tr>
<td>DM_GLM_COEFF</td>
<td>The coefficient and associated statistics of an attribute in a Generalized Linear Model.</td>
</tr>
<tr>
<td>DM_GLM_COEFF_SET</td>
<td>A collection of DM_GLM_COEFF. Returned by GET_MODEL_DETAILS_GLM Function.</td>
</tr>
<tr>
<td>DM_HISTOGRAM_BIN</td>
<td>A histogram associated with a cluster.</td>
</tr>
<tr>
<td>DM_HISTOGRAMS</td>
<td>A collection of DM_HISTOGRAM_BIN. A member of DM_CLUSTER.</td>
</tr>
<tr>
<td>DM_ITEM</td>
<td>An item in an association rule.</td>
</tr>
<tr>
<td>DM_ITEMS</td>
<td>A collection of DM_ITEM.</td>
</tr>
<tr>
<td>DM_ITEMSET</td>
<td>A collection of DM_ITEMS.</td>
</tr>
<tr>
<td>DM_ITEMSETS</td>
<td>A collection of DM_ITEMSET. Returned by GET_FREQUENT_ITEMSETS Function.</td>
</tr>
<tr>
<td>DM_MODEL_GLOBAL_DETAIL</td>
<td>High-level statistics about a model.</td>
</tr>
</tbody>
</table>
Table 43-31  (Cont.) DBMS_DATA_MINING Summary of Deprecated Datatypes

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_NB_DETAIL</td>
<td>Information about an attribute in a Naive Bayes model.</td>
</tr>
<tr>
<td>DM_NMF_ATTRIBUTE</td>
<td>An attribute in a feature of a Non-Negative Matrix Factorization model.</td>
</tr>
<tr>
<td>DM_NMF_ATTRIBUTE_SET</td>
<td>A collection of DM_NMF_ATTRIBUTE. A member of DM_NMF_FEATURE.</td>
</tr>
<tr>
<td>DM_NMF_FEATURE</td>
<td>A feature in a Non-Negative Matrix Factorization model.</td>
</tr>
<tr>
<td>DM_NMF_FEATURE_SET</td>
<td>A collection of DM_NMF_FEATURE. Returned by GET_MODEL_DETAILS_NMF Function.</td>
</tr>
<tr>
<td>DM_PREDICATE</td>
<td>Antecedent and consequent in a rule.</td>
</tr>
<tr>
<td></td>
<td>See also, DM_CLUSTER Fields.</td>
</tr>
<tr>
<td>DM_RANKED_ATTRIBUTE</td>
<td>An attribute ranked by its importance in an Attribute Importance model.</td>
</tr>
<tr>
<td>DM_RANKED_ATTRIBUTES</td>
<td>A collection of DM_RANKED_ATTRIBUTE. Returned by GET_MODEL_DETAILS_AI Function.</td>
</tr>
<tr>
<td>DM_RULE</td>
<td>A rule that defines a conditional relationship. The rule can be one of the association rules returned by GET_ASSOCIATION_RULES Function, or it can be a rule associated with a cluster in the collection of clusters returned by GET_MODEL_DETAILS_KM Function and GET_MODEL_DETAILS_OC Function.</td>
</tr>
<tr>
<td></td>
<td>See also, DM_CLUSTER Fields.</td>
</tr>
<tr>
<td>DM_RULES</td>
<td>A collection of DM_RULE. Returned by GET_ASSOCIATION_RULES Function.</td>
</tr>
<tr>
<td></td>
<td>See also, DM_CLUSTER Fields.</td>
</tr>
<tr>
<td>DM_SVD_MATRIX</td>
<td>A factorized matrix S, V, or U returned by a Singular Value Decomposition model.</td>
</tr>
<tr>
<td>DM_SVD_MATRIX_SET</td>
<td>A collection of DM_SVD_MATRIX. Returned by GET_MODEL_DETAILS_SVD Function.</td>
</tr>
<tr>
<td>DM_SVM_ATTRIBUTE</td>
<td>The name, value, and coefficient of an attribute in a Support Vector Machine model.</td>
</tr>
<tr>
<td>DM_SVM_ATTRIBUTE_SET</td>
<td>A collection of DM_SVM_ATTRIBUTE. Returned by GET_MODEL_DETAILS_SVM Function. Also a member of DM_SVM_LINEAR_COEFF.</td>
</tr>
<tr>
<td>DM_SVM_LINEAR_COEFF</td>
<td>The linear coefficient of each attribute in a Support Vector Machine model.</td>
</tr>
</tbody>
</table>
### Table 43-31  DBMS_DATA_MINING Summary of Deprecated Datatypes

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_SVM_LINEAR_COEFF_SET</td>
<td>A collection of DM_SVM_LINEAR_COEFF. Returned by GET_MODEL_DETAILS_SVM Function for an SVM model built using the linear kernel.</td>
</tr>
<tr>
<td>DM_TRANSFORM</td>
<td>The transformation and reverse transformation expressions for an attribute.</td>
</tr>
<tr>
<td>DM_TRANSFORMS</td>
<td>A collection of DM_TRANSFORM. Returned by GET_MODEL_TRANSFORMATIONS Function.</td>
</tr>
</tbody>
</table>

#### Return Values for Clustering Algorithms

The table contains description of DM_CLUSTER return value columns, nested table columns, and rows.

### Table 43-32  DM_CLUSTER Return Values for Clustering Algorithms

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_CLUSTERS</td>
<td>A set of rows of type DM_CLUSTER. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(id NUMBER, cluster_id VARCHAR2(4000), record_count NUMBER, parent NUMBER, tree_level NUMBER, dispersion NUMBER, split_predicate DM_PREDICATES, child DM_CHILDREN, centroid DM_CENTROIDS, histogram DM_HISTOGRAMS, rule DM_RULE)</td>
</tr>
<tr>
<td>DM_PREDICATE</td>
<td>The antecedent and consequent columns each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(attribute_name VARCHAR2(4000), attribute_subname VARCHAR2(4000), conditional_operator CHAR(2)/<em>=,&lt;&gt;,&lt;,,&gt;,&gt;=</em>/, attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000), attribute_support NUMBER, attribute_confidence NUMBER)</td>
</tr>
</tbody>
</table>

### DM_CLUSTER Fields

The following table describes DM_CLUSTER fields.
Table 43-33 DM_CLUSTER Fields

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Cluster identifier</td>
</tr>
<tr>
<td>cluster_id</td>
<td>The ID of a cluster in the model</td>
</tr>
<tr>
<td>record_count</td>
<td>Specifies the number of records</td>
</tr>
<tr>
<td>parent</td>
<td>Parent ID</td>
</tr>
<tr>
<td>tree_level</td>
<td>Specifies the number of splits from the root</td>
</tr>
<tr>
<td>dispersion</td>
<td>A measure used to quantify whether a set of observed occurrences are dispersed compared to a standard statistical model.</td>
</tr>
<tr>
<td>split_predicate</td>
<td>The split_predicate column of DM_CLUSTER returns a nested table of type DM_PREDICATES. Each row, of type DM_PREDICATE, has the following columns:</td>
</tr>
<tr>
<td></td>
<td>(attribute_name VARCHAR2(4000), attribute_subname VARCHAR2(4000), conditional_operator CHAR(2) / <em>=,&lt;&gt;,&lt;,&gt;,&lt;=,&gt;=</em>/, attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000), attribute_support NUMBER, attribute_confidence NUMBER)</td>
</tr>
<tr>
<td>child</td>
<td>The child column of DM_CLUSTER returns a nested table of type DM_CHILDREN. The rows, of type DM_CHILD, have a single column of type NUMBER, which contains the identifiers of each child.</td>
</tr>
<tr>
<td>centroid</td>
<td>The centroid column of DM_CLUSTER returns a nested table of type DM_CENTROIDS. The rows, of type DM_CENTROID, have the following columns:</td>
</tr>
<tr>
<td>histogram</td>
<td>The histogram column of DM_CLUSTER returns a nested table of type DM_HISTOGRAMS. The rows, of type DM_HISTOGRAM_BIN, have the following columns:</td>
</tr>
</tbody>
</table>

Note: The Expectation Maximization algorithm uses all the fields except dispersion and split_predicate.
Table 43-33  (Cont.) DM_CLUSTER Fields

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule</td>
<td>The rule column of DM_CLUSTER returns a single row of type DM_RULE. The columns are:</td>
</tr>
<tr>
<td></td>
<td>(rule_id INTEGER,</td>
</tr>
<tr>
<td></td>
<td>antecedent DM_PREDICATES,</td>
</tr>
<tr>
<td></td>
<td>consequent DM_PREDICATES,</td>
</tr>
<tr>
<td></td>
<td>rule_support NUMBER,</td>
</tr>
<tr>
<td></td>
<td>rule_confidence NUMBER,</td>
</tr>
<tr>
<td></td>
<td>rule_lift NUMBER,</td>
</tr>
<tr>
<td></td>
<td>antecedent_support NUMBER,</td>
</tr>
<tr>
<td></td>
<td>consequent_support NUMBER,</td>
</tr>
<tr>
<td></td>
<td>number_of_items INTEGER)</td>
</tr>
</tbody>
</table>

Usage Notes

- The table function pipes out rows of type DM_CLUSTER. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".
- For descriptions of predicates (DM_PREDICATE) and rules (DM_RULE), see GET_ASSOCIATION_RULES Function.

43.7 Summary of DBMS_DATA_MINING Subprograms

This table summarizes the subprograms included in the DBMS_DATA_MINING package.

The GET_ * interfaces are replaced by model views. Oracle recommends that users leverage model detail views instead. For more information, refer to “Model Detail Views” in Oracle Data Mining User’s Guide and “Static Data Dictionary Views: ALL_ALL_TABLES to ALL_OUTLINES” in Oracle Database Reference.

Table 43-34  DBMS_DATA_MINING Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_COST_MATRIX Procedure</td>
<td>Adds a cost matrix to a classification model</td>
</tr>
<tr>
<td>ADD_PARTITION Procedure</td>
<td>Adds single or multiple partitions in an existing partition model</td>
</tr>
<tr>
<td>ALTER_REVERSE_EXPRESSION Procedure</td>
<td>Changes the reverse transformation expression to an expression that you specify</td>
</tr>
<tr>
<td>APPLY Procedure</td>
<td>Applies a model to a data set (scores the data)</td>
</tr>
<tr>
<td>COMPUTE_CONFUSION_MATRIX Procedure</td>
<td>Computes the confusion matrix for a classification model</td>
</tr>
<tr>
<td>COMPUTE_CONFUSION_MATRIX _PART Procedure</td>
<td>Computes the evaluation matrix for partitioned models</td>
</tr>
<tr>
<td>COMPUTE_LIFT Procedure</td>
<td>Computes lift for a classification model</td>
</tr>
<tr>
<td>COMPUTE_LIFT_PART Procedure</td>
<td>Computers lift for partitioned models</td>
</tr>
<tr>
<td>COMPUTE_ROC Procedure</td>
<td>Computes Receiver Operating Characteristic (ROC) for a classification model</td>
</tr>
</tbody>
</table>
Table 43-34  (Cont.) DBMS_DATA_MINING Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTE_ROC_PART Procedure</td>
<td>Computes Receiver Operating Characteristic (ROC) for a partitioned model</td>
</tr>
<tr>
<td>CREATE_MODEL Procedure</td>
<td>Creates a model</td>
</tr>
<tr>
<td>CREATE_MODEL2 Procedure</td>
<td>Creates a model without extra persistent stages</td>
</tr>
<tr>
<td>Create Model Using Registration Information</td>
<td>Fetches setting information from JSON object</td>
</tr>
<tr>
<td>DROP_ALGORITHM Procedure</td>
<td>Drops the registered algorithm information.</td>
</tr>
<tr>
<td>DROP_PARTITION Procedure</td>
<td>Drops a single partition</td>
</tr>
<tr>
<td>DROP_MODEL Procedure</td>
<td>Drops a model</td>
</tr>
<tr>
<td>EXPORT_MODEL Procedure</td>
<td>Exports a model to a dump file</td>
</tr>
<tr>
<td>EXPORT_SERMODEL Procedure</td>
<td>Exports a model in a serialized format</td>
</tr>
<tr>
<td>FETCH_JSON_SCHEMA Procedure</td>
<td>Fetches and reads JSON schema from all_mining_algorithms view</td>
</tr>
<tr>
<td>GET_MODEL_COST_MATRIX Procedure</td>
<td>Returns the cost matrix for a model</td>
</tr>
<tr>
<td>IMPORT_MODEL Procedure</td>
<td>Imports a model into a user schema</td>
</tr>
<tr>
<td>IMPORT_SERMODEL Procedure</td>
<td>Imports a serialized model back into the database</td>
</tr>
<tr>
<td>JSON Schema for R Extensible Algorithm</td>
<td>Displays flexibility in creating JSON schema for R Extensible</td>
</tr>
<tr>
<td>REGISTER_ALGORITHM Procedure</td>
<td>Registers a new algorithm</td>
</tr>
<tr>
<td>RANK_APPLY Procedure</td>
<td>Ranks the predictions from the APPLY results for a classification model</td>
</tr>
<tr>
<td>REMOVE_COST_MATRIX Procedure</td>
<td>Removes a cost matrix from a model</td>
</tr>
<tr>
<td>RENAME_MODEL Procedure</td>
<td>Renames a model</td>
</tr>
</tbody>
</table>

Deprecated GET_MODEL_DETAILS

Starting from Oracle Database 12c Release 2, the following GET_MODEL_DETAILS are deprecated:

Table 43-35  Deprecated GET_MODEL_DETAILS Functions

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_ASSOCIATION_RULES Function</td>
<td>Returns the rules from an association model</td>
</tr>
<tr>
<td>GET_FREQUENT_ITEMSETS Function</td>
<td>Returns the frequent itemsets for an association model</td>
</tr>
<tr>
<td>GET_MODELDETAILS_AI Function</td>
<td>Returns details about an Attribute Importance model</td>
</tr>
<tr>
<td>GET_MODELDETAILS_EM Function</td>
<td>Returns details about an Expectation Maximization model</td>
</tr>
</tbody>
</table>
Table 43-35  (Cont.) Deprecated GET_MODEL_DETAILS Functions

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_MODEL_DETAILS_EM_COMP Function</td>
<td>Returns details about the parameters of an Expectation Maximization model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_EM_PROJ Function</td>
<td>Returns details about the projects of an Expectation Maximization model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_GLM Function</td>
<td>Returns details about a Generalized Linear Model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_GLOBAL Function</td>
<td>Returns high-level statistics about a model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_KM Function</td>
<td>Returns details about a k-Means model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_NB Function</td>
<td>Returns details about a Naive Bayes model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_NMF Function</td>
<td>Returns details about a Non-Negative Matrix Factorization model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_OC Function</td>
<td>Returns details about an O-Cluster model</td>
</tr>
<tr>
<td>GET_MODEL_SETTINGS Function</td>
<td>Returns the settings used to build the given model</td>
</tr>
<tr>
<td></td>
<td>This function is replaced with USER/ALL/DBA_MINING_MODEL_SETTINGS</td>
</tr>
<tr>
<td>GET_MODEL_SIGNATURE Function</td>
<td>Returns the list of columns from the build input table</td>
</tr>
<tr>
<td></td>
<td>This function is replaced with USER/ALL/DBA_MINING_MODEL_ATTRIBUTES</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_SVD Function</td>
<td>Returns details about a Singular Value Decomposition model</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_SVM Function</td>
<td>Returns details about a Support Vector Machine model with a linear kernel</td>
</tr>
<tr>
<td>GET_MODEL_TRANSFORMATIONS Function</td>
<td>Returns the transformations embedded in a model</td>
</tr>
<tr>
<td></td>
<td>This function is replaced with USER/ALL/DBA_MINING_MODEL_XFORMS</td>
</tr>
<tr>
<td>GET_MODEL_DETAILS_XML Function</td>
<td>Returns details about a Decision Tree model</td>
</tr>
<tr>
<td>GET_TRANSFORM_LIST Procedure</td>
<td>Converts between two different transformation specification formats</td>
</tr>
</tbody>
</table>

Related Topics

- Oracle Data Mining User’s Guide
- Oracle Database Reference

43.7.1 ADD_COST_MATRIX Procedure

The ADD_COST_MATRIX procedure associates a cost matrix table with a Classification model. The cost matrix biases the model by assigning costs or benefits to specific model outcomes.

The cost matrix is stored with the model and taken into account when the model is scored.

You can also specify a cost matrix inline when you invoke a Data Mining SQL function for scoring. To view the scoring matrix for a model, query the DM$VC prefixed model view. Refer to Model Detail View for Classification Algorithm.
To obtain the default scoring matrix for a model, query the DM$VC prefixed model view. To remove the default scoring matrix from a model, use the REMOVE_COST_MATRIX procedure. See "GET_MODEL_COST_MATRIX Function" and "REMOVE_COST_MATRIX Procedure".

See Also:

- "Biasing a Classification Model" in Oracle Data Mining Concepts for more information about costs
- Oracle Database SQL Language Reference for syntax of inline cost matrix
- Oracle Data Mining User’s Guide

Syntax

```sql
DBMS_DATA_MINING.ADD_COST_MATRIX (  
    model_name                IN VARCHAR2,  
    cost_matrix_table_name    IN VARCHAR2,  
    cost_matrix_schema_name   IN VARCHAR2 DEFAULT NULL);  
    partition_name            IN VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 43-36 ADD_COST_MATRIX Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name]model_name. If you do not specify a schema, then your own schema is assumed.</td>
</tr>
<tr>
<td>cost_matrix_table_name</td>
<td>Name of the cost matrix table (described in Table 43-37).</td>
</tr>
<tr>
<td>cost_matrix_schema_name</td>
<td>Schema of the cost matrix table. If no schema is specified, then the current schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition in a partitioned model</td>
</tr>
</tbody>
</table>

Usage Notes

1. If the model is not in your schema, then ADD_COST_MATRIX requires the ALTER ANY MINING MODEL system privilege or the ALTER object privilege for the mining model.
2. The cost matrix table must have the columns shown in Table 43-37.

**Table 43-37 Required Columns in a Cost Matrix Table**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL_TARGET_VALUE</td>
<td>Valid target data type</td>
</tr>
<tr>
<td>PREDICTED_TARGET_VALUE</td>
<td>Valid target data type</td>
</tr>
<tr>
<td>COST</td>
<td>NUMBER,FLOAT,BINARY_DOUBLE, or BINARY_FLOAT</td>
</tr>
</tbody>
</table>
3. The types of the actual and predicted target values must be the same as the type of the model target. For example, if the target of the model is `BINARY_DOUBLE`, then the actual and predicted values must be `BINARY_DOUBLE`. If the actual and predicted values are `CHAR` or `VARCHAR`, then `ADD_COST_MATRIX` treats them as `VARCHAR2` internally.

If the types do not match, or if the actual or predicted value is not a valid target value, then the `ADD_COST_MATRIX` procedure raises an error.

**Note:**

If a reverse transformation is associated with the target, then the actual and predicted values must be consistent with the target after the reverse transformation has been applied.

See “Reverse Transformations and Model Transparency” under the “About Transformation Lists” section in `DBMS_DATA_MINING_TRANSFORM Operational Notes` for more information.

4. Since a benefit can be viewed as a negative cost, you can specify a benefit for a given outcome by providing a negative number in the `costs` column of the cost matrix table.

5. All Classification algorithms can use a cost matrix for scoring. The Decision Tree algorithm can also use a cost matrix at build time. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the `CLAS_COST_TABLE_NAME` setting in the settings table for the model. See Table 43-7.

The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, use the `REMOVE_COST_MATRIX` procedure to remove the cost matrix and the `ADD_COST_MATRIX` procedure to add a new one.

6. Scoring on a partitioned model is partition-specific. Scoring cost matrices can be added to or removed from an individual partition in a partitioned model. If `PARTITION_NAME` is `NOT NULL`, then the model must be a partitioned model. The `COST_MATRIX` is added to that partition of the partitioned model.

If the `PARTITION_NAME` is `NULL`, but the model is a partitioned model, then the `COST_MATRIX` table is added to every partition in the model.

**Example**

This example creates a cost matrix table called `COSTS_NB` and adds it to a Naive Bayes model called `NB_SH_CLAS_SAMPLE`. The model has a binary target: 1 means that the customer responds to a promotion; 0 means that the customer does not respond. The cost matrix assigns a cost of .25 to misclassifications of customers who do not respond and a cost of .75 to misclassifications of customers who do respond. This
means that it is three times more costly to misclassify responders than it is to misclassify non-responders.

CREATE TABLE costs_nb (  
  actual_target_value NUMBER,  
  predicted_target_value NUMBER,  
  cost NUMBER);

INSERT INTO costs_nb values (0, 0, 0);  
INSERT INTO costs_nb values (0, 1, .25);  
INSERT INTO costs_nb values (1, 0, .75);  
INSERT INTO costs_nb values (1, 1, 0);  
COMMIT;

EXEC dbms_data_mining.add_cost_matrix('nb_sh_clas_sample', 'costs_nb');

SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
FROM mining_data_apply_v
WHERE PREDICTION(nb_sh_clas_sample COST MODEL
  USING cust_marital_status, education, household_size) = 1
GROUP BY cust_gender
ORDER BY cust_gender;

C        CNT    AVG_AGE
- ---------- ----------
F         72         39
M        555         44

43.7.2 ADD_PARTITION Procedure

ADD_PARTITION procedure supports a single or multiple partition addition to an existing partitioned model.

The ADD_PARTITION procedure derives build settings and user-defined expressions from the existing model. The target column must exist in the input data query when adding partitions to a supervised model.

Syntax

DBMS_DATA_MINING.ADD_PARTITION (  
  model_name IN VARCHAR2,  
  data_query IN CLOB,  
  add_options IN VARCHAR2 DEFAULT ERROR);

Parameters

Table 43-38 ADD_PARTITION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>data_query</td>
<td>An arbitrary SQL statement that provides data to the model build. The user must have privilege to evaluate this query.</td>
</tr>
</tbody>
</table>
Table 43-38  (Cont.) ADD_PARTITION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| add_options | Allows users to control the conditional behavior of ADD for cases where rows in the input dataset conflict with existing partitions in the model. The following are the possible values:  
  - REPLACE: Replaces the existing partition for which the conflicting keys are found.  
  - ERROR: Terminates the ADD operation without adding any partitions.  
  - IGNORE: Eliminates the rows having the conflicting keys. |

Note:
For better performance, Oracle recommends using DROP_PARTITION followed by the ADD_PARTITION instead of using the REPLACE option.

43.7.3 ALTER_REVERSE_EXPRESSION Procedure

This procedure replaces a reverse transformation expression with an expression that you specify. If the attribute does not have a reverse expression, the procedure creates one from the specified expression.

You can also use this procedure to customize the output of clustering, feature extraction, and anomaly detection models.

Syntax

```sql
DBMS_DATA_MINING.ALTER_REVERSE_EXPRESSION (  
  model_name             VARCHAR2,  
  expression             CLOB,  
  attribute_name         VARCHAR2 DEFAULT NULL,  
  attribute_subname      VARCHAR2 DEFAULT NULL);
```

Parameters

Table 43-39  ALTER_REVERSE_EXPRESSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.</td>
</tr>
<tr>
<td>expression</td>
<td>An expression to replace the reverse transformation associated with the attribute.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Name of the attribute. Specify NULL if you wish to apply expression to a cluster, feature, or One-Class SVM prediction.</td>
</tr>
<tr>
<td>attribute_subname</td>
<td>Name of the nested attribute if attribute_name is a nested column, otherwise NULL.</td>
</tr>
</tbody>
</table>
Usage Notes

1. For purposes of model transparency, Oracle Data Mining provides reverse transformations for transformations that are embedded in a model. Reverse transformations are applied to the attributes returned in model details (`GET_MODELDETAILS_*` functions) and to the scored target of predictive models.

   ![See Also:](#) "About Transformation Lists" under DBMS_DATA_MINING_TRANSFORM Operational Notes

2. If you alter the reverse transformation for the target of a model that has a cost matrix, you must specify a transformation expression that has the same type as the actual and predicted values in the cost matrix. Also, the reverse transformation that you specify must result in values that are present in the cost matrix.

   ![See Also:](#) "ADD_COST_MATRIX Procedure" and Oracle Data Mining Concepts for information about cost matrices.

3. To prevent reverse transformation of an attribute, you can specify `NULL` for the expression.

4. The reverse transformation expression can contain a reference to a PL/SQL function that returns a valid Oracle datatype. For example, you could define a function like the following for a categorical attribute named `blood_pressure` that has values 'Low', 'Medium' and 'High'.

   ```sql
   CREATE OR REPLACE FUNCTION numx(c char) RETURN NUMBER IS
   BEGIN
     CASE c WHEN 'Low' THEN RETURN 1;
     WHEN 'Medium' THEN RETURN 2;
     WHEN 'High' THEN RETURN 3;
     ELSE RETURN null;
   END CASE;
   END numx;
   
   Then you could invoke `ALTER_REVERSE_EXPRESSION` for `blood_pressure` as follows.
   
   ```sql
   EXEC dbms_data_mining.alter_reverse_expression( 'model_name', 'NUMX(blood_pressure)', 'blood_pressure' );
   ```

5. You can use `ALTER_REVERSE_EXPRESSION` to label clusters produced by clustering models and features produced by feature extraction.

   You can use `ALTER_REVERSE_EXPRESSION` to replace the zeros and ones returned by anomaly-detection models. By default, anomaly-detection models label anomalous records with 0 and all other records with 1.
Examples

1. In this example, the target (affinity_card) of the model CLASS MODEL is manipulated internally as yes or no instead of 1 or 0 but returned as 1s and 0s when scored. The ALTER_REVERSE_EXPRESSION procedure causes the target values to be returned as TRUE or FALSE.

The data sets MINING_DATA_BUILD and MINING_DATA_TEST are included with the Oracle Data Mining sample programs. See Oracle Data Mining User's Guide for information about the sample programs.

```
DECLARE
  v_xlst dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(v_xlst,'affinity_card', NULL,
    'decode(affinity_card, 1, ''yes'', ''no'')',
    'decode(affinity_card, ''yes'', 1, 0)');
  dbms_data_mining.CREATE_MODEL(
    model_name => 'CLASS_MODEL',
    mining_function => dbms_data_mining.classification,
    data_table_name => 'mining_data_build',
    case_id_column_name => 'cust_id',
    target_column_name => 'affinity_card',
    settings_table_name => NULL,
    data_schema_name => 'dmuser',
    settings_schema_name => NULL,
    xform_list => v_xlst );
END;
/
SELECT cust_income_level, occupation,
PREDICTION(CLASS_MODEL USING *) predict_response
FROM mining_data_test WHERE age = 60 AND cust_gender IN 'M'
ORDER BY cust_income_level;

<table>
<thead>
<tr>
<th>CUST_INCOME_LEVEL</th>
<th>OCCUPATION</th>
<th>PREDICT_RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Below 30,000</td>
<td>Transp.</td>
<td>1</td>
</tr>
<tr>
<td>E: 90,000 - 109,999</td>
<td>Transp.</td>
<td>1</td>
</tr>
<tr>
<td>E: 90,000 - 109,999</td>
<td>Sales</td>
<td>1</td>
</tr>
<tr>
<td>G: 130,000 - 149,999</td>
<td>Handler</td>
<td>0</td>
</tr>
<tr>
<td>G: 130,000 - 149,999</td>
<td>Crafts</td>
<td>0</td>
</tr>
<tr>
<td>H: 150,000 - 169,999</td>
<td>Prof.</td>
<td>1</td>
</tr>
<tr>
<td>J: 190,000 - 249,999</td>
<td>Prof.</td>
<td>1</td>
</tr>
<tr>
<td>J: 190,000 - 249,999</td>
<td>Sales</td>
<td>1</td>
</tr>
</tbody>
</table>

BEGIN
  dbms_data_mining.ALTER_REVERSE_EXPRESSION (model_name => 'CLASS_MODEL',
    expression => 'decode(affinity_card, ''yes'', ''TRUE'',''
    'FALSE'')',
    attribute_name => 'affinity_card');
END;
/
```
column predict_response on
column predict_response format a20
SELECT cust_income_level, occupation,
       PREDICTION(CLASS_MODEL USING *) predict_response
FROM mining_data_test WHERE age = 60 AND cust_gender IN 'M'
ORDER BY cust_income_level;

<table>
<thead>
<tr>
<th>CUST_INCOME_LEVEL</th>
<th>OCCUPATION</th>
<th>PREDICT_RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Below 30,000</td>
<td>Transp.</td>
<td>TRUE</td>
</tr>
<tr>
<td>E: 90,000 - 109,999</td>
<td>Transp.</td>
<td>TRUE</td>
</tr>
<tr>
<td>E: 90,000 - 109,999</td>
<td>Sales</td>
<td>TRUE</td>
</tr>
<tr>
<td>G: 130,000 - 149,999</td>
<td>Handler</td>
<td>FALSE</td>
</tr>
<tr>
<td>G: 130,000 - 149,999</td>
<td>Crafts</td>
<td>FALSE</td>
</tr>
<tr>
<td>H: 150,000 - 169,999</td>
<td>Prof.</td>
<td>TRUE</td>
</tr>
<tr>
<td>J: 190,000 - 249,999</td>
<td>Prof.</td>
<td>TRUE</td>
</tr>
<tr>
<td>J: 190,000 - 249,999</td>
<td>Sales</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

2. This example specifies labels for the clusters that result from the sh_clus model. The labels consist of the word "Cluster" and the internal numeric identifier for the cluster.

BEGIN
   dbms_data_mining.ALTER_REVERSE_EXPRESSION( 'sh_clus', '''Cluster ''||value');
END;
/

SELECT cust_id, cluster_id(sh_clus using *) cluster_id
FROM sh_aprep_num
WHERE cust_id < 100011
ORDER by cust_id;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>CLUSTER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>Cluster 18</td>
</tr>
<tr>
<td>100002</td>
<td>Cluster 14</td>
</tr>
<tr>
<td>100003</td>
<td>Cluster 14</td>
</tr>
<tr>
<td>100004</td>
<td>Cluster 18</td>
</tr>
<tr>
<td>100005</td>
<td>Cluster 19</td>
</tr>
<tr>
<td>100006</td>
<td>Cluster 7</td>
</tr>
<tr>
<td>100007</td>
<td>Cluster 18</td>
</tr>
<tr>
<td>100008</td>
<td>Cluster 14</td>
</tr>
<tr>
<td>100009</td>
<td>Cluster 8</td>
</tr>
<tr>
<td>100010</td>
<td>Cluster 8</td>
</tr>
</tbody>
</table>

43.7.4 APPLY Procedure

The APPLY procedure applies a mining model to the data of interest, and generates the results in a table. The APPLY procedure is also referred to as scoring.

For predictive mining functions, the APPLY procedure generates predictions in a target column. For descriptive mining functions such as Clustering, the APPLY process assigns each case to a cluster with a probability.

In Oracle Data Mining, the APPLY procedure is not applicable to Association models and Attribute Importance models.
Note:

Scoring can also be performed directly in SQL using the Data Mining functions. See

- "Data Mining Functions" in Oracle Database SQL Language Reference
- "Scoring and Deployment" in Oracle Data Mining User's Guide

Syntax

```sql
DBMS_DATA_MINING.APPLY (  
    model_name           IN VARCHAR2,  
    data_table_name      IN VARCHAR2,  
    case_id_column_name  IN VARCHAR2,  
    result_table_name    IN VARCHAR2,  
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 43-40 APPLY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form <code>[schema_name.]model_name</code>. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of table or view containing the data to be scored</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Name of the case identifier column</td>
</tr>
<tr>
<td>result_table_name</td>
<td>Name of the table in which to store apply results</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Name of the schema containing the data to be scored</td>
</tr>
</tbody>
</table>

Usage Notes

1. The data provided for APPLY must undergo the same preprocessing as the data used to create and test the model. When you use Automatic Data Preparation, the preprocessing required by the algorithm is handled for you by the model: both at build time and apply time. (See “Automatic Data Preparation”.)

2. APPLY creates a table in the user's schema to hold the results. The columns are algorithm-specific.

   The columns in the results table are listed in Table 43-41 through Table 43-45. The case ID column name in the results table will match the case ID column name provided by you. The type of the incoming case ID column is also preserved in APPLY output.
Note:

Make sure that the case ID column does not have the same name as one of the columns that will be created by APPLY. For example, when applying a Classification model, the case ID in the scoring data must not be PREDICTION or PROBABILITY (See Table 43-41).

3. The datatype for the PREDICTION, CLUSTER_ID, and FEATURE_ID output columns is influenced by any reverse expression that is embedded in the model by the user. If the user does not provide a reverse expression that alters the scored value type, then the types will conform to the descriptions in the following tables. See "ALTER_REVERSE_EXPRESSION Procedure".

4. If the model is partitioned, the result_table_name can contain results from different partitions depending on the data from the input data table. An additional column called PARTITION_NAME is added to the result table indicating the partition name that is associated with each row.

For a non-partitioned model, the behavior does not change.

Classification

The results table for Classification has the columns described in Table 43-41. If the target of the model is categorical, the PREDICTION column will have a VARCHAR2 datatype. If the target has a binary type, the PREDICTION column will have the binary type of the target.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case ID column name</td>
<td>Type of the case ID</td>
</tr>
<tr>
<td>PREDICTION</td>
<td>Type of the target</td>
</tr>
<tr>
<td>PROBABILITY</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>

Anomaly Detection

The results table for Anomaly Detection has the columns described in Table 43-42.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case ID column name</td>
<td>Type of the case ID</td>
</tr>
<tr>
<td>PREDICTION</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PROBABILITY</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>

Regression

The results table for Regression has the columns described in APPLY Procedure.
Table 43-43  APPLY Results Table for Regression

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case ID column name</td>
<td>Type of the case ID</td>
</tr>
<tr>
<td>PREDICTION</td>
<td>Type of the target</td>
</tr>
</tbody>
</table>

Clustering

Clustering is an unsupervised mining function, and hence there are no targets. The results of an APPLY procedure will contain simply the cluster identifier corresponding to a case, and the associated probability. The results table has the columns described in Table 43-44.

Table 43-44  APPLY Results Table for Clustering

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case ID column name</td>
<td>Type of the case ID</td>
</tr>
<tr>
<td>CLUSTER_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PROBABILITY</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>

Feature Extraction

Feature Extraction is also an unsupervised mining function, and hence there are no targets. The results of an APPLY procedure will contain simply the feature identifier corresponding to a case, and the associated match quality. The results table has the columns described in Table 43-45.

Table 43-45  APPLY Results Table for Feature Extraction

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case ID column name</td>
<td>Type of the case ID</td>
</tr>
<tr>
<td>FEATURE_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>MATCH_QUALITY</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>

Examples

This example applies the GLM Regression model GLMR_SH_REGR_SAMPLE to the data in the MINING_DATA_APPLY_V view. The APPLY results are output of the table REGRESSION_APPLY_RESULT.

```sql
SQL> BEGIN
    DBMS_DATA_MINING.APPLY (
        model_name     => 'glmr_sh_regr_sample',
        data_table_name     => 'mining_data_apply_v',
        case_id_column_name => 'cust_id',
        result_table_name   => 'regression_apply_result');
    END;
/```

```sql
SQL> SELECT * FROM regression_apply_result WHERE cust_id > 101485;
```
## 43.7.5 COMPUTE_CONFUSION_MATRIX Procedure

This procedure computes a confusion matrix, stores it in a table in the user's schema, and returns the model accuracy.

A confusion matrix is a test metric for classification models. It compares the predictions generated by the model with the actual target values in a set of test data. The confusion matrix lists the number of times each class was correctly predicted and the number of times it was predicted to be one of the other classes.

**COMPUTE_CONFUSION_MATRIX** accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

**See Also:**

*Oracle Data Mining Concepts* for more details about confusion matrixes and other test metrics for classification

"COMPUTE_LIFT Procedure"

"COMPUTE_ROC Procedure"
Syntax

\[
\text{DBMS\_DATA\_MINING.COMPUTE\_CONFUSION\_MATRIX}\ (\ \\
\text{accuracy} \quad \text{OUT NUMBER,} \\
\text{apply\_result\_table\_name} \quad \text{IN VARCHAR2,} \\
\text{target\_table\_name} \quad \text{IN VARCHAR2,} \\
\text{case\_id\_column\_name} \quad \text{IN VARCHAR2,} \\
\text{target\_column\_name} \quad \text{IN VARCHAR2,} \\
\text{confusion\_matrix\_table\_name} \quad \text{IN VARCHAR2,} \\
\text{score\_column\_name} \quad \text{IN VARCHAR2 DEFAULT 'PREDICTION',} \\
\text{score\_criterion\_column\_name} \quad \text{IN VARCHAR2 DEFAULT 'PROBABILITY',} \\
\text{cost\_matrix\_table\_name} \quad \text{IN VARCHAR2 DEFAULT NULL,} \\
\text{apply\_result\_schema\_name} \quad \text{IN VARCHAR2 DEFAULT NULL,} \\
\text{target\_schema\_name} \quad \text{IN VARCHAR2 DEFAULT NULL,} \\
\text{cost\_matrix\_schema\_name} \quad \text{IN VARCHAR2 DEFAULT NULL,} \\
\text{score\_criterion\_type} \quad \text{IN VARCHAR2 DEFAULT 'PROBABILITY')};
\]

Parameters

Table 43-46  COMPUTE\_CONFUSION\_MATRIX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>Output parameter containing the overall percentage accuracy of the predictions.</td>
</tr>
<tr>
<td>apply_result_table_name</td>
<td>Table containing the predictions.</td>
</tr>
<tr>
<td>target_table_name</td>
<td>Table containing the known target values from the test data.</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case ID column in the apply results table. Must match the case identifier in the targets table.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Target column in the targets table. Contains the known target values from the test data.</td>
</tr>
<tr>
<td>confusion_matrix_table_name</td>
<td>Table containing the confusion matrix. The table will be created by the procedure in the user's schema. The columns in the confusion matrix table are described in the Usage Notes.</td>
</tr>
<tr>
<td>score_column_name</td>
<td>Column containing the predictions in the apply results table. The default column name is PREDICTION, which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
<tr>
<td>score_criterion_column_name</td>
<td>Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted. The score criterion type parameter indicates whether probabilities or costs will be used for scoring. The default column name is 'PROBABILITY', which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
</tbody>
</table>

See the Usage Notes for additional information.
Table 43-46  (Cont.) COMPUTE_CONFUSION_MATRIX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost_matrix_table_name</td>
<td>(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the score_criterion_type parameter is set to 'COSTS', the costs in this table will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.</td>
</tr>
<tr>
<td>apply_result_schema_name</td>
<td>Schema of the apply results table. If null, the user's schema is assumed.</td>
</tr>
<tr>
<td>target_schema_name</td>
<td>Schema of the table containing the known targets. If null, the user's schema is assumed.</td>
</tr>
<tr>
<td>cost_matrix_schema_name</td>
<td>Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.</td>
</tr>
<tr>
<td>score_criterion_type</td>
<td>Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the score_criterion_column_name parameter. The default value of score_criterion_type is 'PROBABILITY'. To use costs as the scoring criterion, specify 'COST'. If score_criterion_type is set to 'COST' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.</td>
</tr>
</tbody>
</table>

Usage Notes

- The predictive information you pass to COMPUTE_CONFUSION_MATRIX may be generated using SQL PREDICTION functions, the DBMS_DATA_MINING.APPLY procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the confusion matrix.
- Instead of passing a cost matrix to COMPUTE_CONFUSION_MATRIX, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL PREDICTION_COST function to populate the score criterion column.
- The predictions that you pass to COMPUTE_CONFUSION_MATRIX are in a table or view specified in apply_result_table_name.

```
CREATE TABLE apply_result_table_name AS (
    case_id_column_name VARCHAR2,
    score_column_name   VARCHAR2,
    score_criterion_column_name VARCHAR2);
```

- A cost matrix must have the columns described in Table 43-47.
Table 43-47  Columns in a Cost Matrix

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual_target_value</td>
<td>Type of the target column in the build data</td>
</tr>
<tr>
<td>predicted_target_value</td>
<td>Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.</td>
</tr>
<tr>
<td>cost</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining User's Guide for valid target datatypes
Oracle Data Mining Concepts for more information about cost matrixes

- The confusion matrix created by `COMPUTE_CONFUSION_MATRIX` has the columns described in Table 43-48.

Table 43-48  Columns in a Confusion Matrix

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual_target_value</td>
<td>Type of the target column in the build data</td>
</tr>
<tr>
<td>predicted_target_value</td>
<td>Type of the predicted target in the test data. The type of the predicted target is the same as the type of the actual target unless the predicted target has an associated reverse transformation.</td>
</tr>
<tr>
<td>value</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for more information about confusion matrixes

Examples

These examples use the Naive Bayes model `nb_sh_clas_sample`, which is created by one of the Oracle Data Mining sample programs.

Compute a Confusion Matrix Based on Probabilities

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
SELECT cust_id,
       PREDICTION(nb_sh_clas_sample USING *) prediction,
       PREDICTION_PROBABILITY(nb_sh_clas_sample USING *) probability
FROM mining_data_test_v;
```
Using probabilities as the scoring criterion, you can compute the confusion matrix as follows.

```sql
DECLARE
  v_accuracy    NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
    accuracy                     => v_accuracy,
    apply_result_table_name      => 'nb_apply_results',
    target_table_name            => 'mining_data_test_v',
    case_id_column_name          => 'cust_id',
    target_column_name           => 'affinity_card',
    confusion_matrix_table_name  => 'nb_confusion_matrix',
    score_column_name            => 'PREDICTION',
    score_criterion_column_name  => 'PROBABILITY'
  );
  DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
/
```

The confusion matrix and model accuracy are shown as follows.

```sql
**** MODEL ACCURACY ****: .7847
```

```sql
SQL>SELECT * from nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
   1                0          60
    0                0        891
   1                1        286
    0                1        263
```

**Compute a Confusion Matrix Based on a Cost Matrix Table**

The confusion matrix in the previous example shows a high rate of false positives. For 263 cases, the model predicted 1 when the actual value was 0. You could use a cost matrix to minimize this type of error.

The cost matrix table `nb_cost_matrix` specifies that a false positive is 3 times more costly than a false negative.

```sql
SQL> SELECT * from nb_cost_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
    0                0          0
    0                1        .75
   1                0        .25
   1                1          0
```

This statement shows how to generate the predictions using `APPLY`.

```sql
BEGIN
  DBMS_DATA_MINING.APPLY(
    model_name          => 'nb_sh_clas_sample',
    data_table_name     => 'mining_data_test_v',
    case_id_column_name => 'cust_id',
    result_table_name   => 'nb_apply_results');
END;
/
```
This statement computes the confusion matrix using the cost matrix table. The score criterion column is named 'PROBABILITY', which is the name generated by APPLY.

```oraql
DECLARE
  v_accuracy NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (accuracy => v_accuracy,
                            apply_result_table_name => 'nb_apply_results',
                            target_table_name => 'mining_data_test_v',
                            case_id_column_name => 'cust_id',
                            target_column_name => 'affinity_card',
                            confusion_matrix_table_name => 'nb_confusion_matrix',
                            score_column_name => 'PREDICTION',
                            score_criterion_column_name => 'PROBABILITY',
                            cost_matrix_table_name => 'nb_cost_matrix',
                            apply_result_schema_name => null,
                            target_schema_name => null,
                            cost_matrix_schema_name => null,
                            score_criterion_type => 'COST');
  DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
/
```

The resulting confusion matrix shows a decrease in false positives (212 instead of 263).

**** MODEL ACCURACY ****: .798

```oraql
SQL> SELECT * FROM nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
1                      0         91
0                      0        942
1                      1        255
0                      1        212
```

**Compute a Confusion Matrix Based on Embedded Costs**

You can use the `ADD_COST_MATRIX` procedure to embed a cost matrix in a model. The embedded costs can be used instead of probabilities for scoring. This statement adds the previously-defined cost matrix to the model.

```oraql
BEGIN    DBMS_DATA_MINING.ADD_COST_MATRIX ('nb_sh_clas_sample',
                            'nb_cost_matrix');END;/
```

The following statement applies the model to the test data using the embedded costs and stores the results in a table.

```oraql
CREATE TABLE nb_apply_results AS
  SELECT cust_id,
         PREDICTION(nb_sh_clas_sample COST MODEL USING *) prediction,
         PREDICTION_COST(nb_sh_clas_sample COST MODEL USING *) cost
  FROM mining_data_test_v;
```

You can compute the confusion matrix using the embedded costs.

```oraql
DECLARE
  v_accuracy NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (accuracy => v_accuracy,
                            apply_result_table_name => 'nb_apply_results',
                            target_table_name => 'mining_data_test_v',
                            case_id_column_name => 'cust_id',
                            target_column_name => 'affinity_card',
                            confusion_matrix_table_name => 'nb_confusion_matrix',
                            score_column_name => 'PREDICTION',
                            score_criterion_column_name => 'PROBABILITY',
                            cost_matrix_table_name => 'nb_cost_matrix',
                            apply_result_schema_name => null,
                            target_schema_name => null,
                            cost_matrix_schema_name => null,
                            score_criterion_type => 'COST');
  DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
/```
The results are:

**** MODEL ACCURACY ****: .798

SQL> SELECT * FROM nb_confusion_matrix;

<table>
<thead>
<tr>
<th>ACTUAL_TARGET_VALUE</th>
<th>PREDICTED_TARGET_VALUE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>91</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>942</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>212</td>
</tr>
</tbody>
</table>

43.7.6 COMPUTE_CONFUSION_MATRIX_PART Procedure

The COMPUTE_CONFUSION_MATRIX_PART procedure computes a confusion matrix, stores it in a table in the user's schema, and returns the model accuracy.

COMPUTE_CONFUSION_MATRIX_PART provides support to computation of evaluation metrics per-partition for partitioned models. For non-partitioned models, refer to COMPUTE_CONFUSION_MATRIX Procedure.

A confusion matrix is a test metric for Classification models. It compares the predictions generated by the model with the actual target values in a set of test data. The confusion matrix lists the number of times each class was correctly predicted and the number of times it was predicted to be one of the other classes.

COMPUTE_CONFUSION_MATRIX_PART accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.
See Also:

*Oracle Data Mining Concepts* for more details about confusion matrixes and other test metrics for classification

"COMPUTE_LIFT_PART Procedure"

"COMPUTE_ROC_PART Procedure"

Syntax

```sql
DBMS_DATA_MINING.compute_confusion_matrix_part(
    accuracy                    OUT DM_NESTED_NUMERICALS,
    apply_result_table_name     IN  VARCHAR2,
    target_table_name           IN  VARCHAR2,
    case_id_column_name         IN  VARCHAR2,
    target_column_name          IN  VARCHAR2,
    confusion_matrix_table_name IN  VARCHAR2,
    score_column_name           IN  VARCHAR2 DEFAULT 'PREDICTION',
    score_criterion_column_name IN  VARCHAR2 DEFAULT 'PROBABILITY',
    score_partition_column_name IN  VARCHAR2 DEFAULT 'PARTITION_NAME',
    cost_matrix_table_name      IN  VARCHAR2 DEFAULT NULL,
    apply_result_schema_name    IN  VARCHAR2 DEFAULT NULL,
    target_schema_name          IN  VARCHAR2 DEFAULT NULL,
    cost_matrix_schema_name     IN  VARCHAR2 DEFAULT NULL,
    score_criterion_type        IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 43-49  COMPUTE_CONFUSION_MATRIX_PART Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>Output parameter containing the overall percentage accuracy of the predictions. The output argument is changed from NUMBER to DM_NESTED_NUMERICALS</td>
</tr>
<tr>
<td>apply_result_table_name</td>
<td>Table containing the predictions</td>
</tr>
<tr>
<td>target_table_name</td>
<td>Table containing the known target values from the test data</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case ID column in the apply results table. Must match the case identifier in the targets table.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Target column in the targets table. Contains the known target values from the test data</td>
</tr>
<tr>
<td>confusion_matrix_table_name</td>
<td>Table containing the confusion matrix. The table will be created by the procedure in the user's schema. The columns in the confusion matrix table are described in the Usage Notes.</td>
</tr>
<tr>
<td>score_column_name</td>
<td>Column containing the predictions in the apply results table. The default column name is PREDICTION, which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
</tbody>
</table>
Table 43-49  (Cont.) COMPUTE_CONFUSION_MATRIX_PART Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>score_criterion_column_name</td>
<td>Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, then the class with the lowest cost is predicted. The score_criterion_type parameter indicates whether probabilities or costs will be used for scoring. The default column name is PROBABILITY, which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;). See the Usage Notes for additional information.</td>
</tr>
<tr>
<td>score_partition_column_name</td>
<td>(Optional) Parameter indicating the column which contains the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.</td>
</tr>
<tr>
<td>cost_matrix_table_name</td>
<td>(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the score_criterion_type parameter is set to COSTS, the costs in this table will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.</td>
</tr>
<tr>
<td>apply_result_schema_name</td>
<td>Schema of the apply results table. If null, then the user’s schema is assumed.</td>
</tr>
<tr>
<td>target_schema_name</td>
<td>Schema of the table containing the known targets. If null, then the user’s schema is assumed.</td>
</tr>
<tr>
<td>cost_matrix_schema_name</td>
<td>Schema of the cost matrix table, if one is provided. If null, then the user’s schema is assumed.</td>
</tr>
<tr>
<td>score_criterion_type</td>
<td>Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the score_criterion_column_name parameter. The default value of score_criterion_type is PROBABILITY. To use costs as the scoring criterion, specify COST. If score_criterion_type is set to COST but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.</td>
</tr>
</tbody>
</table>

Usage Notes

- The predictive information you pass to COMPUTE_CONFUSION_MATRIX_PART may be generated using SQL PREDICTION functions, the DBMS_DATA_MINING.APPLY procedure, or
some other mechanism. As long as you pass the appropriate data, the procedure
can compute the confusion matrix.

- Instead of passing a cost matrix to COMPUTE_CONFUSION_MATRIX_PART, you can use
  a scoring cost matrix associated with the model. A scoring cost matrix can be
  embedded in the model or it can be defined dynamically when the model is
  applied. To use a scoring cost matrix, invoke the SQL PREDICTION_COST function to
  populate the score criterion column.

- The predictions that you pass to COMPUTE_CONFUSION_MATRIX_PART are in a table
  or view specified in apply_result_table_name.

  ```sql
  CREATE TABLE apply_result_table_name AS (  
    case_id_column_name VARCHAR2,  
    score_column_name VARCHAR2,  
    score_criterion_column_name VARCHAR2);  
  ```

- A cost matrix must have the columns described in Table 43-47.

  **Table 43-50  Columns in a Cost Matrix**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual_target_value</td>
<td>Type of the target column in the test data</td>
</tr>
<tr>
<td>predicted_target_value</td>
<td>Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.</td>
</tr>
<tr>
<td>cost</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>

  **See Also:**

  *Oracle Data Mining User’s Guide* for valid target datatypes
  *Oracle Data Mining Concepts* for more information about cost matrixes

- The confusion matrix created by COMPUTE_CONFUSION_MATRIX_PART has the
columns described in Table 43-48.

  **Table 43-51  Columns in a Confusion Matrix Part**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual_target_value</td>
<td>Type of the target column in the test data</td>
</tr>
<tr>
<td>predicted_target_value</td>
<td>Type of the predicted target in the test data. The type of the predicted target is the same as the type of the actual target unless the predicted target has an associated reverse transformation.</td>
</tr>
<tr>
<td>value</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>
Examples

These examples use the Naive Bayes model `nb_sh_clas_sample`, which is created by one of the Oracle Data Mining sample programs.

**Compute a Confusion Matrix Based on Probabilities**

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```sql
CREATE TABLE nb_apply_results AS
SELECT cust_id,
       PREDICTION(nb_sh_clas_sample USING *) prediction,
       PREDICTION_PROBABILITY(nb_sh_clas_sample USING *) probability
FROM mining_data_test_v;
```

Using probabilities as the scoring criterion, you can compute the confusion matrix as follows.

```sql
DECLARE
  v_accuracy NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
    accuracy                     => v_accuracy,
    apply_result_table_name      => 'nb_apply_results',
    target_table_name            => 'mining_data_test_v',
    case_id_column_name          => 'cust_id',
    target_column_name           => 'affinity_card',
    confusion_matrix_table_name  => 'nb_confusion_matrix',
    score_column_name            => 'PREDICTION',
    score_criterion_column_name  => 'PROBABILITY'
  );
  DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
/```

The confusion matrix and model accuracy are shown as follows.

```
**** MODEL ACCURACY ****: .7847
```

```sql
SELECT * FROM NB_CONFUSION_MATRIX;
```

<table>
<thead>
<tr>
<th>ACTUAL_TARGET_VALUE</th>
<th>PREDICTED_TARGET_VALUE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>891</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>286</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>263</td>
</tr>
</tbody>
</table>
```

**Compute a Confusion Matrix Based on a Cost Matrix Table**
The confusion matrix in the previous example shows a high rate of false positives. For 263 cases, the model predicted 1 when the actual value was 0. You could use a cost matrix to minimize this type of error.

The cost matrix table `nb_cost_matrix` specifies that a false positive is 3 times more costly than a false negative.

```sql
SELECT * from NB_COST_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
0                      0          0
0                      1        .75
1                      0        .25
1                      1          0
```

This statement shows how to generate the predictions using `APPLY`.

```sql
BEGIN
  DBMS_DATA_MINING.APPLY(
    model_name          => 'nb_sh_clas_sample',
    data_table_name     => 'mining_data_test_v',
    case_id_column_name => 'cust_id',
    result_table_name   => 'nb_apply_results');
END;
/
```

This statement computes the confusion matrix using the cost matrix table. The score criterion column is named 'PROBABILITY', which is the name generated by `APPLY`.

```sql
DECLARE
  v_accuracy    NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
    accuracy                     => v_accuracy,
    apply_result_table_name      => 'nb_apply_results',
    target_table_name            => 'mining_data_test_v',
    case_id_column_name          => 'cust_id',
    target_column_name           => 'affinity_card',
    confusion_matrix_table_name  => 'nb_confusion_matrix',
    score_column_name            => 'PREDICTION',
    score_criterion_column_name  => 'PROBABILITY',
    score_partition_column_name  => 'PARTITION_NAME',
    cost_matrix_table_name       => 'nb_cost_matrix',
    apply_result_schema_name     => null,
    target_schema_name           => null,
    cost_matrix_schema_name      => null,
    score_criterion_type         => 'COST');
  DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
/
```

The resulting confusion matrix shows a decrease in false positives (212 instead of 263).

```sql
**** MODEL ACCURACY ****: .798
SELECT * FROM NB_CONFUSION_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
1                      0         91
0                      0        942
```
Compute a Confusion Matrix Based on Embedded Costs

You can use the ADD_COST_MATRIX procedure to embed a cost matrix in a model. The embedded costs can be used instead of probabilities for scoring. This statement adds the previously-defined cost matrix to the model.

```
BEGIN
  DBMS_DATA_MINING.ADD_COST_MATRIX ('nb_sh_clas_sample', 'nb_cost_matrix');
END;/
```

The following statement applies the model to the test data using the embedded costs and stores the results in a table.

```
CREATE TABLE nb_apply_results AS
  SELECT cust_id,
         PREDICTION(nb_sh_clas_sample COST MODEL USING *) prediction,
         PREDICTION_COST(nb_sh_clas_sample COST MODEL USING *) cost
  FROM mining_data_test_v;
```

You can compute the confusion matrix using the embedded costs.

```
DECLARE
  v_accuracy         NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART
    (accuracy                     => v_accuracy,
     apply_result_table_name      => 'nb_apply_results',
     target_table_name            => 'mining_data_test_v',
     case_id_column_name          => 'cust_id',
     target_column_name           => 'affinity_card',
     confusion_matrix_table_name  => 'nb_confusion_matrix',
     score_column_name            => 'PREDICTION',
     score_criterion_column_name  => 'PREDICTION',
     score_partition_column_name  => 'PARTITION',
     cost_matrix_table_name       => null,
     apply_result_schema_name     => null,
     target_schema_name           => null,
     cost_matrix_schema_name      => null,
     score_criterion_type         => 'COST');
END;
/```

The results are:

```
**** MODEL ACCURACY ****: .798

SELECT * FROM NB_CONFUSION_MATRIX;
```

<table>
<thead>
<tr>
<th>ACTUAL_TARGET_VALUE</th>
<th>PREDICTED_TARGET_VALUE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>91</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>942</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>205</td>
</tr>
</tbody>
</table>
43.7.7 COMPUTE_LIFT Procedure

This procedure computes lift and stores the results in a table in the user’s schema.

Lift is a test metric for binary classification models. To compute lift, one of the target values must be designated as the positive class. COMPUTE_LIFT compares the predictions generated by the model with the actual target values in a set of test data. Lift measures the degree to which the model's predictions of the positive class are an improvement over random chance.

Lift is computed on scoring results that have been ranked by probability (or cost) and divided into quantiles. Each quantile includes the scores for the same number of cases.

COMPUTE_LIFT calculates quantile-based and cumulative statistics. The number of quantiles and the positive class are user-specified. Additionally, COMPUTE_LIFT accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs associated with the predictions

- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

See Also:

Oracle Data Mining Concepts for more details about lift and test metrics for classification

"COMPUTE_CONFUSION_MATRIX Procedure"
"COMPUTE_ROC Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_LIFT (  
  apply_result_table_name  IN VARCHAR2, 
  target_table_name       IN VARCHAR2, 
  case_id_column_name     IN VARCHAR2, 
  target_column_name      IN VARCHAR2, 
  lift_table_name         IN VARCHAR2, 
  positive_target_value   IN VARCHAR2, 
  score_column_name       IN VARCHAR2 DEFAULT 'PREDICTION',
```

score_criterion_column_name  IN VARCHAR2 DEFAULT 'PROBABILITY',
num_quantiles                IN NUMBER DEFAULT 10,
cost_matrix_table_name       IN VARCHAR2 DEFAULT NULL,
apply_result_schema_name     IN VARCHAR2 DEFAULT NULL,
target_schema_name           IN VARCHAR2 DEFAULT NULL,
cost_matrix_schema_name      IN VARCHAR2 DEFAULT NULL,
score_criterion_type         IN VARCHAR2 DEFAULT 'PROBABILITY');

Parameters

Table 43-52  COMPUTE_LIFT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_result_table_name</td>
<td>Table containing the predictions.</td>
</tr>
<tr>
<td>target_table_name</td>
<td>Table containing the known target values from the test data.</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case ID column in the apply results table. Must match the case identifier in the targets table.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Target column in the targets table. Contains the known target values from the test data.</td>
</tr>
<tr>
<td>lift_table_name</td>
<td>Table containing the lift statistics. The table will be created by the procedure in the user's schema. The columns in the lift table are described in the Usage Notes.</td>
</tr>
<tr>
<td>positive_target_value</td>
<td>The positive class. This should be the class of interest, for which you want to calculate lift. If the target column is a NUMBER, you can use the TO_CHAR() operator to provide the value as a string.</td>
</tr>
<tr>
<td>score_column_name</td>
<td>Column containing the predictions in the apply results table. The default column name is 'PREDICTION', which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
<tr>
<td>score_criterion_column_name</td>
<td>Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted. The score_criterion_type parameter indicates whether probabilities or costs will be used for scoring. The default column name is 'PROBABILITY', which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;). See the Usage Notes for additional information.</td>
</tr>
<tr>
<td>num_quantiles</td>
<td>Number of quantiles to be used in calculating lift. The default is 10.</td>
</tr>
<tr>
<td>cost_matrix_table_name</td>
<td>(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the score_criterion_type parameter is set to 'COST', the costs will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.</td>
</tr>
</tbody>
</table>
Table 43-52  (Cont.) COMPUTE_LIFT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_result_schema_name</td>
<td>Schema of the apply results table. If null, the user's schema is assumed.</td>
</tr>
<tr>
<td>target_schema_name</td>
<td>Schema of the table containing the known targets. If null, the user's schema is assumed.</td>
</tr>
<tr>
<td>cost_matrix_schema_name</td>
<td>Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.</td>
</tr>
<tr>
<td>score_criterion_type</td>
<td>Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the score_criterion_column_name parameter. The default value of score_criterion_type is 'PROBABILITY'. To use costs as the scoring criterion, specify 'COST'. If score_criterion_type is set to 'COST' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.</td>
</tr>
</tbody>
</table>

Usage Notes

• The predictive information you pass to COMPUTE_LIFT may be generated using SQL PREDICTION functions, the DBMS_DATA_MINING.APPLY procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the lift.

• Instead of passing a cost matrix to COMPUTE_LIFT, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL PREDICTION_COST function to populate the score criterion column.

• The predictions that you pass to COMPUTE_LIFT are in a table or view specified in apply_results_table_name.

CREATE TABLE apply_result_table_name AS (  
  case_id_column_name VARCHAR2,  
  score_column_name VARCHAR2,  
  score_criterion_column_name VARCHAR2);  

• A cost matrix must have the columns described in Table 43-53.

Table 43-53  Columns in a Cost Matrix

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual_target_value</td>
<td>Type of the target column in the build data</td>
</tr>
</tbody>
</table>
Table 43-53  (Cont.) Columns in a Cost Matrix

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>predicted_target_value</td>
<td>Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.</td>
</tr>
<tr>
<td>cost</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for more information about cost matrixes

- The table created by COMPUTE_LIFT has the columns described in Table 43-54

Table 43-54  Columns in a Lift Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantile_number</td>
<td>NUMBER</td>
</tr>
<tr>
<td>probability_threshold</td>
<td>NUMBER</td>
</tr>
<tr>
<td>gain_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>quantile_total_count</td>
<td>NUMBER</td>
</tr>
<tr>
<td>quantile_target_count</td>
<td>NUMBER</td>
</tr>
<tr>
<td>percent_records_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>lift_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>target_density_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>targets_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>non_targets_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>lift_quantile</td>
<td>NUMBER</td>
</tr>
<tr>
<td>target_density</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for details about the information in the lift table

- When a cost matrix is passed to COMPUTE_LIFT, the cost threshold is returned in the probability_threshold column of the lift table.

Examples

This example uses the Naive Bayes model nb_sh_clas_sample, which is created by one of the Oracle Data Mining sample programs.

The example illustrates lift based on probabilities. For examples that show computation based on costs, see "COMPUTE_CONFUSION_MATRIX Procedure".
The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```sql
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using probabilities as the scoring criterion, you can compute lift as follows.

```sql
BEGIN
    DBMS_DATA_MINING.COMPUTE_LIFT (
        apply_result_table_name            => 'nb_apply_results',
        target_table_name                  => 'mining_data_test_v',
        case_id_column_name                => 'cust_id',
        target_column_name                 => 'affinity_card',
        lift_table_name                    => 'nb_lift',
        positive_target_value              => to_char(1),
        score_column_name                  => 'PREDICTION',
        score_criterion_column_name        => 'PROBABILITY',
        num_quantiles                      => 10,
        cost_matrix_table_name             => null,
        apply_result_schema_name           => null,
        target_schema_name                 => null,
        cost_matrix_schema_name            => null,
        score_criterion_type               => 'PROBABILITY');
END;
/
```

This query displays some of the statistics from the resulting lift table.

```sql
SQL>SELECT quantile_number, probability_threshold, gain_cumulative, quantile_total_count
    FROM nb_lift;
```

<table>
<thead>
<tr>
<th>QUANTILE_NUMBER</th>
<th>PROBABILITY_THRESHOLD</th>
<th>GAIN_CUMULATIVE</th>
<th>QUANTILE_TOTAL_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.989335775</td>
<td>.15034965</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>.980534911</td>
<td>.26048951</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>.968506098</td>
<td>.374125874</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>.958975196</td>
<td>.493006993</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>.946705997</td>
<td>.587412587</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>.927454174</td>
<td>.66958042</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>.904403627</td>
<td>.748251748</td>
<td>55</td>
</tr>
<tr>
<td>8</td>
<td>.836482525</td>
<td>.839160839</td>
<td>55</td>
</tr>
<tr>
<td>10</td>
<td>.500184953</td>
<td>1</td>
<td>54</td>
</tr>
</tbody>
</table>

### 43.7.8 COMPUTE_LIFT_PART Procedure

The `COMPUTE_LIFT_PART` procedure computes Lift and stores the results in a table in the user's schema. This procedure provides support to the computation of evaluation metrics per-partition for partitioned models.

Lift is a test metric for binary Classification models. To compute Lift, one of the target values must be designated as the positive class. The `COMPUTE_LIFT_PART` compares the predictions generated by the model with the actual target values in a set of test data. Lift measures the degree to which the model's predictions of the positive class are an improvement over random chance.
Lift is computed on scoring results that have been ranked by probability (or cost) and divided into quantiles. Each quantile includes the scores for the same number of cases.

`COMPUTE_LIFT_PART` calculates quantile-based and cumulative statistics. The number of quantiles and the positive class are user-specified. Additionally, `COMPUTE_LIFT_PART` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs associated with the predictions
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

See Also:

Oracle Data Mining Concepts for more details about Lift and test metrics for classification

"COMPUTE_LIFT Procedure"
"COMPUTE_CONFUSION_MATRIX Procedure"
"COMPUTE_CONFUSION_MATRIX_PART Procedure"
"COMPUTE_ROC Procedure"
"COMPUTE_ROC_PART Procedure"

Syntax

```sql
DBMS_DATA_MINING.COMPUTE_LIFT_PART (  
  apply_result_table_name     IN VARCHAR2,
  target_table_name           IN VARCHAR2,
  case_id_column_name         IN VARCHAR2,
  target_column_name          IN VARCHAR2,
  lift_table_name             IN VARCHAR2,
  positive_target_value       IN VARCHAR2,
  score_column_name           IN VARCHAR2 DEFAULT 'PREDICTION',
  score_criterion_column_name IN VARCHAR2 DEFAULT 'PROBABILITY',
  score_partition_column_name IN VARCHAR2 DEFAULT 'PARTITION_NAME',
  num_quantiles               IN NUMBER   DEFAULT 10,
  cost_matrix_table_name      IN VARCHAR2 DEFAULT NULL,
  apply_result_schema_name    IN VARCHAR2 DEFAULT NULL,
  target_schema_name          IN VARCHAR2 DEFAULT NULL,
  cost_matrix_schema_name     IN VARCHAR2 DEFAULT NULL,
  score_criterion_type        IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 43-55  COMPUTE_LIFT_PART Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_result_table_name</td>
<td>Table containing the predictions</td>
</tr>
<tr>
<td>target_table_name</td>
<td>Table containing the known target values from the test data</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case ID column in the apply results table. Must match the case identifier in the targets table.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Target column in the targets table. Contains the known target values from the test data.</td>
</tr>
<tr>
<td>lift_table_name</td>
<td>Table containing the Lift statistics. The table will be created by the procedure in the user's schema. The columns in the Lift table are described in the Usage Notes.</td>
</tr>
<tr>
<td>positive_target_value</td>
<td>The positive class. This should be the class of interest, for which you want to calculate Lift. If the target column is a NUMBER, then you can use the TO_CHAR() operator to provide the value as a string.</td>
</tr>
<tr>
<td>score_column_name</td>
<td>Column containing the predictions in the apply results table. The default column name is PREDICTION, which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
<tr>
<td>score_criterion_column_name</td>
<td>Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, then the class with the lowest cost is predicted. The score_criterion_type parameter indicates whether probabilities or costs will be used for scoring. The default column name is PROBABILITY, which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;). See the Usage Notes for additional information.</td>
</tr>
<tr>
<td>score_partition_column_name</td>
<td>Optional parameter indicating the column containing the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.</td>
</tr>
<tr>
<td>num_quantiles</td>
<td>Number of quantiles to be used in calculating Lift. The default is 10.</td>
</tr>
<tr>
<td>cost_matrix_table_name</td>
<td>(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the score_criterion_type parameter is set to COST, then the costs will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.</td>
</tr>
</tbody>
</table>
Table 43-55  (Cont.) COMPUTE_LIFT_PART Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_result_schema_name</td>
<td>Schema of the apply results table If null, then the user’s schema is assumed.</td>
</tr>
<tr>
<td>target_schema_name</td>
<td>Schema of the table containing the known targets If null, then the user’s schema is assumed.</td>
</tr>
<tr>
<td>cost_matrix_schema_name</td>
<td>Schema of the cost matrix table, if one is provided If null, then the user’s schema is assumed.</td>
</tr>
<tr>
<td>score_criterion_type</td>
<td>Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the score_criterion_column_name parameter. The default value of score_criterion_type is PROBABILITY. To use costs as the scoring criterion, specify COST. If score_criterion_type is set to COST but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.</td>
</tr>
</tbody>
</table>

Usage Notes

- The predictive information you pass to COMPUTE_LIFT_PART may be generated using SQL PREDICTION functions, the DBMS_DATA_MINING.APPLY procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the Lift.

- Instead of passing a cost matrix to COMPUTE_LIFT_PART, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL PREDICTION_COST function to populate the score criterion column.

- The predictions that you pass to COMPUTE_LIFT_PART are in a table or view specified in apply_results_table_name.

```sql
CREATE TABLE apply_result_table_name AS (  
  case_id_column_name VARCHAR2,  
  score_column_name VARCHAR2,  
  score_criterion_column_name VARCHAR2);  
```

- A cost matrix must have the columns described in Table 43-53.

Table 43-56  Columns in a Cost Matrix

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual_target_value</td>
<td>Type of the target column in the test data</td>
</tr>
<tr>
<td>predicted_target_value</td>
<td>Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.</td>
</tr>
</tbody>
</table>
Table 43-56  (Cont.) Columns in a Cost Matrix

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for more information about cost matrix

- The table created by COMPUTE_LIFT_PART has the columns described in Table 43-54

Table 43-57  Columns in a COMPUTE_LIFT_PART Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantile_number</td>
<td>NUMBER</td>
</tr>
<tr>
<td>probability_threshold</td>
<td>NUMBER</td>
</tr>
<tr>
<td>gain_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>quantile_total_count</td>
<td>NUMBER</td>
</tr>
<tr>
<td>quantile_target_count</td>
<td>NUMBER</td>
</tr>
<tr>
<td>percent_records_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>lift_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>target_density_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>targets_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>non_targets_cumulative</td>
<td>NUMBER</td>
</tr>
<tr>
<td>lift_quantile</td>
<td>NUMBER</td>
</tr>
<tr>
<td>target_density</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for details about the information in the Lift table

- When a cost matrix is passed to COMPUTE_LIFT_PART, the cost threshold is returned in the probability_threshold column of the Lift table.

Examples

This example uses the Naive Bayes model nb_sh_clas_sample, which is created by one of the Oracle Data Mining sample programs.

The example illustrates Lift based on probabilities. For examples that show computation based on costs, see "COMPUTE_CONFUSION_MATRIX Procedure".
For a partitioned model example, see "COMPUTE_CONFUSION_MATRIX_PART Procedure".

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using probabilities as the scoring criterion, you can compute Lift as follows.

```
BEGIN
    DBMS_DATA_MINING.COMPUTE_LIFT_PART (  
        apply_result_table_name     => 'nb_apply_results',
        target_table_name           => 'mining_data_test_v',
        case_id_column_name         => 'cust_id',
        target_column_name          => 'affinity_card',
        lift_table_name             => 'nb_lift',
        positive_target_value       => to_char(1),
        score_column_name           => 'PREDICTION',
        score_criterion_column_name => 'PROBABILITY',
        score_partition_column_name => 'PARTITION_NAME',
        num_quantiles               => 10,
        cost_matrix_table_name      => null,
        apply_result_schema_name    => null,
        target_schema_name          => null,
        cost_matrix_schema_name     => null,
        score_criterion_type        => 'PROBABILITY');
END;
/
```

This query displays some of the statistics from the resulting Lift table.

```
SELECT quantile_number, probability_threshold, gain_cumulative,
    quantile_total_count
FROM nb_lift;
```

<table>
<thead>
<tr>
<th>QUANTILE_NUMBER</th>
<th>PROBABILITY_THRESHOLD</th>
<th>GAIN_CUMULATIVE</th>
<th>QUANTILE_TOTAL_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.989335775</td>
<td>.15034965</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>.980534911</td>
<td>.26048951</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>.968506098</td>
<td>.374125874</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>.958975196</td>
<td>.493006993</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>.946705997</td>
<td>.587412587</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>.927454174</td>
<td>.66958042</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>.904403627</td>
<td>.748251748</td>
<td>55</td>
</tr>
<tr>
<td>8</td>
<td>.836482525</td>
<td>.839160839</td>
<td>55</td>
</tr>
<tr>
<td>10</td>
<td>.500184953</td>
<td>1</td>
<td>54</td>
</tr>
</tbody>
</table>
43.7.9 COMPUTE_ROC Procedure

This procedure computes the receiver operating characteristic (ROC), stores the results in a table in the user's schema, and returns a measure of the model accuracy.

ROC is a test metric for binary classification models. To compute ROC, one of the target values must be designated as the positive class. COMPUTE_ROC compares the predictions generated by the model with the actual target values in a set of test data.

ROC measures the impact of changes in the probability threshold. The probability threshold is the decision point used by the model for predictions. In binary classification, the default probability threshold is 0.5. The value predicted for each case is the one with a probability greater than 50%.

ROC can be plotted as a curve on an X-Y axis. The false positive rate is placed on the X axis. The true positive rate is placed on the Y axis. A false positive is a positive prediction for a case that is negative in the test data. A true positive is a positive prediction for a case that is positive in the test data.

COMPUTE_ROC accepts two input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing probabilities
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

See Also:

Oracle Data Mining Concepts for more details about ROC and test metrics for classification

"COMPUTE_CONFUSION_MATRIX Procedure"
"COMPUTE_LIFT Procedure"

Syntax

```sql
DBMS_DATA_MINING.COMPUTE_ROC (  
    roc_area_under_curve OUT NUMBER,  
    apply_result_table_name IN VARCHAR2,  
    target_table_name IN VARCHAR2,  
    case_id_column_name IN VARCHAR2,  
    target_column_name IN VARCHAR2,  
    roc_table_name IN VARCHAR2,  
    positive_target_value IN VARCHAR2,  
    score_column_name IN VARCHAR2 DEFAULT 'PREDICTION',
);```
Parameters

Table 43-58  COMPUTE_ROC Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roc_area_under_the_curve</td>
<td>Output parameter containing the area under the ROC curve (AUC). The AUC measures the likelihood that an actual positive will be predicted as positive. The greater the AUC, the greater the flexibility of the model in accommodating trade-offs between positive and negative class predictions. AUC can be especially important when one target class is rarer or more important to identify than another.</td>
</tr>
<tr>
<td>apply_result_table_name</td>
<td>Table containing the predictions.</td>
</tr>
<tr>
<td>target_table_name</td>
<td>Table containing the known target values from the test data.</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case ID column in the apply results table. Must match the case identifier in the targets table.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Target column in the targets table. Contains the known target values from the test data.</td>
</tr>
<tr>
<td>roc_table_name</td>
<td>Table containing the ROC output. The table will be created by the procedure in the user's schema. The columns in the ROC table are described in the Usage Notes.</td>
</tr>
<tr>
<td>positive_target_value</td>
<td>The positive class. This should be the class of interest, for which you want to calculate ROC. If the target column is a NUMBER, you can use the TO_CHAR() operator to provide the value as a string.</td>
</tr>
<tr>
<td>score_column_name</td>
<td>Column containing the predictions in the apply results table. The default column name is 'PREDICTION', which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
<tr>
<td>score_criterion_column_name</td>
<td>Column containing the scoring criterion in the apply results table. Contains the probabilities that determine the predictions. The default column name is 'PROBABILITY', which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
<tr>
<td>apply_result_schema_name</td>
<td>Schema of the apply results table. If null, the user's schema is assumed.</td>
</tr>
<tr>
<td>target_schema_name</td>
<td>Schema of the table containing the known targets. If null, the user's schema is assumed.</td>
</tr>
</tbody>
</table>

Usage Notes

- The predictive information you pass to COMPUTE_ROC may be generated using SQL PREDICTION functions, the DBMS_DATA_MINING.APPLY procedure, or some other
mechanism. As long as you pass the appropriate data, the procedure can compute the receiver operating characteristic.

- The predictions that you pass to `COMPUTE_ROC` are in a table or view specified in `apply_results_table_name`.

```sql
CREATE TABLE apply_result_table_name AS (
    case_id_column_name VARCHAR2,
    score_column_name VARCHAR2,
    score_criterion_column_name VARCHAR2);
```

- The table created by `COMPUTE_ROC` has the columns shown in Table 43-59.

**Table 43-59  COMPUTE_ROC Output**

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability</td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>true_positives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>false_negatives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>false_positives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>true_negatives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>true_positive_fraction</td>
<td>NUMBER</td>
</tr>
<tr>
<td>false_positive_fraction</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

**See Also:**

*Oracle Data Mining Concepts* for details about the output of `COMPUTE_ROC`.

- ROC is typically used to determine the most desirable probability threshold. This can be done by examining the true positive fraction and the false positive fraction. The true positive fraction is the percentage of all positive cases in the test data that were correctly predicted as positive. The false positive fraction is the percentage of all negative cases in the test data that were incorrectly predicted as positive.

Given a probability threshold, the following statement returns the positive predictions in an apply result table ordered by probability.

```sql
SELECT case_id_column_name
FROM apply_result_table_name
WHERE probability > probability_threshold
ORDER BY probability DESC;
```

- There are two approaches to identifying the most desirable probability threshold. Which approach you use depends on whether or not you know the relative cost of positive versus negative class prediction errors.

If the costs are known, you can apply the relative costs to the ROC table to compute the minimum cost probability threshold. Suppose the relative cost ratio is: Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query like this.
WITH cost AS (  
SELECT probability_threshold, 20 * false_negatives + false_positives cost  
FROM ROC_table  
GROUP BY probability_threshold),  
minCost AS (  
SELECT min(cost) minCost  
FROM cost)  
SELECT max(probability_threshold) probability_threshold  
FROM cost, minCost  
WHERE cost = minCost;

If relative costs are not well known, you can simply scan the values in the ROC table (in  
sorted order) and make a determination about which of the displayed trade-offs  
(misclassified positives versus misclassified negatives) is most desirable.

SELECT * FROM ROC_table  
ORDER BY probability_threshold;

Examples

This example uses the Naive Bayes model nb_sh_clas_sample, which is created by one of  
the Oracle Data Mining sample programs.

The following statement applies the model to the test data and stores the predictions and  
probabilities in a table.

CREATE TABLE nb_apply_results AS  
SELECT cust_id, t.prediction, t.probability  
FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;

Using the predictions and the target values from the test data, you can compute ROC as  
follows.

DECLARE  
v_area_under_curve NUMBER;  
BEGIN  
DBMS_DATA_MINING.COMPUTE_ROC (  
  roc_area_under_curve => v_area_under_curve,  
  apply_result_table_name => 'nb_apply_results',  
  target_table_name => 'mining_data_test_v',  
  case_id_column_name => 'cust_id',  
  target_column_name => 'mining_data_test_v',  
  roc_table_name => 'nb_roc',  
  positive_target_value => '1',  
  score_column_name => 'PREDICTION',  
  score_criterion_column_name => 'PROBABILITY');  
DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' || ROUND(v_area_under_curve,4));
END;  
/

The resulting AUC and a selection of columns from the ROC table are shown as follows.

**** AREA UNDER ROC CURVE ****: .8212

SELECT PROBABILITY, TRUE_POSITIVE_FRACTION, FALSE_POSITIVE_FRACTION  
FROM NB_ROC;
### 43.7.10 COMPUTE_ROC_PART Procedure

The `COMPUTE_ROC_PART` procedure computes Receiver Operating Characteristic (ROC), stores the results in a table in the user's schema, and returns a measure of the model accuracy. This procedure provides support for computation of evaluation metrics per-partition for partitioned models.

ROC is a test metric for binary classification models. To compute ROC, one of the target values must be designated as the positive class. `COMPUTE_ROC_PART` compares the predictions generated by the model with the actual target values in a set of test data.

ROC measures the impact of changes in the probability threshold. The probability threshold is the decision point used by the model for predictions. In binary classification, the default probability threshold is 0.5. The value predicted for each case is the one with a probability greater than 50%.

ROC can be plotted as a curve on an x-y axis. The false positive rate is placed on the x-axis. The true positive rate is placed on the y-axis. A false positive is a positive prediction for a case that is negative in the test data. A true positive is a positive prediction for a case that is positive in the test data.

`COMPUTE_ROC_PART` accepts two input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing probabilities
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

### Table: Probability, True Positive Fraction, False Positive Fraction

<table>
<thead>
<tr>
<th>Probability</th>
<th>True Positive Fraction</th>
<th>False Positive Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.5018</td>
<td>0.826589595</td>
<td>0.227902946</td>
</tr>
<tr>
<td>0.53851</td>
<td>0.823699422</td>
<td>0.221837088</td>
</tr>
<tr>
<td>0.54991</td>
<td>0.820809249</td>
<td>0.217504333</td>
</tr>
<tr>
<td>0.55628</td>
<td>0.815028902</td>
<td>0.215771231</td>
</tr>
<tr>
<td>0.55628</td>
<td>0.812138728</td>
<td>0.215771231</td>
</tr>
<tr>
<td>0.57563</td>
<td>0.800578035</td>
<td>0.214904679</td>
</tr>
<tr>
<td>0.57563</td>
<td>0.812138728</td>
<td>0.214904679</td>
</tr>
</tbody>
</table>
Syntax

```sql
DBMS_DATA_MINING.compute_roc_part(
    roc_area_under_curve        OUT DM_NESTED_NUMERICALS,
    apply_result_table_name     IN  VARCHAR2,
    target_table_name           IN  VARCHAR2,
    case_id_column_name         IN  VARCHAR2,
    target_column_name          IN  VARCHAR2,
    roc_table_name              IN  VARCHAR2,
    positive_target_value       IN  VARCHAR2,
    score_column_name           IN  VARCHAR2 DEFAULT 'PREDICTION',
    score_criterion_column_name IN  VARCHAR2 DEFAULT 'PROBABILITY',
    score_partition_column_name IN  VARCHAR2 DEFAULT 'PARTITION_NAME',
    apply_result_schema_name    IN  VARCHAR2 DEFAULT NULL,
    target_schema_name          IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roc_area_under_the_curve</td>
<td>Output parameter containing the area under the ROC curve (AUC). The AUC measures the likelihood that an actual positive will be predicted as positive. The greater the AUC, the greater the flexibility of the model in accommodating trade-offs between positive and negative class predictions. AUC can be especially important when one target class is rarer or more important to identify than another. The output argument is changed from NUMBER to DM_NESTED_NUMERICALS.</td>
</tr>
<tr>
<td>apply_result_table_name</td>
<td>Table containing the predictions.</td>
</tr>
<tr>
<td>target_table_name</td>
<td>Table containing the known target values from the test data.</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case ID column in the apply results table. Must match the case identifier in the targets table.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Target column in the targets table. Contains the known target values from the test data.</td>
</tr>
</tbody>
</table>

See Also:

*Oracle Data Mining Concepts* for more details about ROC and test metrics for Classification

"COMPUTE_ROC Procedure"

"COMPUTE_CONFUSION_MATRIX Procedure"

"COMPUTE_LIFT_PART Procedure"

"COMPUTE_LIFT Procedure"
Table 43-60 (Cont.) COMPUTE_ROC_PART Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roc_table_name</td>
<td>Table containing the ROC output. The table will be created by the procedure in the user's schema. The columns in the ROC table are described in the Usage Notes.</td>
</tr>
<tr>
<td>positive_target_value</td>
<td>The positive class. This should be the class of interest, for which you want to calculate ROC. If the target column is a NUMBER, then you can use the TO_CHAR() operator to provide the value as a string.</td>
</tr>
<tr>
<td>score_column_name</td>
<td>Column containing the predictions in the apply results table. The default column name is PREDICTION, which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
<tr>
<td>score_criterion_column_name</td>
<td>Column containing the scoring criterion in the apply results table. Contains the probabilities that determine the predictions. The default column name is PROBABILITY, which is the default name created by the APPLY procedure (See &quot;APPLY Procedure&quot;).</td>
</tr>
<tr>
<td>score_partition_column_name</td>
<td>Optional parameter indicating the column which contains the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.</td>
</tr>
<tr>
<td>apply_result_schema_name</td>
<td>Schema of the apply results table. If null, then the user's schema is assumed.</td>
</tr>
<tr>
<td>target_schema_name</td>
<td>Schema of the table containing the known targets. If null, then the user's schema is assumed.</td>
</tr>
</tbody>
</table>

Usage Notes

- The predictive information you pass to COMPUTE_ROC_PART may be generated using SQL PREDICTION functions, the DBMS_DATA_MINING.APPLY procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the receiver operating characteristic.
- The predictions that you pass to COMPUTE_ROC_PART are in a table or view specified in apply_results_table_name.

```
CREATE TABLE apply_result_table_name AS (  
  case_id_column_name      VARCHAR2,  
  score_column_name        VARCHAR2,  
  score_criterion_column_name VARCHAR2);  
```
- The COMPUTE_ROC_PART table has the following columns:

Table 43-61 COMPUTE_ROC_PART Output

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability</td>
<td>BINARY_DOUBLE</td>
</tr>
</tbody>
</table>
Table 43-61 (Cont.) COMPUTE_ROC_PART Output

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>true_positives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>false_negatives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>false_positives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>true_negatives</td>
<td>NUMBER</td>
</tr>
<tr>
<td>true_positive_fraction</td>
<td>NUMBER</td>
</tr>
<tr>
<td>false_positive_fraction</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

See Also:

Oracle Data Mining Concepts for details about the output of COMPUTE_ROC_PART

- ROC is typically used to determine the most desirable probability threshold. This can be done by examining the true positive fraction and the false positive fraction. The true positive fraction is the percentage of all positive cases in the test data that were correctly predicted as positive. The false positive fraction is the percentage of all negative cases in the test data that were incorrectly predicted as positive.

Given a probability threshold, the following statement returns the positive predictions in an apply result table ordered by probability.

```sql
SELECT case_id_column_name
FROM apply_result_table_name
WHERE probability > probability_threshold
ORDER BY probability DESC;
```

- There are two approaches to identify the most desirable probability threshold. The approach you use depends on whether you know the relative cost of positive versus negative class prediction errors.

If the costs are known, then you can apply the relative costs to the ROC table to compute the minimum cost probability threshold. Suppose the relative cost ratio is: Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query as follows:

```sql
WITH cost AS (
  SELECT probability_threshold, 20 * false_negatives + false_positives cost
  FROM ROC_table
  GROUP BY probability_threshold),
  minCost AS (  
    SELECT min(cost) minCost
    FROM cost
  ),
  maxCost AS (  
    SELECT max(probability_threshold) probability_threshold
    FROM ROC_table
    WHERE cost = minCost;
  )

SELECT * FROM ROC_table
WHERE probability_threshold = maxCost;
```

If relative costs are not well known, then you can simply scan the values in the ROC table (in sorted order) and make a determination about which of the displayed trade-offs (misclassified positives versus misclassified negatives) is most desirable.

```sql
SELECT * FROM ROC_table
ORDER BY probability_threshold;
```
Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`, which is created by one of the Oracle Data Mining sample programs.

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```sql
CREATE TABLE nb_apply_results AS
SELECT cust_id, t.prediction, t.probability
FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using the predictions and the target values from the test data, you can compute ROC as follows.

```sql
DECLARE
  v_area_under_curve NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_ROC_PART (
    roc_area_under_curve => v_area_under_curve,
    apply_result_table_name => 'nb_apply_results',
    target_table_name => 'mining_data_test_v',
    case_id_column_name => 'cust_id',
    target_column_name => 'affinity_card',
    roc_table_name => 'nb_roc',
    positive_target_value => '1',
    score_column_name => 'PREDICTION',
    score_criterion_column_name => 'PROBABILITY',
    score_partition_column_name => 'PARTITION_NAME'
  )
  DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
    ROUND(v_area_under_curve,4));
END;
/
```

The resulting AUC and a selection of columns from the ROC table are shown as follows.

```
**** AREA UNDER ROC CURVE ****: .8212

SELECT PROBABILITY, TRUE_POSITIVE_FRACTION, FALSE_POSITIVE_FRACTION
FROM NB_ROC;
```

<table>
<thead>
<tr>
<th>PROBABILITY</th>
<th>TRUE_POSITIVE_FRACTION</th>
<th>FALSE_POSITIVE_FRACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>.00000</td>
<td>.826589595</td>
<td>.227902946</td>
</tr>
<tr>
<td>.50018</td>
<td>.825699422</td>
<td>.221837088</td>
</tr>
<tr>
<td>.54991</td>
<td>.820809249</td>
<td>.217504333</td>
</tr>
<tr>
<td>.55628</td>
<td>.817919075</td>
<td>.215771231</td>
</tr>
<tr>
<td>.55628</td>
<td>.812138728</td>
<td>.215771231</td>
</tr>
<tr>
<td>.57563</td>
<td>.800578035</td>
<td>.214904679</td>
</tr>
<tr>
<td>.57563</td>
<td>.812138728</td>
<td>.214904679</td>
</tr>
</tbody>
</table>
```

---

Chapter 43

Summary of DBMS_DATA_MINING Subprograms

---

43-90
43.7.11 CREATE_MODEL Procedure

This procedure creates a mining model with a given mining function.

Syntax

```sql
DBMS_DATA_MINING.CREATE_MODEL (
    model_name            IN VARCHAR2,
    mining_function       IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    case_id_column_name   IN VARCHAR2,
    target_column_name    IN VARCHAR2 DEFAULT NULL,
    settings_table_name   IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2 DEFAULT NULL,
    settings_schema_name  IN VARCHAR2 DEFAULT NULL,
    xform_list            IN TRANSFORM_LIST DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used. See the Usage Notes for model naming restrictions.</td>
</tr>
<tr>
<td>mining_function</td>
<td>The mining function. Values are listed in Table 43-3.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Table or view containing the build data.</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case identifier column in the build data.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>For supervised models, the target column in the build data. NULL for unsupervised models.</td>
</tr>
<tr>
<td>settings_table_name</td>
<td>Table containing build settings for the model. NULL if there is no settings table (only default settings are used).</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema hosting the build data. If NULL, then the user's schema is assumed.</td>
</tr>
<tr>
<td>settings_schema_name</td>
<td>Schema hosting the settings table. If NULL, then the user's schema is assumed.</td>
</tr>
</tbody>
</table>
Table 43-62   (Cont.) CREATE_MODEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xform_list</td>
<td>A list of transformations to be used in addition to or instead of automatic transformations, depending on the value of the PREP_AUTO setting. (See &quot;Automatic Data Preparation&quot;). The datatype of xform_list is TRANSFORM_LIST, which consists of records of type TRANSFORM_REC. Each TRANSFORM_REC specifies the transformation information for a single attribute.</td>
</tr>
</tbody>
</table>

The expression field stores a SQL expression for transforming the attribute. The reverse_expression field stores a SQL expression for reversing the transformation in model details and, if the attribute is a target, in the results of scoring. The SQL expressions are manipulated by routines in the DBMS_DATA_MINING_TRANSFORM package:

- SET_EXPRESSION Procedure
- GET_EXPRESSION Function
- SET_TRANSFORM Procedure

The attribute_spec field identifies individualized treatment for the attribute. See the Usage Notes for details. See Table 44-1 for details about the TRANSFORM_REC type.

### Usage Notes

1. You can use the attribute_spec field of the xform_list argument to identify an attribute as unstructured text or to disable Automatic Data Preparation for the attribute. The attribute_spec can have the following values:

   - **TEXT**: Indicates that the attribute contains unstructured text. The TEXT value may optionally be followed by POLICY_NAME, TOKEN_TYPE, MAX_FEATURES, and MIN_DOCUMENTS parameters.

     **TOKEN_TYPE** has the following possible values: NORMAL, STEM, THEME, SYNONYM, BIGRAM, STEM_BIGRAM. SYNONYM may be optionally followed by a thesaurus name in square brackets.

     **MAX_FEATURES** specifies the maximum number of tokens extracted from the text.

     **MIN_DOCUMENTS** specifies the minimal number of documents in which every selected token shall occur. (For information about creating a text policy, see CTX_DDL.CREATE_POLICY in Oracle Text Reference).

   Oracle Data Mining can process columns of VARCHAR2/CHAR, CLOB, BLOB, and BFILE as text. If the column is VARCHAR2 or CHAR and you do not specify TEXT, Oracle Data Mining will process the column as categorical data. If the column is CLOB, then Oracle Data Mining will process it as text by default (You do not need to specify it as TEXT). However, you do need to provide an Oracle Text
Policy in the settings). If the column is BLOB or BFILE, you must specify it as TEXT, otherwise CREATE_MODEL will return an error.

If you specify TEXT for a nested column or for an attribute in a nested column, CREATE_MODEL will return an error.

- NOPREP: Disables ADP for the attribute. When ADP is OFF, the NOPREP value is ignored.

You can specify NOPREP for a nested column, but not for an attribute in a nested column. If you specify NOPREP for an attribute in a nested column when ADP is on, CREATE_MODEL will return an error.

2. You can obtain information about a model by querying the Data Dictionary views.

   ALL/USER/DBA_MINING_MODELS
   ALL/USER/DBA_MINING_MODEL_ATTRIBUTES
   ALL/USER/DBA_MINING_MODEL_SETTINGS
   ALL/USER/DBA_MINING_MODEL.Views
   ALL/USER/DBA_MINING_MODEL_PARTITIONS
   ALL/USER/DBA_MINING_MODEL_XFORMS

You can obtain information about model attributes by querying the model details through model views. Refer to Oracle Data Mining User's Guide.

3. The naming rules for models are more restrictive than the naming rules for most database schema objects. A model name must satisfy the following additional requirements:

   - It must be 123 or fewer characters long.
   - It must be a nonquoted identifier. Oracle requires that nonquoted identifiers contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#); the initial character must be alphabetic. Oracle strongly discourages the use of the dollar sign and pound sign in nonquoted literals.

   Naming requirements for schema objects are fully documented in Oracle Database SQL Language Reference.

4. To build a partitioned model, you must provide additional settings.

   The setting for partitioning columns are as follows:

   INSERT INTO settings_table VALUES ('ODMS_PARTITION_COLUMNS', 'GENDER, AGE');

   To set user-defined partition number for a model, the setting is as follows:

   INSERT INTO settings_table VALUES ('ODMS_MAX_PARTITIONS', '10');

   The default value for maximum number of partitions is 1000.

5. By passing an xform_list to CREATE_MODEL, you can specify a list of transformations to be performed on the input data. If the PREP_AUTO setting is ON, the transformations are used in addition to the automatic transformations. If the PREP_AUTO setting is OFF, the specified transformations are the only ones implemented by the model. In both cases, transformation definitions are embedded in the model and executed automatically whenever the model is applied. See "Automatic Data Preparation". Other transforms that
can be specified with xform_list include FORCE_IN. Refer to Oracle Data Mining User's Guide.

Examples

The first example builds a Classification model using the Support Vector Machine algorithm.

-- Create the settings table
CREATE TABLE svm_model_settings (  
    setting_name VARCHAR2(30),  
    setting_value VARCHAR2(30));

-- Populate the settings table
-- Specify SVM. By default, Naive Bayes is used for classification.
-- Specify ADP. By default, ADP is not used.
BEGIN
    INSERT INTO svm_model_settings (setting_name, setting_value) VALUES  
        (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
    INSERT INTO svm_model_settings (setting_name, setting_value) VALUES  
        (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
    COMMIT;
END;
/

-- Create the model using the specified settings
BEGIN
    DBMS_DATA_MINING.CREATE_MODEL(
        model_name => 'svm_model',
        mining_function => dbms_data_mining.classification,
        data_table_name => 'mining_data_build_v',
        case_id_column_name => 'cust_id',
        target_column_name  => 'affinity_card',
        settings_table_name => 'svm_model_settings');
END;
/

You can display the model settings with the following query:

SELECT * FROM user_mining_model_settings  
WHERE model_name IN 'SVM_MODEL';

MODEL_NAME SETTING_NAME            SETTING_VALUE                  SETTING
------------- ----------------------  -----------------------------  -------
SVM_MODEL     ALGO_NAME               ALGO_SUPPORT VECTOR MACHINES  INPUT
SVM_MODEL     SVMS_STD_DEV            3.004524                      DEFAULT
SVM_MODEL     PREP_AUTO               ON                            INPUT
SVM_MODEL     SVMS_COMPLEXITY_FACTOR  1.887389                      DEFAULT
SVM_MODEL     SVMS_KERNEL_FUNCTION    SVMS_LINEAR                      DEFAULT
SVM_MODEL     SVMS_CONV_TOLERANCE     .001                          DEFAULT

The following is an example of querying a model view instead of the older GEL_MODEL_DETAILS_SVM routine.

SELECT target_value, attribute_name, attribute_value, coefficient  
FROM DM$VLSVM_MODEL;
The second example creates an Anomaly Detection model. Anomaly Detection uses SVM Classification without a target. This example uses the same settings table created for the SVM Classification model in the first example.

```
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name        => 'anomaly_detect_model',
    mining_function   => dbms_data_mining.classification,
    data_table_name   => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name => null,
    settings_table_name => 'svm_model_settings');
END;
/
```

This query shows that the models created in these examples are the only ones in your schema.

```
SELECT model_name, mining_function, algorithm FROM user_mining_models;
```

```
MODEL_NAME              MINING_FUNCTION      ALGORITHM
----------------------  -------------------- ------------------------------
SVM_MODEL               CLASSIFICATION       SUPPORT_VECTOR_MACHINES
ANOMALY_DETECT_MODEL    CLASSIFICATION       SUPPORT_VECTOR_MACHINES
```

This query shows that only the SVM Classification model has a target.

```
SELECT model_name, attribute_name, attribute_type, target
FROM user_mining_model_attributes
WHERE target = 'YES';
```

```
MODEL_NAME          ATTRIBUTE_NAME   ATTRIBUTE_TYPE     TARGET
------------------  ---------------  -----------------  ------
SVM_MODEL           AFFINITY_CARD    CATEGORICAL         YES
```

43.7.12 CREATE_MODEL2 Procedure

The `CREATE_MODEL2` procedure is an alternate procedure to the `CREATE_MODEL` procedure, which enables creating a model without extra persistence stages. In the `CREATE_MODEL` procedure, the input is a table or a view and if such an object is not already present, the user must create it. By using the `CREATE_MODEL2` procedure, the user does not need to create such transient database objects.

**Syntax**

```
DBMS_DATA_MINING.CREATE_MODEL2 (  
  model_name IN VARCHAR2,  
  mining_function IN VARCHAR2,  
  data_query IN CLOB,  
  set_list IN SETTING_LIST,  
  case_id_column_name IN VARCHAR2 DEFAULT NULL,  
  target_column_name IN VARCHAR2 DEFAULT NULL,  
  xform_list IN TRANSFORM_LIST DEFAULT NULL);
```
Parameters

Table 43-63  CREATE_MODEL2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then the current schema is used. See the Usage Notes, CREATE_MODEL Procedure for model naming restrictions.</td>
</tr>
<tr>
<td>mining_function</td>
<td>The mining function. Values are listed in DBMS_DATA_MINING — Mining Function Settings.</td>
</tr>
<tr>
<td>data_query</td>
<td>A query which provides training data for building the model.</td>
</tr>
<tr>
<td>set_list</td>
<td>Specifies the SETTING LIST SETTING_LIST is a table of CLOB index by VARCHAR2(30); Where the index is the setting name and the CLOB is the setting value for that name.</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Case identifier column in the build data.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>For supervised models, the target column in the build data. NULL for unsupervised models.</td>
</tr>
<tr>
<td>xform_list</td>
<td>Refer to CREATE_MODEL Procedure.</td>
</tr>
</tbody>
</table>

Usage Notes

Refer to CREATE_MODEL Procedure for Usage Notes.

Examples

The following example uses the Support Vector Machine algorithm.

```sql
declare
  v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
  v_setlst(dbms_data_mining.algo_name) :=
  dbms_data_mining.algo_support_vector_machines;
  v_setlst(dbms_data_mining.prep_auto) :=
  dbms_data_mining.prep_auto_on;
  DBMS_DATA_MINING.CREATE_MODEL2(
    model_name => 'svm_model',
    mining_function => dbms_data_mining.classification,
    data_query => 'select * from mining_data_build_v',
    data_table_name => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name => 'affinity_card',
    set_list => v_setlst,
    case_id_column_name => 'cust_id',
    target_column_name => 'affinity_card');
END;
/```
43.7.13 Create Model Using Registration Information

Create model function fetches the setting information from JSON object.

Usage Notes

If an algorithm is registered, user can create model using the registered algorithm name. Since all R scripts and default setting values are already registered, providing the value through the setting table is not necessary. This makes the use of this algorithm easier.

Examples

The first example builds a Classification model using the GLM algorithm.

```sql
CREATE TABLE GLM_RDEMO_SETTINGS_CL (   setting_name   VARCHAR2(30),   setting_value   VARCHAR2(4000));
BEGIN   INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES   (   'ALGO_EXTENSIBLE_LANG', 'R');   INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES   (dbms_data_mining.ralg_registration_algo_name, 't1');   INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES   (dbms_data_mining.odms_formula,   'AGE + EDUCATION + HOUSEHOLD_SIZE + OCCUPATION');   INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES   (   'RALG_PARAMETER_FAMILY',   'binomial(logit)');
END;
/
BEGIN
DBMS_DATA_MINING.CREATE_MODEL(
   model_name                    =>    'GLM_RDEMO_CLASSIFICATION',
   mining_function               =>     dbms_data_mining.classification,
   data_table_name               =>    'mining_data_build_v',
   case_id_column_name           =>    'CUST_ID',
   target_column_name            =>    'AFFINITY_CARD',
   settings_table_name           =>    'GLM_RDEMO_SETTINGS_CL');
END;
/
```

43.7.14 DROP_ALGORITHM Procedure

This function is used to drop the registered algorithm information.

Syntax

```sql
DBMS_DATA_MINING.DROP_ALGORITHM (algorithm_name IN VARCHAR2(30),
cascade IN BOOLEAN default FALSE)
```
Parameters

Table 43-64  DROP_ALGORITHM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm_name</td>
<td>Name of the algorithm.</td>
</tr>
<tr>
<td>cascade</td>
<td>If the cascade option is TRUE, all the models with this algorithms are forced to drop. There after, the algorithm is dropped. The default value is FALSE.</td>
</tr>
</tbody>
</table>

Usage Note

- To drop a mining model, you must be the owner or you must have the RQADMIN privilege. See Oracle Data Mining User’s Guide for information about privileges for data mining.
- Make sure a model is not built on the algorithm, then drop the algorithm from the system table.
- If you try to drop an algorithm with a model built on it, then an error is displayed.

43.7.15 DROP_PARTITION Procedure

The DROP_PARTITION procedure drops a single partition that is specified in the parameter partition_name.

Syntax

```sql
DBMS_DATA_MINING.DROP_PARTITION (
    model_name IN VARCHAR2,
    partition_name IN VARCHAR2);
```

Parameters

Table 43-65  DROP_PARTITION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the mining model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition that must be dropped.</td>
</tr>
</tbody>
</table>

43.7.16 DROP_MODEL Procedure

This procedure deletes the specified mining model.

Syntax

```sql
DBMS_DATA_MINING.DROP_MODEL (model_name IN VARCHAR2,
    force IN BOOLEAN DEFAULT FALSE);
```
Parameters

Table 43-66  DROP_MODEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the mining model in the form [schema_name].model_name. If you do not specify a schema, your own schema is used.</td>
</tr>
<tr>
<td>force</td>
<td>Forces the mining model to be dropped even if it is invalid. A mining model may be invalid if a serious system error interrupted the model build process.</td>
</tr>
</tbody>
</table>

Usage Note

To drop a mining model, you must be the owner or you must have the DROP ANY MINING MODEL privilege. See Oracle Data Mining User's Guide for information about privileges for data mining.

Example

You can use the following command to delete a valid mining model named nb_sh_clas_sample that exists in your schema.

```
BEGIN
    DBMS_DATA_MINING.DROP_MODEL(model_name => 'nb_sh_clas_sample');
END;
/
```

43.7.17 EXPORT_MODEL Procedure

This procedure exports the specified data mining models to a dump file set.

To import the models from the dump file set, use the IMPORT_MODEL Procedure. EXPORT_MODEL and IMPORT_MODEL use Oracle Data Pump technology.

When Oracle Data Pump is used to export/import an entire schema or database, the mining models in the schema or database are included. However, EXPORT_MODEL and IMPORT_MODEL are the only utilities that support the export/import of individual models.

See Also:

Oracle Database Utilities for information about Oracle Data Pump

Oracle Data Mining User's Guide for more information about exporting and importing mining models

Syntax

```
DBMS_DATA_MINING.EXPORT_MODEL (  
    filename          IN VARCHAR2,  
    directory         IN VARCHAR2,  
    model_filter      IN VARCHAR2 DEFAULT NULL,  
    filesize          IN VARCHAR2 DEFAULT NULL,  
    operation         IN VARCHAR2 DEFAULT NULL,  
);  
```
remote_link       IN VARCHAR2 DEFAULT NULL,
jobname           IN VARCHAR2 DEFAULT NULL);

Parameters

Table 43-67    EXPORT_MODEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name of the dump file set to which the models should be exported. The name must be unique within the schema.</td>
</tr>
<tr>
<td></td>
<td>The dump file set can contain one or more files. The number of files in a dump file set is determined by the size of the models being exported (both metadata and data) and a specified or estimated maximum file size. You can specify the file size in the filesize parameter, or you can use the operation parameter to cause Oracle Data Pump to estimate the file size. If the size of the models to export is greater than the maximum file size, one or more additional files are created. When the export operation completes successfully, the name of the dump file set is automatically expanded to filename01.dmp, even if there is only one file in the dump set. If there are additional files, they are named sequentially as filename02.dmp, filename03.dmp, and so forth.</td>
</tr>
<tr>
<td>directory</td>
<td>Name of a pre-defined directory object that specifies where the dump file set should be created.</td>
</tr>
<tr>
<td></td>
<td>The exporting user must have read/write privileges on the directory object and on the file system directory that it identifies. See Oracle Database SQL Language Reference for information about directory objects.</td>
</tr>
<tr>
<td>model_filter</td>
<td>Optional parameter that specifies which model or models to export. If you do not specify a value for model_filter, all models in the schema are exported. You can also specify NULL (the default) or 'ALL' to export all models. You can export individual models by name and groups of models based on mining function or algorithm. For instance, you could export all regression models or all Naive Bayes models. Examples are provided in Table 43-68.</td>
</tr>
<tr>
<td>filesize</td>
<td>Optional parameter that specifies the maximum size of a file in the dump file set. The size may be specified in bytes, kilobytes (K), megabytes (M), or gigabytes (G). The default size is 50 MB. If the size of the models to export is larger than filesize, one or more additional files are created within the dump set. See the description of the filename parameter for more information.</td>
</tr>
<tr>
<td>operation</td>
<td>Optional parameter that specifies whether or not to estimate the size of the files in the dump set. By default the size is not estimated and the value of the filesize parameter determines the size of the files. You can specify either of the following values for operation:</td>
</tr>
<tr>
<td></td>
<td>· 'EXPORT' — Export all or the specified models. (Default)</td>
</tr>
<tr>
<td></td>
<td>· 'ESTIMATE' — Estimate the size of the exporting models.</td>
</tr>
<tr>
<td>remote_link</td>
<td>Optional parameter that specifies the name of a database link to a remote system. The default value is NULL. A database link is a schema object in a local database that enables access to objects in a remote database. When you specify a value for remote_link, you can export the models in the remote database. The EXP_FULL_DATABASE role is required for exporting the remote models. The EXP_FULL_DATABASE privilege, the CREATE DATABASE LINK privilege, and other privileges may also be required.</td>
</tr>
</tbody>
</table>
Table 43-67  (Cont.) EXPORT_MODEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobname</td>
<td>Optional parameter that specifies the name of the export job. By default, the name has the form <code>username_exp_nnnn</code>, where <code>nnnn</code> is a number. For example, a job name in the <code>SCOTT</code> schema might be <code>SCOTT_exp_134</code>. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the export job, named <code>jobname.log</code>, is created in the same directory as the dump file set.</td>
</tr>
</tbody>
</table>

Usage Notes

The `model_filter` parameter specifies which models to export. You can list the models by name, or you can specify all models that have the same mining function or algorithm. You can query the `USER_MINING_MODELS` view to list the models in your schema.

```
SQL> describe user_mining_models
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>MINING_FUNCTION</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>ALGORITHM</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td>BUILD_DURATION</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>MODEL_SIZE</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>COMMENTS</td>
<td></td>
<td>VARCHAR2(4000)</td>
</tr>
</tbody>
</table>

Examples of model filters are provided in Table 43-68.

Table 43-68  Sample Values for the Model Filter Parameter

<table>
<thead>
<tr>
<th>Sample Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>'mymodel'</td>
<td>Export the model named <code>mymodel</code></td>
</tr>
<tr>
<td>'name= ''mymodel'''</td>
<td>Export the model named <code>mymodel</code></td>
</tr>
<tr>
<td>'name IN (''mymodel2'',''mymodel3'')'</td>
<td>Export the models named <code>mymodel2</code> and <code>mymodel3</code></td>
</tr>
<tr>
<td>'ALGORITHM_NAME = ''NAIVE_BAYES'''</td>
<td>Export all Naive Bayes models. See Table 43-5 for a list of algorithm names.</td>
</tr>
<tr>
<td>'FUNCTION_NAME =''CLASSIFICATION'''</td>
<td>Export all classification models. See Table 43-3 for a list of mining functions.</td>
</tr>
</tbody>
</table>

Examples

1. The following statement exports all the models in the `DMUSER3` schema to a dump file set called `models_out` in the directory `$ORACLE_HOME/rdbms/log`. This directory is mapped to a directory object called `DATA_PUMP_DIR`. The `DMUSER3` user has read/write access to the directory and to the directory object.

   ```sql
   SQL>execute dbms_data_mining.export_model ('models_out', 'DATA_PUMP_DIR');
   ```

   You can exit SQL*Plus and list the resulting dump file and log file.
2. The following example uses the same directory object and is executed by the same user. This example exports the models called NMF_SH_SAMPLE and SVMR_SH_REGR_SAMPLE to a different dump file set in the same directory.

```sql
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL ('models2_out', 'DATA_PUMP_DIR', 'name in ("NMF_SH_SAMPLE", "SVMR_SH_REGR_SAMPLE")');
```

3. The following examples show how to export models with specific algorithm and mining function names.

```sql
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL('algo.dmp','DM_DUMP', 'ALGORITHM_NAME IN ("O_CLUSTER","GENERALIZED_LINEAR_MODEL","SUPPORT_VECTOR_MACHINES","NAIVE_BAYES")');
```

```sql
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL('func.dmp', 'DM_DUMP', 'FUNCTION_NAME IN {CLASSIFICATION,CLUSTERING,FEATURE_EXTRACTION}');
```

### 43.7.18 EXPORT_SERMODEL Procedure

This procedure exports the model in a serialized format so that they can be moved to another platform for scoring.

When exporting a model in serialized format, the user must pass in an empty `BLOB` locator and specify the model name to be exported. If the model is partitioned, the user can optionally select an individual partition to export, otherwise all partitions are exported. The returned `BLOB` contains the content that can be deployed.

#### Syntax

```sql
DBMS_DATA_MINING.EXPORT_SERMODEL (
    model_data   IN OUT NOCOPY BLOB,
    model_name   IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_data</td>
<td>Provides serialized model data.</td>
</tr>
<tr>
<td>model_name</td>
<td>Name of the mining model in the form [schema_name.]model_name. If you do</td>
</tr>
<tr>
<td></td>
<td>not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition that must be exported.</td>
</tr>
</tbody>
</table>
Examples

The following statement exports all the models in a serialized format.

```
DECLARE
  v_blob blob;
BEGIN
  dbms_lob.createtemporary(v_blob, FALSE);
  dbms_data_mining.export_sermodel(v_blob, 'MY_MODEL');
  -- save v_blob somewhere (e.g., bfile, etc.)
  dbms_lob.freetemporary(v_blob);
END;
/
```

See Also:
Oracle Data Mining User's Guide for more information about exporting and importing mining models

43.7.19 FETCH_JSON_SCHEMA Procedure

User can fetch and read JSON schema from the ALL_MINING_ALGORITHMS view. This function returns the pre-registered JSON schema for R extensible algorithms.

Syntax

```
DBMS_DATA_MINING.FETCH_JSON_SCHEMA RETURN CLOB;
```

Parameters

Table 43-70  FETCH_JSON_SCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN</td>
<td>This function returns the pre-registered JSON schema for R extensibility. The default value is CLOB.</td>
</tr>
</tbody>
</table>

Usage Note

If a user wants to register a new algorithm using the algorithm registration function, they must fetch and follow the pre-registered JSON schema using this function, when they create the required JSON object metadata, and then pass it to the registration function.

43.7.20 GET_ASSOCIATION_RULES Function

The GET_ASSOCIATION_RULES function returns the rules produced by an Association model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User's Guide

You can specify filtering criteria to GET_ASSOCIATION_RULES to return a subset of the rules. Filtering criteria can improve the performance of the table function. If the number of rules is
large, then the greatest performance improvement will result from specifying the \texttt{topn} parameter.

**Syntax**

```sql
DBMS_DATA_MINING.get_association_rules(
    model_name       IN VARCHAR2,
    topn             IN NUMBER DEFAULT NULL,
    rule_id          IN INTEGER DEFAULT NULL,
    min_confidence   IN NUMBER DEFAULT NULL,
    min_support      IN NUMBER DEFAULT NULL,
    max_rule_length  IN INTEGER DEFAULT NULL,
    min_rule_length  IN INTEGER DEFAULT NULL,
    sort_order       IN ORA_MINING_VARCHAR2_NT DEFAULT NULL,
    antecedent_items IN DM_ITEMS DEFAULT NULL,
    consequent_items IN DM_ITEMS DEFAULT NULL,
    min_lift         IN NUMBER DEFAULT NULL,
    partition_name   IN VARCHAR2 DEFAULT NULL)
RETURN DM_Rules PIPELINED;
```

**Parameters**

**Table 43-71  GET_ASSOCIATION_RULES Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{model_name}</td>
<td>Name of the model in the form [schema_name.]\texttt{model_name}. If you do not specify a schema, then your own schema is used. This is the only required parameter of GET_ASSOCIATION_RULES. All other parameters specify optional filters on the rules to return.</td>
</tr>
<tr>
<td>\texttt{topn}</td>
<td>Returns the \textit{n} top rules ordered by confidence and then support, both descending. If you specify a sort order, then the top \textit{n} rules are derived after the sort is performed. If \texttt{topn} is specified and no maximum or minimum rule length is specified, then the only columns allowed in the sort order are \texttt{RULE_CONFIDENCE} and \texttt{RULE_SUPPORT}. If \texttt{topn} is specified and a maximum or minimum rule length is specified, then \texttt{RULE_CONFIDENCE}, \texttt{RULE_SUPPORT}, and \texttt{NUMBER_OF_ITEMS} are allowed in the sort order.</td>
</tr>
<tr>
<td>\texttt{rule_id}</td>
<td>Identifier of the rule to return. If you specify a value for \texttt{rule_id}, do not specify values for the other filtering parameters.</td>
</tr>
<tr>
<td>\texttt{min_confidence}</td>
<td>Returns the rules with confidence greater than or equal to this number.</td>
</tr>
<tr>
<td>\texttt{min_support}</td>
<td>Returns the rules with support greater than or equal to this number.</td>
</tr>
<tr>
<td>\texttt{max_rule_length}</td>
<td>Returns the rules with a length less than or equal to this number. Rule length refers to the number of items in the rule (See \texttt{NUMBER_OF_ITEMS} in Table 43-72). For example, in the rule \texttt{A}$\Rightarrow$\texttt{B} (if \texttt{A}, then \texttt{B}), the number of items is 2. If \texttt{max_rule_length} is specified, then the \texttt{NUMBER_OF_ITEMS} column is permitted in the sort order.</td>
</tr>
<tr>
<td>\texttt{min_rule_length}</td>
<td>Returns the rules with a length greater than or equal to this number. See \texttt{max_rule_length} for a description of rule length. If \texttt{min_rule_length} is specified, then the \texttt{NUMBER_OF_ITEMS} column is permitted in the sort order.</td>
</tr>
</tbody>
</table>
Table 43-71  (Cont.) GET_ASSOCIATION_RULES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort_order</td>
<td>Sorts the rules by the values in one or more of the returned columns. Specify one or more column names, each followed by ASC for ascending order or DESC for descending order. (See Table 43-72 for the column names.) For example, to sort the result set in descending order first by the NUMBER_OF_ITEMS column, then by the RULE_CONFIDENCE column, you must specify: ORA_MINING_VARCHAR2_NT('NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC') If you specify topn, the results will vary depending on the sort order. By default, the results are sorted by Confidence in descending order, then by Support in descending order.</td>
</tr>
<tr>
<td>antecedent_items</td>
<td>Returns the rules with these items in the antecedent.</td>
</tr>
<tr>
<td>consequent_items</td>
<td>Returns the rules with this item in the consequent.</td>
</tr>
<tr>
<td>min_lift</td>
<td>Returns the rules with lift greater than or equal to this number.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

Return Values

The object type returned by GET_ASSOCIATION_RULES is described in Table 43-72. For descriptions of each field, see the Usage Notes.

Table 43-72  GET_ASSOCIATION RULES Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_RULES</td>
<td>A set of rows of type DM_RULE. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(rule_id INTEGER, antecedent DM_PREDICATES, consequent DM_PREDICATES, rule_support NUMBER, rule_confidence NUMBER, rule_lift NUMBER, antecedent_support NUMBER, consequent_support NUMBER, number_of_items INTEGER)</td>
</tr>
<tr>
<td>DM_PREDICATES</td>
<td>The antecedent and consequent columns each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(attribute_name VARCHAR2(4000), attribute_subname VARCHAR2(4000), conditional_operator CHAR(2)/<em>=,&lt;&gt;,&lt;,&gt;,&lt;=,&gt;=</em>/, attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000), attribute_support NUMBER, attribute_confidence NUMBER)</td>
</tr>
</tbody>
</table>
Usage Notes

1. This table function pipes out rows of type `DM_RULES`. For information on Data Mining data types and piped output from table functions, see "Datatypes".

2. The columns returned by `GET_ASSOCIATION_RULES` are described as follows:

<table>
<thead>
<tr>
<th>Column in <code>DM_RULES</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_id</td>
<td>Unique identifier of the rule. The independent condition in the rule. When this condition exists, the dependent condition in the consequent also exists. The condition is a combination of attribute values called a predicate (<code>DM_PREDICATE</code>). The predicate specifies a condition for each attribute. The condition may specify equality (=), inequality (&lt;&gt;), greater than (&gt;), less than (&lt;), greater than or equal to (&gt;=), or less than or equal to (&lt;=) a given value. Support and Confidence for each attribute condition in the antecedent is returned in the predicate. Support is the number of transactions that satisfy the antecedent. Confidence is the likelihood that a transaction will satisfy the antecedent. <strong>Note:</strong> The occurrence of the attribute as a <code>DM_PREDICATE</code> indicates the presence of the item in the transaction. The actual value for <code>attribute_num_value</code> or <code>attribute_str_value</code> is meaningless. For example, the following predicate indicates that 'Mouse Pad' is present in the transaction even though the attribute value is NULL. <code>DM_PREDICATE('PROD_NAME', 'Mouse Pad', '=' , NULL, NULL, NULL, NULL)</code></td>
</tr>
<tr>
<td>antecedent</td>
<td>The dependent condition in the rule. This condition exists when the antecedent exists. The consequent, like the antecedent, is a predicate (<code>DM_PREDICATE</code>). Support and confidence for each attribute condition in the consequent is returned in the predicate. Support is the number of transactions that satisfy the consequent. Confidence is the likelihood that a transaction will satisfy the consequent.</td>
</tr>
<tr>
<td>consequent</td>
<td>The number of transactions that satisfy the rule. The likelihood of a transaction satisfying the rule. The degree of improvement in the prediction over random chance when the rule is satisfied. The ratio of the number of transactions that satisfy the antecedent to the total number of transactions. The ratio of the number of transactions that satisfy the consequent to the total number of transactions. The total number of attributes referenced in the antecedent and consequent of the rule.</td>
</tr>
<tr>
<td>rule_support</td>
<td></td>
</tr>
<tr>
<td>rule_confidence</td>
<td></td>
</tr>
<tr>
<td>rule_lift</td>
<td></td>
</tr>
<tr>
<td>antecedent_support</td>
<td></td>
</tr>
<tr>
<td>consequent_support</td>
<td></td>
</tr>
<tr>
<td>number_of_items</td>
<td></td>
</tr>
</tbody>
</table>
Examples

The following example demonstrates an Association model build followed by several invocations of the GET_ASSOCIATION_RULES table function:

```sql
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
SELECT *
FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE market_settings
SET setting_value = TO_CHAR(0.081)
WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;
END;
/
-- build an AR model
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'market_model',
  function => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name => 'market_build',
  case_id_column_name => 'item_id',
  target_column_name => NULL,
  settings_table_name => 'market_settings');
END;
/
-- View the (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
  rule_confidence
FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model'));

In the previous example, you view all rules. To view just the top 20 rules, use the following statement.

```sql
-- View the top 20 (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
  rule_confidence
FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model', 20));
```

The following query uses the Association model AR_SH_SAMPLE, which is created from one of the Oracle Data Mining sample programs:

```sql
SELECT * FROM TABLE (
  DBMS_DATA_MINING.GET_ASSOCIATION_RULES ('AR_SH_SAMPLE', 10, NULL, 0.5, 0.01, 2, 1,
  ORA_MINING_VARCHAR2_NT (   'NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC', 'RULE_SUPPORT DESC'),
  DM_ITEMS(DM_ITEM('CUSTPRODS', 'Mouse Pad', 1, NULL),
    DM_ITEM('CUSTPRODS', 'Standard Mouse', 1, NULL),
  DM_ITEMS(DM_ITEM('CUSTPRODS', 'Extension Cable', 1, NULL)));
```

The query returns three rules, shown as follows:

```
13  DM_PREDICATES(
    DM_PREDICATE('CUSTPRODS', 'Mouse Pad', '= ', 1, NULL, NULL, NULL),
    DM_PREDICATE('CUSTPRODS', 'Standard Mouse', '= ', 1, NULL, NULL, NULL))
DM_PREDICATES(
    DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
  .15532  .84393  2.7075  .18404  .3117  2
```
43.7.21 GET_FREQUENT_ITEMSETS Function

The GET_FREQUENT_ITEMSETS function returns a set of rows that represent the frequent itemsets from an Association model. Starting from Oracle Database 12c Release 2, this function is deprecated. See “Model Detail Views” in Oracle Data Mining User’s Guide.

For a detailed description of frequent itemsets, consult Oracle Data Mining Concepts.

Syntax

```sql
DBMS_DATA_MINING.get_frequent_itemsets(
    model_name IN VARCHAR2,
    topn IN NUMBER DEFAULT NULL,
    max_itemset_length IN NUMBER DEFAULT NULL,
    partition_name IN VARCHAR2 DEFAULT NULL
) RETURN DM_ItemSets PIPELINED;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form <code>[schema_name.]model_name</code>. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>topn</td>
<td>When not NULL, return the top n rows ordered by support in descending order</td>
</tr>
<tr>
<td>max_itemset_length</td>
<td>Maximum length of an item set.</td>
</tr>
</tbody>
</table>
**Table 43-73**  (Cont.) GET_FREQUENT_ITEMSETS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

*Note:* The `partition_name` columns applies only when the model is partitioned.

**Return Values**

**Table 43-74**  GET_FREQUENT_ITEMSETS Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_ITEMSETS</td>
<td>A set of rows of type DM_ITEMSET. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(partition_name VARCHAR2(128), itemsets_id NUMBER, items DM_ITEMS, support NUMBER, number_of_items NUMBER)</td>
</tr>
</tbody>
</table>

*Note:* The `partition_name` columns applies only when the model is partitioned.

The `items` column returns a nested table of type DM_ITEMS. The rows have type DM_ITEM:

| attribute_name   | VARCHAR2(4000), attribute_subname VARCHAR2(4000), attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000) |

**Usage Notes**

This table function pipes out rows of type DM_ITEMSETS. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".

**Examples**

The following example demonstrates an Association model build followed by an invocation of `GET_FREQUENT_ITEMSETS` table function from Oracle SQL.

```sql
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
```
SELECT *
FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
WHERE setting_name LIKE 'ASSO_%';
BEGIN
  -- update the value of the minimum confidence
  UPDATE market_settings
  SET setting_value = TO_CHAR(0.081)
  WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;
/* build a AR model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'market_model',
  function => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name => 'market_build',
  case_id_column_name => 'item_id',
  target_column_name => NULL,
  settings_table_name => 'market_settings');
END;
/

-- View the (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model'));

In the example above, you view all itemsets. To view just the top 20 itemsets, use the following statement:

-- View the top 20 (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model', 20));

See Also:
Oracle Data Mining User's Guide

43.7.22 GET_MODEL_COST_MATRIX Function

The GET * interfaces are replaced by model views, and Oracle recommends that users leverage the views instead. The GET_MODEL_COST_MATRIX function is replaced by the DM$VC prefixed view, Scoring Cost Matrix. The cost matrix used when building a Decision Tree is made available by the DM$VM prefixed view, Decision Tree Build Cost Matrix.

Refer to Model Detail View for Classification Algorithm.

The GET_MODEL_COST_MATRIX function returns the rows of a cost matrix associated with the specified model.

By default, this function returns the scoring cost matrix that was added to the model with the ADD_COST_MATRIX procedure. If you wish to obtain the cost matrix used to create a model, specify cost_matrix_type_create as the matrix_type. See Table 43-75.

See also ADD_COST_MATRIX Procedure.
Syntax

```sql
DBMS_DATA_MINING.GET_MODEL_COST_MATRIX (
  model_name IN VARCHAR2,
  matrix_type IN VARCHAR2 DEFAULT cost_matrix_type_score,
  partition_name IN VARCHAR2 DEFAULT NULL);
RETURN DM_COST_MATRIX PIPELINED;
```

Parameters

**Table 43-75 GET_MODEL_COST_MATRIX Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name].model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>matrix_type</td>
<td>The type of cost matrix.</td>
</tr>
<tr>
<td></td>
<td>COST_MATRIX_TYPE_SCORE — cost matrix used for scoring. (Default.)</td>
</tr>
<tr>
<td></td>
<td>COST_MATRIX_TYPE_CREATE — cost matrix used to create the model (Decision Tree only).</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition in a partitioned model</td>
</tr>
</tbody>
</table>

Return Values

**Table 43-76 GET_MODEL_COST_MATRIX Function Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_COST_MATRIX</td>
<td>A set of rows of type DM_COST_ELEMENT. The rows have the following columns: actual VARCHAR2(4000), NUMBER, predicted VARCHAR2(4000), cost NUMBER</td>
</tr>
</tbody>
</table>

Usage Notes

Only Decision Tree models can be built with a cost matrix. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the CLAS_COST_TABLE_NAME setting in the settings table for the model. See Table 43-7.

The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, you can use the REMOVE_COST_MATRIX procedure to remove the cost matrix and the ADD_COST_MATRIX procedure to add a new one.

The GET_MODEL_COST_MATRIX may return either the build or scoring cost matrix defined for a model or model partition.

If you do not specify a partitioned model name, then an error is displayed.

Example

This example returns the scoring cost matrix associated with the Naive Bayes model NB_SH_CLAS_SAMPLE.
column actual format a10
column predicted format a10
SELECT *
    FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
ORDER BY predicted, actual;

<table>
<thead>
<tr>
<th>ACTUAL</th>
<th>PREDICTED</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>.75</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>.25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>.00</td>
</tr>
</tbody>
</table>

### 43.7.23 GET_MODEL_DETAILS_AI Function

The `GET_MODEL_DETAILS_AI` function returns a set of rows that provide the details of an Attribute Importance model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User’s Guide.

#### Syntax

```sql
DBMS_DATA_MINING.get_model_details_ai(
    model_name IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL
) RETURN dm_ranked_attributes pipelined;
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name].model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

#### Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_RANKED_ATTRIBUTES</td>
<td>A set of rows of type DM_RANKED_ATTRIBUTE. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(attribute_name VARCHAR2(4000), attribute_subname VARCHAR2(4000), importance_value NUMBER, rank NUMBER(38))</td>
</tr>
</tbody>
</table>

#### Examples

The following example returns model details for the Attribute Importance model AI_SH_sample, which was created by the sample program dmaidemo.sql. For information about the sample programs, see Oracle Data Mining User’s Guide.
SELECT attribute_name, importance_value, rank
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_AI('AI_SH_sample'))
ORDER BY RANK;

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>IMPORTANCE_VALUE</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOUSEHOLD_SIZE</td>
<td>0.151685183</td>
<td>1</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>0.145294546</td>
<td>2</td>
</tr>
<tr>
<td>YRS_RESIDENCE</td>
<td>0.07838928</td>
<td>3</td>
</tr>
<tr>
<td>AGE</td>
<td>0.075027496</td>
<td>4</td>
</tr>
<tr>
<td>Y_BOX_GAMES</td>
<td>0.063039952</td>
<td>5</td>
</tr>
<tr>
<td>EDUCATION</td>
<td>0.059605314</td>
<td>6</td>
</tr>
<tr>
<td>HOME_THEATER_PACKAGE</td>
<td>0.056458722</td>
<td>7</td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>0.054652937</td>
<td>8</td>
</tr>
<tr>
<td>CUST_GENDER</td>
<td>0.035264741</td>
<td>9</td>
</tr>
<tr>
<td>BOOKKEEPING_APPLICATION</td>
<td>0.019204751</td>
<td>10</td>
</tr>
<tr>
<td>PRINTER_SUPPLIES</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>OS_DOC_SET_KANJI</td>
<td>-0.00050013</td>
<td>12</td>
</tr>
<tr>
<td>FLAT_PANEL_MONITOR</td>
<td>-0.00509564</td>
<td>13</td>
</tr>
<tr>
<td>BULK_PACK_DISKETTES</td>
<td>-0.00540822</td>
<td>14</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>-0.01201116</td>
<td>15</td>
</tr>
<tr>
<td>CUST_INCOME_LEVEL</td>
<td>-0.03951311</td>
<td>16</td>
</tr>
</tbody>
</table>

43.7.24 GET_MODELDETAILS_EM Function

The GET_MODELDETAILS_EM function returns a set of rows that provide statistics about the clusters produced by an Expectation Maximization model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User's Guide.

By default, the EM algorithm groups components into high-level clusters, and GET_MODELDETAILS_EM returns only the high-level clusters with their hierarchies. Alternatively, you can configure EM model to disable the grouping of components into high-level clusters. In this case, GET_MODELDETAILS_EM returns the components themselves as clusters with their hierarchies. See Table 43-12.

Syntax

```
DBMS_DATA_MINING.get_model_details_em(
    model_name VARCHAR2,
    cluster_id NUMBER DEFAULT NULL,
    attribute  VARCHAR2 DEFAULT NULL,
    centroid   NUMBER DEFAULT 1,
    histogram  NUMBER DEFAULT 1,
    rules      NUMBER DEFAULT 2,
    attribute_subname  VARCHAR2 DEFAULT NULL,
    topn_attributes NUMBER DEFAULT NULL,
    partition_name IN VARCHAR2 DEFAULT NULL)
RETURN dm_clusters PIPELINED;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
</tbody>
</table>
Table 43-79  (Cont.) GET_MODEL_DETAILS_EM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster_id</td>
<td>The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise, the details for all clusters are returned.</td>
</tr>
<tr>
<td>attribute</td>
<td>The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned.</td>
</tr>
<tr>
<td>centroid</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>• 1: Details about centroids are returned (default)</td>
</tr>
<tr>
<td></td>
<td>• 0: Details about centroids are not returned</td>
</tr>
<tr>
<td>histogram</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>• 1: Details about histograms are returned (default)</td>
</tr>
<tr>
<td></td>
<td>• 0: Details about histograms are not returned</td>
</tr>
<tr>
<td>rules</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>• 2: Details about rules are returned (default)</td>
</tr>
<tr>
<td></td>
<td>• 1: Rule summaries are returned</td>
</tr>
<tr>
<td></td>
<td>• 0: No information about rules is returned</td>
</tr>
<tr>
<td>attribute_subname</td>
<td>The name of a nested attribute. The full name of a nested attribute has the form:</td>
</tr>
<tr>
<td></td>
<td>attribute_name.attribute_subname</td>
</tr>
<tr>
<td></td>
<td>where attribute_name is the name of the column and attribute_subname is the name of the nested attribute in that column. If the attribute is not nested, then attribute_subname is null.</td>
</tr>
<tr>
<td>topn_attributes</td>
<td>Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the n attributes with the highest confidence values in the rules are returned.</td>
</tr>
<tr>
<td></td>
<td>If the number of attributes included in the rules is less than topn, then, up to n additional attributes in alphabetical order are returned.</td>
</tr>
<tr>
<td></td>
<td>If both the attribute and topn_attributes parameters are specified, then topn_attributes is ignored.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

Usage Notes

1. For information on Data Mining datatypes and Return Values for Clustering Algorithms piped output from table functions, see "Datatypes".

2. GET_MODEL_DETAILS functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

3. When cluster statistics are disabled (EMCS_CLUSTER_STATISTICS is set to EMCS_CLUS_STATS_DISABLE), GET_MODEL_DETAILS_EM does not return centroids, histograms, or rules. Only taxonomy (hierarchy) and cluster counts are returned.

4. When the partition_name is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.
43.7.25 GET_MODELDETAILS_EM_COMP Function

The GET_MODELDETAILS_EM_COMP table function returns a set of rows that provide details about the parameters of an Expectation Maximization model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User's Guide.

Syntax

```sql
DBMS_DATA_MINING.get_model_details_em_comp(
    model_name IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL
) RETURN DM_EM_COMPONENT_SET PIPELINED;
```

Parameters

Table 43-80  GET_MODELDETAILS_EM_COMP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name].model_name. If you do not specify a schema, your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model to retrieve details for.</td>
</tr>
</tbody>
</table>

Return Values

Table 43-81  GET_MODELDETAILS_EM_COMP Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_EM_COMPONENT_SET</td>
<td>A set of rows of type DM_EM_COMPONENT. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>{info_type VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>component_id NUMBER,</td>
</tr>
<tr>
<td></td>
<td>cluster_id NUMBER,</td>
</tr>
<tr>
<td></td>
<td>attribute_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>covariate_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_value VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>value NUMBER )</td>
</tr>
</tbody>
</table>

Usage Notes

1. This table function pipes out rows of type DM_EM_COMPONENT. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".

The columns in each row returned by GET_MODELDETAILS_EM_COMP are described as follows:
The type of information in the row. The following information types are supported:

- cluster
- prior
- mean
- covariance
- frequency

Unique identifier of a component

Unique identifier of the high-level leaf cluster for each component

Name of an original attribute or a derived feature ID. The derived feature ID is used in models built on data with nested columns. The derived feature definitions can be obtained from the `GET_MODEL_DETAILS_EM_PROJ` Function.

Name of an original attribute or a derived feature ID used in variance/covariance definition

Categorical value or bin interval for binned numerical attributes

Encodes different information depending on the value of `info_type`, as follows:

- cluster — The value field is NULL
- prior — The value field returns the component prior
- mean — The value field returns the mean of the attribute specified in `attribute_name`
- covariance — The value field returns the covariance of the attributes specified in `attribute_name` and `covariate_name`. Using the same attribute in `attribute_name` and `covariate_name`, returns the variance.
- frequency — The value field returns the multivalued Bernoulli frequency parameter for the attribute/value combination specified by `attribute_name` and `attribute_value`

See Usage Note 2 for details.

The following table shows which fields are used for each `info_type`. The blank cells represent NULLs.

<table>
<thead>
<tr>
<th>info_type</th>
<th>component_id</th>
<th>cluster_id</th>
<th>attribute_name</th>
<th>covariate_name</th>
<th>attribute_value</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prior</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>mean</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>covariance</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>frequency</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
3. **GET_MODEL_DETAILS** functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

4. When the value is **NULL** for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

**Related Topics**

- Oracle Data Mining User’s Guide

### 43.7.26 GET_MODEL_DETAILS_EM_PROJ Function

The **GET_MODEL_DETAILS_EM_PROJ** function returns a set of rows that provide statistics about the projections produced by an Expectation Maximization model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User’s Guide.

**Syntax**

```sql
DBMS_DATA_MINING.get_model_details_em_proj(
    model_name IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL
) RETURN DM_EM_PROJECTION_SET PIPELINED;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_EM_PROJECTION_SET</td>
<td>A set of rows of type DM_EM_PROJECTION. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(feature_name VARCHAR2(4000), attribute_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_subname VARCHAR2(4000), attribute_value VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>coefficient NUMBER )</td>
</tr>
</tbody>
</table>

See Usage Notes for details.

**Usage Notes**

1. This table function pipes out rows of type **DM_EM_PROJECTION**. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".
The columns in each row returned by \texttt{GET\_MODEL\_DETAILS\_EM\_PROJ} are described as follows:

<table>
<thead>
<tr>
<th>Column in DM_EM_PROJECTION</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_name</td>
<td>Name of a derived feature. The feature maps to the attribute_name returned by the \texttt{GET_MODEL_DETAILS_EM} Function.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Name of a column in the build data</td>
</tr>
<tr>
<td>attribute_subname</td>
<td>Subname in a nested column</td>
</tr>
<tr>
<td>attribute_value</td>
<td>Categorical value</td>
</tr>
<tr>
<td>coefficient</td>
<td>Projection coefficient. The representation is sparse; only the non-zero coefficients are returned.</td>
</tr>
</tbody>
</table>

2. \texttt{GET\_MODEL\_DETAILS} functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

The coefficients are related to the transformed, not the original, attributes. When returned directly with the model details, the coefficients may not provide meaningful information.

3. When the value is \texttt{NULL} for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

**Related Topics**
- Oracle Data Mining User’s Guide

### 43.7.27 GET\_MODEL\_DETAILS\_GLM Function

The \texttt{GET\_MODEL\_DETAILS\_GLM} function returns the coefficient statistics for a Generalized Linear Model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User’s Guide.

The same set of statistics is returned for both linear and Logistic Regression, but statistics that do not apply to the mining function are returned as \texttt{NULL}. For more details, see the Usage Notes.

**Syntax**

```sql
DBMS\_DATA\_MINING.get_model_details_glm(
  model\_name IN VARCHAR2,
  partition\_name IN VARCHAR2 DEFAULT NULL)
RETURN DM\_GLM\_Coeff\_Set PIPELINED;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model</td>
</tr>
</tbody>
</table>
Return Values

Table 43-85  GET_MODEL_DETAILS_GLM Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_GLM_COEFF_SET</td>
<td>A set of rows of type DM_GLM_COEFF. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(class                   VARCHAR2(4000), attribute_name          VARCHAR2(4000), attribute_subname       VARCHAR2(4000), attribute_value         VARCHAR2(4000), feature_expression      VARCHAR2(4000), coefficient             NUMBER, std_error               NUMBER, test_statistic          NUMBER, p_value                 NUMBER, VIF                     NUMBER, std_coefficient         NUMBER, lower_coeff_limit       NUMBER, upper_coeff_limit       NUMBER, exp_coefficient         BINARY_DOUBLE, exp_lower_coeff_limit   BINARY_DOUBLE, exp_upper_coeff_limit   BINARY_DOUBLE)</td>
</tr>
</tbody>
</table>

GET_MODEL_DETAILS_GLM returns a row of statistics for each attribute and one extra row for the intercept, which is identified by a null value in the attribute name. Each row has the DM_GLM_COEFF datatype. The statistics are described in Table 43-86.

Table 43-86  DM_GLM_COEFF Datatype Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>The non-reference target class for Logistic Regression. The model is built to predict the probability of this class. The other class (the reference class) is specified in the model setting GLMS_REFERENCE_CLASS_NAME. See Table 43-18. For Linear Regression, class is null.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>The attribute name when there is no subname, or first part of the attribute name when there is a subname. The value of attribute_name is also the name of the column in the case table that is the source for this attribute. For the intercept, attribute_name is null. Intercepts are equivalent to the bias term in SVM models.</td>
</tr>
<tr>
<td>attribute_subname</td>
<td>The name of an attribute in a nested table. The full name of a nested attribute has the form: attribute_name.attribute_subname where attribute_name is the name of the nested column in the case table that is the source for this attribute. If the attribute is not nested, then attribute_subname is null. If the attribute is an intercept, then both the attribute_name and the attribute_subname are null.</td>
</tr>
</tbody>
</table>
Table 43-86 (Cont.) DM_GLM_COEFF Datatype Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_value</td>
<td>The value of the attribute (categorical attribute only).</td>
</tr>
<tr>
<td></td>
<td>For numeric attributes, attribute_value is null.</td>
</tr>
<tr>
<td>feature_expression</td>
<td>The feature name constructed by the algorithm when feature generation is enabled and higher-order features are found. If feature selection is not enabled, then the feature name is simply the fully-qualified attribute name (attribute_name.attribute_subname if the attribute is in a nested column). For categorical attributes, the algorithm constructs a feature name that has the following form: fully-qualified_attribute_name.attribute_value For numeric attributes, the algorithm constructs a name for the higher-order feature by taking the product of the resulting values: ( (\text{attrib1})^* (\text{attrib2})^* \ldots ) where attrib1 and attrib2 are fully-qualified attribute names.</td>
</tr>
<tr>
<td>coefficient</td>
<td>The linear coefficient estimate.</td>
</tr>
<tr>
<td>std_error</td>
<td>Standard error of the coefficient estimate.</td>
</tr>
<tr>
<td>test_statistic</td>
<td>For Linear Regression, the t-value of the coefficient estimate.</td>
</tr>
<tr>
<td></td>
<td>For Logistic Regression, the Wald chi-square value of the coefficient estimate.</td>
</tr>
<tr>
<td>p-value</td>
<td>Probability of the test_statistic. Used to analyze the significance of specific attributes in the model.</td>
</tr>
<tr>
<td>VIF</td>
<td>Variance Inflation Factor. The value is zero for the intercept. For Logistic Regression, VIF is null. VIF is not computed if the solver is Cholesky.</td>
</tr>
<tr>
<td>std_coefficient</td>
<td>Standardized estimate of the coefficient.</td>
</tr>
<tr>
<td>lower_coeff_limit</td>
<td>Lower confidence bound of the coefficient.</td>
</tr>
<tr>
<td>upper_coeff_limit</td>
<td>Upper confidence bound of the coefficient.</td>
</tr>
<tr>
<td>exp_coefficient</td>
<td>Exponentiated coefficient for Logistic Regression. For Linear Regression, exp_coefficient is null.</td>
</tr>
<tr>
<td>exp_lower_coeff_limit</td>
<td>Exponentiated coefficient for lower confidence bound of the coefficient for Logistic Regression. For Linear Regression, exp_lower_coeff_limit is null.</td>
</tr>
<tr>
<td>exp_upper_coeff_limit</td>
<td>Exponentiated coefficient for upper confidence bound of the coefficient for Logistic Regression. For Linear Regression, exp_upper_coeff_limit is null.</td>
</tr>
</tbody>
</table>

Usage Notes

Not all statistics are necessarily returned for each coefficient. Statistics will be null if:

- They do not apply to the mining function. For example, exp_coefficient does not apply to Linear Regression.
- They cannot be computed from a theoretical standpoint. For information on ridge regression, see Table 43-18.
- They cannot be computed because of limitations in system resources.
- Their values would be infinity.
- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

**Examples**

The following example returns some of the model details for the GLM Regression model `GLMR_SH_Regr_sample`, which was created by the sample program `dmglrdem.sql`. For information about the sample programs, see *Oracle Data Mining User's Guide*.

```
SET line 120
SET pages 99
COLUMN attribute_name format a30
COLUMN attribute_subname format a20
COLUMN attribute_value format a20
DBMS_OUTPUT.enable(NULL)
SQL> SELECT * FROM
(SELECT attribute_name, attribute_value, coefficient, std_error
 FROM DM$VDGLMR_SH_REGR_SAMPLE order by 1,2)
WHERE rownum < 11;
```

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>COEFFICIENT</th>
<th>STD_ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFFINITY_CARD</td>
<td></td>
<td>-0.5797</td>
<td>0.5283</td>
</tr>
<tr>
<td>BOOKKEEPING_APPLICATION</td>
<td></td>
<td>-0.4689</td>
<td>3.8872</td>
</tr>
<tr>
<td>BULK_PACK_DISKETTES</td>
<td></td>
<td>-0.9819</td>
<td>2.5430</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>Argentina</td>
<td>-1.2020</td>
<td>1.1876</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>Australia</td>
<td>-0.0071</td>
<td>5.1146</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>Brazil</td>
<td>5.2931</td>
<td>1.9233</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>Canada</td>
<td>4.0191</td>
<td>2.4108</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>China</td>
<td>0.8706</td>
<td>3.5889</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>Denmark</td>
<td>-2.9822</td>
<td>3.1803</td>
</tr>
<tr>
<td>COUNTRY_NAME</td>
<td>France</td>
<td>-1.1044</td>
<td>7.1811</td>
</tr>
</tbody>
</table>

**Related Topics**

- *Oracle Data Mining User's Guide*

**43.7.28 GET_MODEL_DETAILS_GLOBAL Function**

The `GET_MODEL_DETAILS_GLOBAL` function returns statistics about the model as a whole. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in *Oracle Data Mining User's Guide*.

Global details are available for Generalized Linear Models, Association Rules, Singular Value Decomposition, and Expectation Maximization. There are new Global model views which show global information for all algorithms. Oracle recommends that users leverage the views instead. Refer to Model Details View Global.

**Syntax**

```
DBMS_DATA_MINING.get_model_details_global(
    model_name IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL)
RETURN DM_model_global_details PIPELINED;
```
Parameters

Table 43-87  GET_MODEL_DETAILS_GLOBAL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name].model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

Return Values

Table 43-88  GET_MODEL_DETAILS_GLOBAL Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_MODEL_GLOBAL_DETAILS</td>
<td>A collection of rows of type DM_MODEL_GLOBAL_DETAIL. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(global_detail_name VARCHAR2(30), global_detail_value NUMBER)</td>
</tr>
</tbody>
</table>

Examples

The following example returns the global model details for the GLM Regression model GLMR_SH_Regr_sample, which was created by the sample program dmglrdem.sql. For information about the sample programs, see Oracle Data Mining User's Guide.

```sql
SELECT *
FROM TABLE(dbms_data_mining.get_model_details_global('GLMR_SH_Regr_sample'))
ORDER BY global_detail_name;
```

<table>
<thead>
<tr>
<th>GLOBAL_DETAIL_NAME</th>
<th>GLOBAL_DETAIL_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUSTED_R_SQUARE</td>
<td>.731412557</td>
</tr>
<tr>
<td>AIC</td>
<td>5931.814</td>
</tr>
<tr>
<td>COEFF_VAR</td>
<td>18.1711243</td>
</tr>
<tr>
<td>CORRECTED_TOTAL_DF</td>
<td>1499</td>
</tr>
<tr>
<td>CORRECTED_TOT_SS</td>
<td>278740.504</td>
</tr>
<tr>
<td>DEPENDENT_MEAN</td>
<td>38.892</td>
</tr>
<tr>
<td>ERROR_DF</td>
<td>1433</td>
</tr>
<tr>
<td>ERROR_MEAN_SQUARE</td>
<td>49.9440956</td>
</tr>
<tr>
<td>ERROR_SUM_SQUARES</td>
<td>71569.8891</td>
</tr>
<tr>
<td>F_VALUE</td>
<td>62.8492452</td>
</tr>
<tr>
<td>GMSEP</td>
<td>52.280819</td>
</tr>
<tr>
<td>HOCKING_SP</td>
<td>.034877162</td>
</tr>
<tr>
<td>J_P</td>
<td>52.1749319</td>
</tr>
<tr>
<td>MODEL_CONVERGED</td>
<td>1</td>
</tr>
<tr>
<td>MODEL_DF</td>
<td>66</td>
</tr>
<tr>
<td>MODEL_F_P_VALUE</td>
<td>0</td>
</tr>
<tr>
<td>MODEL_MEAN_SQUARE</td>
<td>3138.94871</td>
</tr>
<tr>
<td>MODEL_SUM_SQUARES</td>
<td>207170.615</td>
</tr>
<tr>
<td>NUM_PARAMS</td>
<td>67</td>
</tr>
<tr>
<td>NUM_ROWS</td>
<td>1500</td>
</tr>
<tr>
<td>ROOT_MEAN_SQ</td>
<td>7.06711367</td>
</tr>
<tr>
<td>R_SQ</td>
<td>.743238288</td>
</tr>
</tbody>
</table>
43.7.29 GET_MODEL_DETAILS_KM Function

The **GET_MODEL_DETAILS_KM** function returns a set of rows that provide the details of a \( k \)-Means clustering model. Starting from Oracle Database 12c Release 2, this function is deprecated. See “Model Detail Views” in *Oracle Data Mining User’s Guide*.

You can provide input to **GET_MODEL_DETAILS_KM** to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, then **GET_MODEL_DETAILS_KM** returns all the information about the model.

**Syntax**

```sql
DBMS_DATA_MINING.get_model_details_km(
    model_name VARCHAR2,
    cluster_id NUMBER DEFAULT NULL,
    attribute VARCHAR2 DEFAULT NULL,
    centroid NUMBER DEFAULT 1,
    histogram NUMBER DEFAULT 1,
    rules NUMBER DEFAULT 2,
    attribute_subname VARCHAR2 DEFAULT NULL,
    topn_attributes NUMBER DEFAULT NULL,
    partition_name VARCHAR2 DEFAULT NULL)
RETURN dm_clusters PIPELINED;
```

**Parameters**

**Table 43-89  GET_MODEL_DETAILS_KM Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>cluster_id</td>
<td>The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.</td>
</tr>
<tr>
<td>attribute</td>
<td>The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned.</td>
</tr>
<tr>
<td>centroid</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td>histogram</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td>rules</td>
<td>This parameter accepts the following values:</td>
</tr>
</tbody>
</table>
### Table 43-89  (Cont.) GET_MODEL_DETAILS_KM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_subname</td>
<td>The name of a nested attribute. The full name of a nested attribute has the form:  &lt;br&gt; <code>attribute_name.attribute_subname</code>  &lt;br&gt; where <code>attribute_name</code> is the name of the column and  &lt;br&gt; <code>attribute_subname</code> is the name of the nested attribute in that column.  &lt;br&gt; If the attribute is not nested, <code>attribute_subname</code> is null.</td>
</tr>
<tr>
<td>topn_attributes</td>
<td>Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the <code>n</code> attributes with the highest confidence values in the rules are returned.  &lt;br&gt; If the number of attributes included in the rules is less than <code>topn</code>, then up to <code>n</code> additional attributes in alphabetical order are returned.  &lt;br&gt; If both the <code>attribute</code> and <code>topn_attributes</code> parameters are specified, then <code>topn_attributes</code> is ignored.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

### Usage Notes

1. The table function pipes out rows of type `DM_CLUSTERS`. For information on Data Mining datatypes and Return Value for Clustering Algorithms piped output from table functions, see "Datatypes".

2. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

### Examples

The following example returns model details for the `k`-Means clustering model `KM_SH_Clus_sample`, which was created by the sample program `dmxmdemo.sql`. For information about the sample programs, see Oracle Data Mining User's Guide.

```
SELECT T.id clu_id,  
T.record_count rec_cnt,  
T.parent parent,  
T.tree_level tree_level,  
T.dispersion dispersion  
FROM (SELECT *  
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_KM('KM_SH_Clus_sample'))  
ORDER BY id) T  
WHERE ROWNUM < 6;
```

<table>
<thead>
<tr>
<th>CLU_ID</th>
<th>REC_CNT</th>
<th>PARENT</th>
<th>TREE_LEVEL</th>
<th>DISPERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1500</td>
<td></td>
<td>1</td>
<td>5.9152211</td>
</tr>
<tr>
<td>2</td>
<td>638</td>
<td>1</td>
<td>2</td>
<td>3.9845937</td>
</tr>
<tr>
<td>3</td>
<td>862</td>
<td>3</td>
<td>3</td>
<td>5.83732097</td>
</tr>
<tr>
<td>4</td>
<td>376</td>
<td>3</td>
<td>4</td>
<td>5.05192137</td>
</tr>
<tr>
<td>5</td>
<td>486</td>
<td>3</td>
<td>5</td>
<td>5.42901522</td>
</tr>
</tbody>
</table>

### Related Topics

- Oracle Data Mining User's Guide
43.7.30 GET_MODEL_DETAILS_NB Function

The GET_MODEL_DETAILS_NB function returns a set of rows that provide the details of a Naive Bayes model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User’s Guide.

Syntax

```sql
DBMS_DATA_MINING.get_model_details_nb(
    model_name IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL)
RETURN DM_NB_Details PIPELINED;
```

Parameters

Table 43-90  GET_MODEL_DETAILS_NB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model</td>
</tr>
</tbody>
</table>

Return Values

Table 43-91  GET_MODEL_DETAILS_NB Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_NB_DETAILS</td>
<td>A set of rows of type DM_NB_DETAIL. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(target_attribute_name VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>target_attribute_str_value VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>target_attribute_num_value NUMBER,</td>
</tr>
<tr>
<td></td>
<td>prior_probability NUMBER,</td>
</tr>
<tr>
<td></td>
<td>conditionals DM_CONDITIONALS)</td>
</tr>
<tr>
<td></td>
<td>The conditionals column of DM_NB_DETAIL returns a nested table of type</td>
</tr>
<tr>
<td></td>
<td>DM_CONDITIONALS. The rows, of type DM_CONDITIONAL, have the following</td>
</tr>
<tr>
<td></td>
<td>columns:</td>
</tr>
<tr>
<td></td>
<td>(attribute_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_subname VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_str_value VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_num_value NUMBER,</td>
</tr>
<tr>
<td></td>
<td>conditional_probability NUMBER)</td>
</tr>
</tbody>
</table>

Usage Notes

- The table function pipes out rows of type DM_NB_DETAILS. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".
- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.
Examples

The following query is from the sample program `dmnbdemo.sql`. It returns model details about the model `NB_SH_Clas_sample`. For information about the sample programs, see Oracle Data Mining User's Guide.

The query creates labels from the bin boundary tables that were used to bin the training data. It replaces the attribute values with the labels. For numeric bins, the labels are `([lower_boundary,upper_boundary])`; for categorical bins, the label matches the value it represents. (This method of categorical label representation will only work for cases where one value corresponds to one bin.) The target was not binned.

```sql
WITH bin_label_view AS (
    SELECT col, bin, (DECODE(bin,'1','[','(') || lv || ',' || val || ')') label
    FROM (SELECT col, bin,
            LAST_VALUE(val) OVER (
                PARTITION BY col ORDER BY val
                ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING) lv,
            val
    FROM nb_sh_sample_num)
    UNION ALL
    SELECT col, bin, val label
    FROM nb_sh_sample_cat
),
model_details AS (
    SELECT T.target_attribute_name tname,
            NVL(TO_CHAR(T.target_attribute_num_value,T.target_attribute_str_value)) tval,
            C.attribute_name pname,
            NVL(L.label, NVL(C.attribute_str_value, C.attribute_num_value)) pval,
            T.prior_probability priorp,
            C.conditional_probability condp
    FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NB('NB_SH_Clas_sample')) T,
            TABLE(T.conditionals) C,
            bin_label_view L
    WHERE C.attribute_name = L.col (+) AND
          (NVL(C.attribute_str_value,C.attribute_num_value) = L.bin(+))
    ORDER BY 1,2,3,4,5,6
)
SELECT tname, tval, pname, pval, priorp, condp
FROM model_details
WHERE ROWNUM < 11;
```

<table>
<thead>
<tr>
<th>TNAME</th>
<th>TVAL</th>
<th>PNAME</th>
<th>PVAL</th>
<th>PRIORP</th>
<th>CONDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>AGE</td>
<td>(24,30)</td>
<td>.6500</td>
<td>.1714</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>AGE</td>
<td>(30,35)</td>
<td>.6500</td>
<td>.1509</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>AGE</td>
<td>(35,40)</td>
<td>.6500</td>
<td>.1125</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>AGE</td>
<td>(40,46)</td>
<td>.6500</td>
<td>.1134</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>AGE</td>
<td>(46,53)</td>
<td>.6500</td>
<td>.1071</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>AGE</td>
<td>(53,90)</td>
<td>.6500</td>
<td>.1312</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>AGE</td>
<td>[17,24]</td>
<td>.6500</td>
<td>.2134</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>BOOKKEEPING_APPLICATION</td>
<td>0</td>
<td>.6500</td>
<td>.1500</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>BOOKKEEPING_APPLICATION</td>
<td>1</td>
<td>.6500</td>
<td>.8500</td>
</tr>
<tr>
<td>AFFINITY_CARD</td>
<td>0</td>
<td>BULK_PACK_DISKETTES</td>
<td>0</td>
<td>.6500</td>
<td>.3670</td>
</tr>
</tbody>
</table>
43.7.31 GET_MODELDETAILS_NMF Function

The GET_MODELDETAILS_NMF function returns a set of rows that provide the details of a Non-Negative Matrix Factorization model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User's Guide.

Syntax

```sql
DBMS_DATA_MINING.get_model_details_nmf(
    model_name IN VARCHAR2,
    partition_name VARCHAR2 DEFAULT NULL)
RETURN DM_NMF_Feature_Set PIPELINED;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form <code>[schema_name.]model_name</code>. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_NMF_FEATURE_SET</td>
<td>A set of rows of DM_NMF_FEATURE. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(feature_id NUMBER,</td>
</tr>
<tr>
<td></td>
<td>mapped_feature_id VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_set DM_NMF_ATTRIBUTE_SET)</td>
</tr>
<tr>
<td></td>
<td>The attribute_set column of DM_NMF_FEATURE returns a nested table of type DM_NMF_ATTRIBUTE_SET. The rows, of type DM_NMF_ATTRIBUTE, have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(attribute_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_subname VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_value VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>coefficient NUMBER)</td>
</tr>
</tbody>
</table>

Usage Notes

- The table function pipes out rows of type DM_NMF_FEATURE_SET. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".
- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.
Examples

The following example returns model details for the feature extraction model
NMF_SH_Sample, which was created by the sample program dmnmndemo.sql. For
information about the sample programs, see Oracle Data Mining User's Guide.

SELECT * FROM (  
SELECT F.feature_id,  
  A.attribute_name,  
  A.attribute_value,  
  A.coefficient  
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF('NMF_SH_Sample')) F,  
        TABLE(F.attribute_set) A  
) WHERE ROWNUM < 11;

<table>
<thead>
<tr>
<th>FEATURE_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>COEFFICIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AFFINITY_CARD</td>
<td></td>
<td>.051208078859308</td>
</tr>
<tr>
<td>1</td>
<td>AGE</td>
<td></td>
<td>.0390513260041573</td>
</tr>
<tr>
<td>1</td>
<td>BOOKKEEPING_APPLICATION</td>
<td></td>
<td>.0512734004239326</td>
</tr>
<tr>
<td>1</td>
<td>BULK_PACK_DISKETTES</td>
<td></td>
<td>.232471260895683</td>
</tr>
<tr>
<td>1</td>
<td>COUNTRY_NAME Argentina</td>
<td>.00766817464479959</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>COUNTRY_NAME Australia</td>
<td>.000157637881096675</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>COUNTRY_NAME Brazil</td>
<td>.0031409632415604</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>COUNTRY_NAME Canada</td>
<td>.00144213099311427</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>COUNTRY_NAME China</td>
<td>.000102279310968754</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>COUNTRY_NAME Denmark</td>
<td>.000242424084307513</td>
<td></td>
</tr>
</tbody>
</table>

Related Topics

- Oracle Data Mining User's Guide

43.7.32 GET_MODELDETAILS_OC Function

The GET_MODELDETAILS_OC function returns a set of rows that provide the details of an
O-Cluster clustering model. The rows are an enumeration of the Clustering patterns
generated during the creation of the model. Starting from Oracle Database 12c
Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining
User's Guide.

You can provide input to GET_MODELDETAILS_OC to request specific information about
the model, thus improving the performance of the query. If you do not specify filtering
parameters, then GET_MODELDETAILS_OC returns all the information about the model.

Syntax

DBMS_DATA_MINING.get_model_details_oc(  
  model_name VARCHAR2,  
  cluster_id NUMBER DEFAULT NULL,  
  attribute VARCHAR2 DEFAULT NULL,  
  centroid NUMBER DEFAULT 1,  
  histogram NUMBER DEFAULT 1,  
  rules NUMBER DEFAULT 2,  
  topn_attributes NUMBER DEFAULT NULL,  
  partition_name VARCHAR2 DEFAULT NULL)  
RETURN dm_clusters PIPELIINED;
## Parameters

### Table 43-94  GET_MODELDETAILS_OC Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.].model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>cluster_id</td>
<td>The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.</td>
</tr>
<tr>
<td>attribute</td>
<td>The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned</td>
</tr>
<tr>
<td>centroid</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td>histogram</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td>rules</td>
<td>This parameter accepts the following values:</td>
</tr>
<tr>
<td>topn_attributes</td>
<td>Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the ( n ) attributes with the highest confidence values in the rules are returned.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

### Usage Notes

1. For information about Data Mining datatypes and Return Values for Clustering Algorithms piped output from table functions, see “Datatypes”.

2. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

### Examples

The following example returns model details for the clustering model OC_SH_Clus_sample, which was created by the sample program dmocdemo.sql. For information about the sample programs, see Oracle Data Mining User's Guide.

For each cluster in this example, the split predicate indicates the attribute and the condition used to assign records to the cluster's children during model build. It provides an important piece of information on how the population within a cluster can be divided up into two smaller clusters.
SELECT clu_id, attribute_name, op, s_value
FROM (SELECT a.id clu_id, sp.attribute_name, sp.conditional_operator op,
       sp.attribute_str_value s_value
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_OC('OC_SH_Clus_sample')) a,
       TABLE(a.split_predicate) sp
ORDER BY a.id, op, s_value)
WHERE ROWNUM < 11;

Related Topics
- Oracle Data Mining User’s Guide

43.7.33 GET_MODEL_SETTINGS Function

The GET_MODEL_SETTINGS function returns the settings used to build the given model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Static Data Dictionary Views: ALL_ALL_TABLES to ALL_OUTLINES" in Oracle Database Reference.

Syntax

FUNCTION get_model_settings(model_name IN VARCHAR2)
  RETURN DM_Model_Settings PIPELINED;

Parameters

Table 43-95  GET_MODEL_SETTINGS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
</tbody>
</table>
Return Values

Table 43-96  GET_MODEL_SETTINGS Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_MODEL_SETTINGS</td>
<td>A set of rows of type DM_MODEL_SETTINGS. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>DM_MODEL_SETTINGS TABLE OF SYS.DM_MODEL_SETTING</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td>SETTING_NAME</td>
</tr>
<tr>
<td></td>
<td>SETTING_VALUE</td>
</tr>
</tbody>
</table>

Usage Notes

1. This table function pipes out rows of type DM_MODEL_SETTINGS. For information on Data Mining datatypes and piped output from table functions, see "DBMS_DATA_MINING Datatypes".
2. The setting names/values include both those specified by the user and any defaults assigned by the build process.

Examples

The following example returns model model settings for an example Naive Bayes model.

<table>
<thead>
<tr>
<th>SETTING_NAME</th>
<th>SETTING_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_NAIVE_BAYES</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
<tr>
<td>ODM_S_MAX_PARTITIONS</td>
<td>1000</td>
</tr>
<tr>
<td>NABS_SINGLETON_THRESHOLD</td>
<td>0</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>OFF</td>
</tr>
<tr>
<td>NABS_PAIRWISE_THRESHOLD</td>
<td>0</td>
</tr>
<tr>
<td>ODM_PARTITION_COLUMNS</td>
<td>GENDER,Y_BOX_GAMES</td>
</tr>
<tr>
<td>ODM_MISSING_VALUE_TREATMENT</td>
<td>ODM_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODM_SAMPLING</td>
<td>ODM_SAMPLING_DISABLE</td>
</tr>
</tbody>
</table>

9 rows selected.

Related Topics

• Oracle Database Reference

43.7.34 GET_MODEL_SIGNATURE Function

The get_model_signature function returns the list of columns from the build input table that were used by the build process to train the model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Static Data Dictionary Views: ALL_ALL_TABLES to ALL_OUTLINES" in Oracle Database Reference.

Syntax

FUNCTION get_model_signature (model_name IN VARCHAR2)
RETURN DM_Model_Signature PIPELINED;
Parameters

Table 43-97   GET_MODEL_SIGNATURE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
</tbody>
</table>

Return Values

Table 43-98   GET_MODEL_SIGNATURE Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_MODEL_SIGNATURE</td>
<td>A set of rows of type DM_MODEL_SIGNATURE. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>DM_MODEL_SIGNATURE TABLE OF</td>
</tr>
<tr>
<td></td>
<td>SYS.DM_MODEL_SIGNATURE_ATTRIBUTE</td>
</tr>
<tr>
<td>Name</td>
<td>ATTRIBUTE_NAME</td>
</tr>
<tr>
<td>Type</td>
<td>ATTRIBUTE_TYPE</td>
</tr>
</tbody>
</table>

Usage Notes

1. This table function pipes out rows of type DM_MODEL_SIGNATURE. For information on Data Mining datatypes and piped output from table functions, see "DBMS_DATA_MINING Datatypes".

2. The signature names or types include only those attributes used by the build process.

Examples

The following example returns model settings for an example Naive Bayes model.

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE</td>
<td>NUMBER</td>
</tr>
<tr>
<td>ANNUAL_INCOME</td>
<td>NUMBER</td>
</tr>
<tr>
<td>AVERAGE__ITEMS_PURCHASED</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BOOKKEEPING_APPLICATION</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BULK_PACK_DISKETTES</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BULK_PURCH_AVE_AMT</td>
<td>NUMBER</td>
</tr>
<tr>
<td>DISABLE.Cookies</td>
<td>NUMBER</td>
</tr>
<tr>
<td>EDUCATION</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>FLAT_PANEL_MONITOR</td>
<td>NUMBER</td>
</tr>
<tr>
<td>GENDER</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>HOME_THEATER_PACKAGE</td>
<td>NUMBER</td>
</tr>
<tr>
<td>HOUSEHOLD_SIZE</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>MAILING_LIST</td>
<td>NUMBER</td>
</tr>
<tr>
<td>MARITAL_STATUS</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>NO_DIFFERENT_KIND_ITEMS</td>
<td>NUMBER</td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>OS_DOC_SET_KANJI</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>
The `GET_MODEL_DETAILS_SVD` function returns a set of rows that provide the details of a Singular Value Decomposition model. Oracle recommends to use model details view settings. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User's Guide.

Refer to Model Details View for Singular Value Decomposition.

**Syntax**

```sql
DBMS_DATA_MINING.get_model_details_svd(
    model_name IN VARCHAR2,
    matrix_type IN VARCHAR2 DEFAULT NULL,
    partition_name VARCHAR2 DEFAULT NULL
) RETURN DM_SVD_MATRIX_Set PIPELINED;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form <code>[schema_name.]model_name</code>. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>matrix_type</td>
<td>Specifies which of the three SVD matrix types to return. Values are: <code>U</code>, <code>S</code>, <code>V</code>, and <code>NULL</code>. When <code>matrix_type</code> is null (default), all matrices are returned.</td>
</tr>
<tr>
<td></td>
<td>The <code>U</code> matrix is only computed when the <code>SVDS_U_MATRIX_OUTPUT</code> setting is enabled. It is not computed by default. If the model does not contain <code>U</code> matrices and you set <code>matrix_type</code> to <code>U</code>, an empty set of rows is returned. See Table 43-26.</td>
</tr>
<tr>
<td>partition_name</td>
<td>A partition in a partitioned model.</td>
</tr>
</tbody>
</table>
Return Values

Table 43-100  GET_MODELDETAILS_SVD Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_SVD_MATRIX_SET</td>
<td>A set of rows of type DM_SVD_MATRIX. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(matrix_type CHAR(1),</td>
</tr>
<tr>
<td></td>
<td>feature_id NUMBER,</td>
</tr>
<tr>
<td></td>
<td>mapped_feature_id VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>attribute_subname VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>case_id VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>value NUMBER,</td>
</tr>
<tr>
<td></td>
<td>variance NUMBER,</td>
</tr>
<tr>
<td></td>
<td>pct_cum_variance NUMBER)</td>
</tr>
</tbody>
</table>

See Usage Notes for details.

Usage Notes

1. This table function pipes out rows of type DM_SVD_MATRIX. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".

The columns in each row returned by GET_MODELDETAILS_SVD are described as follows:

<table>
<thead>
<tr>
<th>Column in DM_SVD_MATRIX_SET</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix_type</td>
<td>The type of matrix. Possible values are S, V, and U. This field is never null.</td>
</tr>
<tr>
<td>feature_id</td>
<td>The feature that the matrix entry refers to.</td>
</tr>
<tr>
<td>mapped_feature_id</td>
<td>A descriptive name for the feature.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Column name in the V matrix component bases. This field is null for the S and U matrices.</td>
</tr>
<tr>
<td>attribute_subname</td>
<td>Subname in the V matrix component bases. This is relevant only in the case of a nested column. This field is null for the S and U matrices.</td>
</tr>
<tr>
<td>case_id</td>
<td>Unique identifier of the row in the build data described by the U matrix projection. This field is null for the S and V matrices.</td>
</tr>
<tr>
<td>value</td>
<td>The matrix entry value.</td>
</tr>
<tr>
<td>variance</td>
<td>The variance explained by a component. It is non-null only for S matrix entries. This column is non-null only for S matrix entries and for SVD models with setting dbms_data_mining.svds_scoring_mode set to dbms_data_mining.svds_scoring_pca and the build data is centered, either manually or because the setting dbms_data_mining.prep_auto is set to dbms_data_mining.prep_auto_on.</td>
</tr>
</tbody>
</table>
### Column in DM_SVD_MATRIX_SET

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pct_cum_variance</td>
</tr>
<tr>
<td>The percent cumulative variance explained by the components thus far. The components are ranked by the explained variance in descending order. This column is non-null only for $S$ matrix entries and for SVD models with setting <code>dbms_data_mining.svds_scoring_mode</code> set to <code>dbms_data_mining.svds_scoring_pca</code> and the build data is centered, either manually or because the setting <code>dbms_data_mining.prep_auto</code> is set to <code>dbms_data_mining.prep_auto_on</code>.</td>
</tr>
</tbody>
</table>

2. The output of GET_MODELDETAILS is in sparse format. Zero values are not returned. Only the diagonal elements of the $S$ matrix, the non-zero coefficients in the $V$ matrix bases, and the non-zero $U$ matrix projections are returned.

   There is one exception: If the data row does not produce non-zero $U$ Matrix projections, the case ID for that row is returned with `NULL` for the `feature_id` and `value`. This is done to avoid losing any records from the original data.

3. GET_MODELDETAILS functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

4. When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the preferred partition name.

### Related Topics

- *Oracle Data Mining User's Guide*

### 43.7.36 GET_MODELDETAILS_SVM Function

The GET_MODELDETAILS_SVM function returns a set of rows that provide the details of a linear Support Vector Machine (SVM) model. If invoked for nonlinear SVM, it returns ORA-40215. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Model Detail Views" in Oracle Data Mining User's Guide.

In linear SVM models, only nonzero coefficients are stored. This reduces storage and speeds up model loading. As a result, if an attribute is missing in the coefficient list returned by GET_MODELDETAILS_SVM, then the coefficient of this attribute should be interpreted as zero.

### Syntax

```sql
DBMS_DATA_MINING.get_model_details_svm(
    model_name   VARCHAR2,
    reverse_coef NUMBER DEFAULT 0,
    partition_name VARCHAR2 DEFAULT NULL)
RETURN DM_SVM_Linear_Coeff_Set PIPELINED;
```
Parameters

Table 43-101  GET_MODEL_DETAILS_SVM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>reverse_coef</td>
<td>Whether or not GET_MODEL_DETAILS_SVM should transform the attribute coefficients using the original attribute transformations. When reverse_coef is set to 0 (default), GET_MODEL_DETAILS_SVM returns the coefficients directly from the model without applying transformations. When reverse_coef is set to 1, GET_MODEL_DETAILS_SVM transforms the coefficients and bias by applying the normalization shifts and scales that were generated using automatic data preparation. See Usage Note 4.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

Return Values

Table 43-102  GET_MODEL_DETAILS_SVM Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_SVM_LINEAR_COEFF_SET</td>
<td>A set of rows of type DM_SVM_LINEAR_COEFF. The rows have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(class VARCHAR2(4000), attribute_set DM_SVM_ATTRIBUTE_SET)</td>
</tr>
<tr>
<td></td>
<td>The attribute_set column returns a nested table of type DM_SVM_ATTRIBUTE_SET. The rows, of type DM_SVM_ATTRIBUTE, have the following columns:</td>
</tr>
<tr>
<td></td>
<td>(attribute_name VARCHAR2(4000), attribute_subname VARCHAR2(4000), attribute_value VARCHAR2(4000), coefficient NUMBER)</td>
</tr>
</tbody>
</table>

See Usage Notes.

Usage Notes

1. This table function pipes out rows of type DM_SVM_LINEAR_COEFF. For information on Data Mining datatypes and piped output from table functions, see "Datatypes".

2. The class column of DM_SVM_LINEAR_COEFF contains Classification target values. For SVM Regression models, class is null. For each Classification target value, a set of coefficients is returned. For Binary Classification, one-class Classification, and Regression models, only a single set of coefficients is returned.

3. The attribute_value column in DM_SVM_ATTRIBUTE_SET is used for categorical attributes.
4. **GET MODEL DETAILS** functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

The coefficients are related to the transformed, not the original, attributes. When returned directly with the model details, the coefficients may not provide meaningful information. If you want **GET MODEL DETAILS_SVM** to transform the coefficients such that they relate to the original attributes, set the `reverse_coef` parameter to 1.

5. When the value is **NULL** for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

**Examples**

The following example returns model details for the SVM Classification model **SVMC_SH_Clas_sample**, which was created by the sample program **dmsvcdem.sql**. For information about the sample programs, see *Oracle Data Mining User's Guide*.

```
WITH mod_dtls AS (  
  SELECT *  
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM('SVMC_SH_Clas_sample'))  
),  
model_details AS (  
  SELECT D.class, A.attribute_name, A.attribute_value, A.coefficient  
  FROM mod_dtls D,  
  TABLE(D.attribute_set) A  
  ORDER BY D.class, ABS(A.coefficient) DESC  
)  
SELECT class, attribute_name aname, attribute_value aval, coefficient coeff  
FROM model_details  
WHERE ROWNUM < 11;
```

<table>
<thead>
<tr>
<th>CLASS</th>
<th>ANAME</th>
<th>AVAL</th>
<th>COEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOOKEEEPING_APPLICATION</td>
<td></td>
<td>-2.85</td>
</tr>
<tr>
<td>1</td>
<td>OCCUPATION</td>
<td>Other</td>
<td>-.94</td>
</tr>
<tr>
<td>1</td>
<td>HOUSEHOLD_SIZE</td>
<td>4-5</td>
<td>.88</td>
</tr>
<tr>
<td>1</td>
<td>CUST_MARITAL_STATUS</td>
<td>Married</td>
<td>.82</td>
</tr>
<tr>
<td>1</td>
<td>YRS_RESIDENCE</td>
<td></td>
<td>.76</td>
</tr>
<tr>
<td>1</td>
<td>HOUSEHOLD_SIZE</td>
<td>6-8</td>
<td>-.74</td>
</tr>
<tr>
<td>1</td>
<td>OCCUPATION</td>
<td>Exec.</td>
<td>.71</td>
</tr>
<tr>
<td>1</td>
<td>EDUCATION</td>
<td>11th</td>
<td>-.71</td>
</tr>
<tr>
<td>1</td>
<td>EDUCATION</td>
<td>Masters</td>
<td>.63</td>
</tr>
</tbody>
</table>

**Related Topics**

- *Oracle Data Mining User's Guide*

### 43.7.37 GET_MODEL_DETAILS_XML Function

This function returns an XML object that provides the details of a Decision Tree model.

**Syntax**

```sql
DBMS_DATA_MINING.get_model_details_xml(
  model_name IN VARCHAR2,
)"
```
partition_name IN VARCHAR2 DEFAULT NULL)
RETURN XMLType;

Parameters

Table 43-103  GET_MODELDETAILS_XML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form [schema_name.]model_name. If you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model.</td>
</tr>
</tbody>
</table>

Return Values

Table 43-104  GET_MODELDETAILS_XML Function Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLTYPE</td>
<td>The XML definition for the Decision Tree model. See &quot;XMLTYPE&quot; for details.</td>
</tr>
<tr>
<td></td>
<td>The XML definition conforms to the Data Mining Group Predictive Model Markup Language (PMML) version 2.1 specification. The specification is available at <a href="https://dmg.org">https://dmg.org</a>.</td>
</tr>
<tr>
<td></td>
<td>If a nested attribute is used as a splitter, the attribute will appear in the XML document as field=&quot;&lt;column_name&gt;.&lt;subname&gt;&quot;, as opposed to the non-nested attributes which appear in the document as field=&quot;&lt;column_name&gt;&quot;.</td>
</tr>
</tbody>
</table>

Note: The column names are surrounded by single quotes and a period separates the column_name from the subname. The rest of the document style remains unchanged.

Usage Notes

Special characters that cannot be displayed by Oracle XML are converted to '#'.

Examples

The following statements in SQL*Plus return the details of the Decision Tree model dt_sh_clas_sample. This model is created by the program dmdtdemo.sql, one of the sample data mining programs provided with Oracle Database Examples.

Note: The "&quot" characters you will see in the XML output are a result of SQL*Plus behavior. To display the XML in proper format, cut and past it into a file and open the file in a browser.

column dt_details format a320
SELECT
  dbms_data_mining.get_model_details_xml('dt_sh_clas_sample')
AS DT_DETAILS
FROM dual;

DT_DETAILS
--------------------------------------------------------------------------------
<PMML version="2.1">
<Header copyright="Copyright (c) 2004, Oracle Corporation. All rights reserved."/>
<DataDictionary numberOfFields="9">
<DataField name="AFFINITY_CARD" optype="categorical"/>
<DataField name="AGE" optype="continuous"/>
<DataField name="BOOKKEEPING_APPLICATION" optype="continuous"/>
<DataField name="CUSTOMER_STATUS" optype="categorical"/>
<DataField name="EDUCATION" optype="categorical"/>
<DataField name="HOUSEHOLD_SIZE" optype="categorical"/>
<DataField name="OCCUPATION" optype="categorical"/>
<DataField name="YRS_RESIDENCE" optype="continuous"/>
<DataField name="Y_BOX_GAMES" optype="continuous"/>
</DataDictionary>
<TreeModel modelName="DT_SH_CLAS_SAMPLE" functionName="classification" splitCharacteristic="binarySplit">
<Extension name="buildSettings">
<Setting name="TREE_IMPURITY_METRIC" value="TREE_IMPURITY_GINI"/>
<Setting name="TREE_TERM_MAX_DEPTH" value="7"/>
<Setting name="TREE_TERM_MINPCT_NODE" value=".05"/>
<Setting name="TREE_TERM_MINPCT_SPLIT" value=".1"/>
<Setting name="TREE_TERM_MINREC_NODE" value="10"/>
<Setting name="TREE_TERM_MINREC_SPLIT" value="20"/>
<costMatrix>
<costElement>
<actualValue>0</actualValue>
<predictedValue>0</predictedValue>
<cost>0</cost>
</costElement>
<costElement>
<actualValue>0</actualValue>
<predictedValue>1</predictedValue>
<cost>1</cost>
</costElement>
<costElement>
<actualValue>1</actualValue>
<predictedValue>0</predictedValue>
<cost>8</cost>
</costElement>
<costElement>
<actualValue>1</actualValue>
<predictedValue>0</predictedValue>
<cost>8</cost>
</costElement>
</costMatrix>
</Extension>
</TreeModel>
</PMML>
43.7.38 GET_MODEL_TRANSFORMATIONS Function

This function returns the transformation expressions embedded in the specified model. Starting from Oracle Database 12c Release 2, this function is deprecated. See "Static Data Dictionary Views: ALL_ALL_TABLES to ALL_OUTLINES" in Oracle Database Reference.

All GET_* interfaces are replaced by model views, and Oracle recommends that users reference the model views to retrieve the relevant information. The GET_MODEL_TRANSFORMATIONS function is replaced by the following:

- USER/(DBA/ALL)_MINING_MODEL_XFORMS: provides the user-embedded transformations
- DM$VX prefixed model view: provides text feature extraction information
- D$VN prefixed mode view: provides normalization and missing value information
- DM$VB: provides binning information

See Also:

- "About Transformation Lists" in DBMS_DATA_MINING_TRANSFORM Operational Notes
- "GET_TRANSFORM_LIST Procedure"
- "CREATE_MODEL Procedure"
- "ALL_MINING_MODEL_XFORMS" in Oracle Database Reference
- "DBA_MINING_MODEL_XFORMS" in Oracle Database Reference
- "USER_MINING_MODEL_XFORMS" in Oracle Database Reference

Model Details View for Binning
Normalization and Missing Value Handling
Data Preparation for Text Features

Syntax

```sql
DBMS_DATA_MINING.get_model_transformations(
    model_name IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL)
RETURN DM_Transforms PIPELINED;
```
Parameters

Table 43-105  GET_MODEL_TRANSFORMATIONS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Indicates the name of the model in the form [schema_name.model_name]. If</td>
</tr>
<tr>
<td></td>
<td>you do not specify a schema, then your own schema is used.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Specifies a partition in a partitioned model</td>
</tr>
</tbody>
</table>

Return Values

Table 43-106  GET_MODEL_TRANSFORMATIONS Function Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_TRANSFORMS</td>
<td>The transformation expressions embedded in model_name.</td>
</tr>
<tr>
<td></td>
<td>The DM_TRANSFORMS type is a table of DM_TRANSFORM objects. Each</td>
</tr>
<tr>
<td></td>
<td>DM_TRANSFORM has these fields:</td>
</tr>
<tr>
<td></td>
<td>attribute_name VARCHAR2(4000)</td>
</tr>
<tr>
<td></td>
<td>attribute_subname VARCHAR2(4000)</td>
</tr>
<tr>
<td></td>
<td>expression CLOB</td>
</tr>
<tr>
<td></td>
<td>reverse_expression CLOB</td>
</tr>
</tbody>
</table>

Usage Notes

When Automatic Data Preparation (ADP) is enabled, both automatic and user-defined transformations may be associated with an attribute. In this case, the user-defined transformations are evaluated before the automatic transformations.

When invoked for a partitioned model, the partition_name parameter must be specified.

Examples

In this example, several columns in the SH.CUSTOMERS table are used to create a Naive Bayes model. A transformation expression is specified for one of the columns. The model does not use ADP.

```sql
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_income_level, cust_credit_limit
  FROM sh.customers;

describe mining_data
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>NOT NULL</td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>CUST_INCOME_LEVEL</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>CUST_CREDIT_LIMIT</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

```sql
CREATE TABLE settings_nb(
  setting_name VARCHAR2(30),
  setting_value VARCHAR2(30));
BEGIN
  INSERT INTO settings_nb (setting_name, setting_value) VALUES
```
DECLARE
mining_data_xforms   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM (
    xform_list           =>  mining_data_xforms,
    attribute_name       => 'cust_year_of_birth',
    attribute_subname    =>  null,
    expression           => 'cust_year_of_birth + 10',
    reverse_expression   => 'cust_year_of_birth - 10');
  dbms_data_mining.CREATE_MODEL (
    model_name           =>  'new_model',
    mining_function      =>   dbms_data_mining.classification,
    data_table_name      =>  'mining_data',
    case_id_column_name  =>  'cust_id',
    target_column_name   =>  'cust_income_level',
    settings_table_name  =>  'settings_nb',
    data_schema_name     =>  null,
    settings_schema_name =>  null,
    xform_list           =>  mining_data_xforms
  );
END;
/
SELECT attribute_name, TO_CHAR(expression), TO_CHAR(reverse_expression)
FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('new_model'));

ATTRIBUTE_NAME      TO_CHAR(EXPRESSION)       TO_CHAR(REVERSE_EXPRESSION)
------------------  ------------------------  -----------------------------
CUST_YEAR_OF_BIRTH  cust_year_of_birth + 10   cust_year_of_birth - 10

Related Topics

- Oracle Database Reference

43.7.39 GET_TRANSFORM_LIST Procedure

This procedure converts transformation expressions specified as DM_TRANSFORMS to a transformation list (TRANSFORM_LIST) that can be used in creating a model. DM_TRANSFORMS is returned by the GET_MODEL_TRANSFORMATIONS function.

You can also use routines in the DBMS_DATA_MINING_TRANSFORM package to construct a transformation list.

See Also:

“About Transformation Lists” in DBMS_DATA_MINING_TRANSFORM

"GET_MODEL_TRANSFORMATIONS Function"

"CREATE_MODEL Procedure"
Syntax

DBMS_DATA_MINING.GET_TRANSFORM_LIST (  
  xform_list           OUT NOCOPY TRANSFORM_LIST,  
  model_xforms         IN  DM_TRANSFORMS);

Parameters

Table 43-107  GET_TRANSFORM_LIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xform_list</td>
<td>A list of transformation specifications that can be embedded in a model. Accepted as a parameter to the CREATE_MODEL Procedure. The TRANSFORM_LIST type is a table of TRANSFORM_REC objects. Each TRANSFORM_REC has these fields: attribute_name VARCHAR2(30) attribute_subname VARCHAR2(4000) expression EXPRESSION_REC reverse_expression EXPRESSION_REC attribute_spec VARCHAR2(4000) For details about the TRANSFORM_LIST collection type, see Table 44-1.</td>
</tr>
<tr>
<td>model_xforms</td>
<td>A list of embedded transformation expressions returned by the GET_MODEL_TRANSFORMATIONS Function for a specific model. The DM_TRANSFORMS type is a table of DM_TRANSFORM objects. Each DM_TRANSFORM has these fields: attribute_name VARCHAR2(4000) attribute_subname VARCHAR2(4000) expression CLOB reverse_expression CLOB</td>
</tr>
</tbody>
</table>

Examples

In this example, a model mod1 is trained using several columns in the SH.CUSTOMERS table. The model uses ADP, which automatically bins one of the columns.

A second model mod2 is trained on the same data without ADP, but it uses a transformation list that was obtained from mod1. As a result, both mod1 and mod2 have the same embedded transformation expression.

CREATE OR REPLACE VIEW mining_data AS  
SELECT cust_id, cust_year_of_birth, cust_income_level, cust_credit_limit  
FROM sh.customers;

describe mining_data
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
CUST_ID                                   NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
CUST_INCOME_LEVEL                                  VARCHAR2(30)
CUST_CREDIT_LIMIT                                  NUMBER

CREATE TABLE setmod1(setting_name VARCHAR2(30),setting_value VARCHAR2(30));
BEGIN
INSERT INTO setmod1 VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
INSERT INTO setmod1 VALUES (dbms_data_mining.prep_auto, dbms_data_mining.prep_auto_on);

dbms_data_mining.CREATE_MODEL(
    model_name => 'mod1',
    mining_function => dbms_data_mining.classification,
    data_table_name => 'mining_data',
    case_id_column_name => 'cust_id',
    target_column_name => 'cust_income_level',
    settings_table_name => 'setmod1');

COMMIT;
END;
/

CREATE TABLE setmod2(setting_name VARCHAR2(30), setting_value VARCHAR2(30));
BEGIN
    INSERT INTO setmod2
    VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
    COMMIT;
END;
/

DECLARE
    v_xform_list dbms_data_mining_transform.TRANSFORM_LIST;
    dmxf DM_TRANSFORMS;
BEGIN
    EXECUTE IMMEDIATE
    'SELECT dm_transform(attribute_name, attribute_subname, expression, reverse_expression)
    FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS ('mod1'))'
    BULK COLLECT INTO dmxf;
    dbms_data_mining.GET_TRANSFORM_LIST(
        xform_list => v_xform_list,
        model_xforms => dmxf);
    dbms_data_mining.CREATE_MODEL(
        model_name => 'mod2',
        mining_function => dbms_data_mining.classification,
        data_table_name => 'mining_data',
        case_id_column_name => 'cust_id',
        target_column_name => 'cust_income_level',
        settings_table_name => 'setmod2',
        xform_list => v_xform_list);
END;
/

-- Transformation expression embedded in mod1
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));

TO_CHAR(EXPRESSION)
-------------------------------------------------------------------------------
CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
....
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

-- Transformation expression embedded in mod2
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));

TO_CHAR(EXPRESSION)
-------------------------------------------------------------------------------
CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
....

43-144
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

-- Reverse transformation expression embedded in mod1
SELECT TO_CHAR(reverse_expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));

TO_CHAR(REVERSE_EXPRESSION)
--------------------------------------------------------------------------------
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915), [1915; 1915]',1,'(1915; 1920.5]',2,'(1920.5; 1924.5]','(1924.5; 1928.5]','(1928.5; 1932.5]','(1932.5; 1936.5]','(1936.5; 1940.5]','(1940.5; 1944.5]','(1944.5; 1948.5]','(1948.5; 1952.5]','(1952.5; 1956.5]','(1956.5; 1960.5]','(1960.5; 1964.5]','(1964.5; 1968.5]','(1968.5; 1972.5]','(1972.5; 1976.5]','(1976.5; 1980.5]','(1980.5; 1984.5]','(1984.5; 1988.5]','(1988.5; 1992.5]','(1992.5; 1996.5]','(1996.5; 2000.5]','(2000.5; 2004.5]','(2004.5; 2008.5]','(2008.5; 2012.5]','(2012.5; )',NULL,'NULL')

-- Reverse transformation expression embedded in mod2
SELECT TO_CHAR(reverse_expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));

TO_CHAR(REVERSE_EXPRESSION)
--------------------------------------------------------------------------------
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915), [1915; 1915]',1,'(1915; 1920.5]',2,'(1920.5; 1924.5]','(1924.5; 1928.5]','(1928.5; 1932.5]','(1932.5; 1936.5]','(1936.5; 1940.5]','(1940.5; 1944.5]','(1944.5; 1948.5]','(1948.5; 1952.5]','(1952.5; 1956.5]','(1956.5; 1960.5]','(1960.5; 1964.5]','(1964.5; 1968.5]','(1968.5; 1972.5]','(1972.5; 1976.5]','(1976.5; 1980.5]','(1980.5; 1984.5]','(1984.5; 1988.5]','(1988.5; 1992.5]','(1992.5; 1996.5]','(1996.5; 2000.5]','(2000.5; 2004.5]','(2004.5; 2008.5]','(2008.5; 2012.5]','(2012.5; )',NULL,'NULL')

43.7.40 IMPORT_MODEL Procedure

This procedure imports one or more data mining models. The procedure is overloaded. You can call it to import mining models from a dump file set, or you can call it to import a single mining model from a PMML document.

Import from a dump file set

You can import mining models from a dump file set that was created by the EXPORT_MODEL Procedure. IMPORT_MODEL and EXPORT_MODEL use Oracle Data Pump technology to export to and import from a dump file set.

When Oracle Data Pump is used directly to export/import an entire schema or database, the mining models in the schema or database are included. EXPORT_MODEL and IMPORT_MODEL export/import mining models only.

Import from PMML

You can import a mining model represented in Predictive Model Markup Language (PMML). The model must be of type RegressionModel, either linear regression or binary logistic regression.

PMML is an XML-based standard specified by the Data Mining Group (https://dmg.org). Applications that are PMML-compliant can deploy PMML-compliant models that were created by any vendor. Oracle Data Mining supports the core features of PMML 3.1 for regression models.
Syntax

Imports a mining model from a dump file set:

```sql
DBMS_DATA_MINING.IMPORT_MODEL (
    filename          IN  VARCHAR2,
    directory         IN  VARCHAR2,
    model_filter      IN  VARCHAR2 DEFAULT NULL,
    operation         IN  VARCHAR2 DEFAULT NULL,
    remote_link       IN  VARCHAR2 DEFAULT NULL,
    jobname           IN  VARCHAR2 DEFAULT NULL,
    schema_remap      IN  VARCHAR2 DEFAULT NULL,
    tablespace_remap  IN  VARCHAR2 DEFAULT NULL);
```

Imports a mining model from a PMML document:

```sql
DBMS_DATA_MINING.IMPORT_MODEL (
    model_name        IN  VARCHAR2,
    pmml_doc           IN  XMLTYPE
    strict_check      IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 43-108  IMPORT_MODEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name of the dump file set from which the models should be imported. The dump file set must have been created by the EXPORT_MODEL procedure or the expdp export utility of Oracle Data Pump. The dump file set can contain one or more files. (Refer to &quot;EXPORT_MODEL Procedure&quot; for details.) If the dump file set contains multiple files, you can specify 'filename%U' instead of listing them. For example, if your dump file set contains 3 files, archive01.dmp, archive02.dmp, and archive03.dmp, you can import them by specifying 'archive%U'.</td>
</tr>
<tr>
<td>directory</td>
<td>Name of a pre-defined directory object that specifies where the dump file set is located. Both the exporting and the importing user must have read/write access to the directory object and to the file system directory that it identifies. Note: The target database must have also have read/write access to the file system directory.</td>
</tr>
</tbody>
</table>
Table 43-108  (Cont.) IMPORT_MODEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_filter</td>
<td>Optional parameter that specifies one or more models to import. If you do not specify a value for model_filter, all models in the dump file set are imported. You can also specify NULL (the default) or 'ALL' to import all models. The value of model_filter can be one or more model names. The following are valid filters. 'mymodel1' 'name IN (''mymodel2'',''mymodel3'')' The first causes IMPORT_MODEL to import a single model named mymodel1. The second causes IMPORT_MODEL to import two models, mymodel2 and mymodel3.</td>
</tr>
</tbody>
</table>
| operation   | Optional parameter that specifies whether to import the models or the SQL statements that create the models. By default, the models are imported. You can specify either of the following values for operation:  
  - 'IMPORT' — Import the models (Default)  
  - 'SQL_FILE' — Write the SQL DDL for creating the models to a text file. The text file is named job_name.sql and is located in the dump set directory. |
| remote_link | Optional parameter that specifies the name of a database link to a remote system. The default value is NULL. A database link is a schema object in a local database that enables access to objects in a remote database. When you specify a value for remote_link, you can import models into the local database from the remote database. The import is fileless; no dump file is involved. The IMP_FULL_DATABASE role is required for importing the remote models. The EXP_FULL_DATABASE privilege, the CREATE DATABASE LINK privilege, and other privileges may also be required. |
| jobname     | Optional parameter that specifies the name of the import job. By default, the name has the form username_imp_nnnn, where nnnn is a number. For example, a job name in the SCOTT schema might be SCOTT_imp_134. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the import job, named jobname.log, is created in the same directory as the dump file set. |
| schema_remap| Optional parameter for importing into a different schema. By default, models are exported and imported within the same schema. If the dump file set belongs to a different schema, you must specify a schema mapping in the form export_user:import_user. For example, you would specify 'SCOTT:MARY' to import a model exported by SCOTT into the MARY schema. Note: In some cases, you may need to have the IMP_FULL_DATABASE privilege or the SYS role to import a model from a different schema. |
Table 43-108  (Cont.) IMPORT_MODEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_remap</td>
<td>Optional parameter for importing into a different tablespace. By default, models are exported and imported within the same tablespace.</td>
</tr>
<tr>
<td></td>
<td>If the dump file set belongs to a different tablespace, you must specify a tablespace mapping in the form export_tablespace:import_tablespace. For example, you would specify 'TBLSPC01:TBLSPC02' to import a model that was exported from tablespace TBLSPC01 into tablespace TBLSPC02.</td>
</tr>
<tr>
<td>model_name</td>
<td>Name for the new model that will be created in the database as a result of an import from PMML. The name must be unique within the user’s schema.</td>
</tr>
<tr>
<td>pmmldoc</td>
<td>The PMML document representing the model to be imported. The PMML document has an XMLTYPE object type. See &quot;XML TYPE&quot; for details.</td>
</tr>
<tr>
<td>strict_check</td>
<td>Whether or not an error occurs when the PMML document contains sections that are not part of core PMML (for example, Output or Targets). Oracle Data Mining supports only core PMML; any non-core features may affect the scoring representation. If the PMML does not strictly conform to core PMML and strict_check is set to TRUE, then IMPORT_MODEL returns an error. If strict_check is FALSE (the default), then the error is suppressed. The model may be imported and scored.</td>
</tr>
</tbody>
</table>

Examples

1. This example shows a model being exported and imported within the schema dmuser2. Then the same model is imported into the dmuser3 schema. The dmuser3 user has the IMP_FULL_DATABASE privilege. The dmuser2 user has been assigned the USER2 tablespace; dmuser3 has been assigned the USER3 tablespace.

   SQL> connect dmuser2
   Enter password: dmuser2_password
   Connected.
   SQL> select model_name from user_mining_models;

   MODEL_NAME
   ------------------------
   NMF_SH_SAMPLE
   SVMO_SH_CLAS_SAMPLE
   SVMR_SH_REGR_SAMPLE

   -- export the model called NMF_SH_SAMPLE to a dump file in same schema
   SQL> EXECUTE DBMS_DATA_MINING.EXPORT_MODEL (filename =>'NMF_SH_SAMPLE_out',
                                            directory =>'DATA_PUMP_DIR',
                                            model_filter => 'name = ''NMF_SH_SAMPLE''');

   -- import the model back into the same schema
   SQL> EXECUTE DBMS_DATA_MINING.IMPORT_MODEL (filename => 'NMF_SH_SAMPLE_out01.dmp',
                                             directory => 'DATA_PUMP_DIR',
                                             model_filter => 'name = ''NMF_SH_SAMPLE''');
-- connect as different user
-- import same model into that schema
SQL> connect dmuser3
Enter password: dmuser3_password
Connected.
SQL> EXECUTE DBMS_DATA_MINING.IMPORT_MODEL (filename => 'NMF_SH_SAMPLE_out01.dmp',
directory => 'DATA_PUMP_DIR',
model_filter => 'name = ''NMF_SH_SAMPLE''',
operation =>'IMPORT',
remote_link => NULL,
jobname => 'nmf_imp_job',
schema_remap => 'dmuser2:dmuser3',
tablespace_remap => 'USER2:USER3');
The following example shows user MARY importing all models from a dump file,
model_exp_001.dmp, which was created by user SCOTT. User MARY has been assigned a
tablespace named USER2; user SCOTT was assigned the tablespace USERS when the
models were exported into the dump file model_exp_001.dmp. The dump file is located in
the file system directory mapped to a directory object called DM_DUMP. If user MARY does
not have IMP_FULL_DATABASE privileges, IMPORT_MODEL will raise an error.

-- import all models
DECLARE
  file_name  VARCHAR2(40);
BEGIN
  file_name := 'model_exp_001.dmp';
  DBMS_DATA_MINING.IMPORT_MODEL(
    filename=> 'file_name',
    directory=>'DM_DUMP',
    schema_remap=>'SCOTT:MARY',
    tablespace_remap=>'USERS:USER2');
  DBMS_OUTPUT.PUT_LINE(
    'DBMS_DATA_MINING.IMPORT_MODEL of all models from SCOTT done!');
END;
/

2. This example shows how the user xuser could import the model dmuser.r1mod from a
remote database. The SQL*Net connection alias for the remote database is R1DB. The
user xuser is assigned the SYSAUX tablespace; the user dmuser is assigned the TBS_1
tablespace.

CONNECT / AS SYSDBA;
GRANT CREATE DATABASE LINK TO xuser;
GRANT imp_full_database TO xuser;
CONNECT xuser/xuserpassword
CREATE DATABASE LINK dmuser_link
  CONNECT TO dmuser IDENTIFIED BY dmuserpassword USING 'R1DB';
EXEC dbms_data_mining.import_model (null,
  'DMUSER_DIR',
  'R1MOD',
  remote_link => 'DMUSER_LINK', schema_remap => 'DMUSER:XUSER',
  tablespace_remap => 'TBS_1:SYSAUX');
SELECT name FROM dm_user_models;

NAME
-----------------------------------------------
R1MOD
3. This example shows how a PMML document called `SamplePMML1.xml` could be imported from a location referenced by directory object `PMMLDIR` into the schema of the current user. The imported model will be called `PMMLMODEL1`.

```
BEGIN
    dbms_data_mining.import_model('PMMLMODEL1',
        XMLType (bfilename('PMMLDIR', 'SamplePMML1.xml'),
            nls_charset_id ('AL32UTF8')));
END;
```

### 43.7.41 IMPORT_SERMODEL Procedure

This procedure imports the serialized format of the model back into a database.

The import routine takes the serialized content in the `BLOB` and the name of the model to be created with the content. This import does not create model views or tables that are needed for querying model details. The import procedure only provides the ability to score the model.

**Syntax**

```
DBMS_DATA_MINING.IMPORT_SERMODEL (
    model_data IN BLOB,
    model_name IN VARCHAR2);
```

**Parameters**

**Table 43-109 IMPORT_SERMODEL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_data</td>
<td>Provides model data in BLOB format.</td>
</tr>
<tr>
<td>model_name</td>
<td>Name of the mining model in the form <code>[schema_name.]model_name</code>. If you do not specify a schema, then your own schema is used.</td>
</tr>
</tbody>
</table>

**Examples**

The following statement imports the serialized format of the models.

```
declare
    v_blob blob;
BEGIN
    dbms_lob.createtemporary(v_blob, FALSE);
    -- fill in v_blob from somewhere (e.g., bfile, etc.)
    dbms_data_mining.import_sermodel(v_blob, 'MY_MODEL');
    dbms_lob.freetemporary(v_blob);
END;
/```
Related Topics

- **EXPORT_SERMODEL Procedure**
  This procedure exports the model in a serialized format so that they can be moved to another platform for scoring.

---

See Also:

*Oracle Data Mining User’s Guide* for more information about exporting and importing mining models

---

### 43.7.42 JSON Schema for R Extensible Algorithm

Follow JSON schema when creating a new JSON object with flexibility.

**Usage Note**

Some flexibility when creating a new JSON object are as follows:

- Partial registration is allowed. For example, detail function can be missing.
- Different orders are allowed. For example, detail function can be written before build function or after the build function.

**Example 43-1  JSON Schema**

JSON schema 1.1 for R extensible algorithm:

```json
{
    "type": "object",
    "properties": {
        "algo_name_display": { "type": "object",
            "properties": {
                "language": { "type": "string",
                    "enum": ["English", "Spanish", "French"],
                    "default": "English"},
                "name": { "type": "string"}
            }
        },
        "function_language": { "type": "string" },
        "mining_function": { 
            "type": "array",
            "items": [{
                "type": "object",
                "properties": {
                    "mining_function_name": { "type": "string"},
                    "build_function": { 
                        "type": "object",
                        "properties": {
                            "function_body": { "type": "CLOB" }
                        }
                    }
                }
            }]
        }
    }
}
```
Chapter 43
Summary of DBMS_DATA_MINING Subprograms

"detail_function": {
    "type": "array",
    "items": [
        {"type": "object",
         "properties": {
             "function_body": { "type": "CLOB" },
             "view_columns": { "type": "array",
                 "items": [
                     {"type": "object",
                     "properties": {
                         "name": { "type": "string"},
                         "type": { "type": "string",
                         "enum": ["VARCHAR2",
                              "NUMBER",
                              "DATE",
                              "BOOLEAN"]
                     }]
                 }
             }
         }
        }
    ],
    "score_function": {
        "type": "object",
        "properties": {
            "function_body": { "type": "CLOB" }
        }
    },
    "weight_function": {
        "type": "object",
        "properties": {
            "function_body": { "type": "CLOB" }
        }
    },
    "algo_setting": {
        "type": "array",
        "items": [
            {"type": "object",
            "properties": {
                "function_body": { "type": "CLOB" }
            }]
        }
    }
},
"items": [
  { "type": "object",
    "properties": {
      "name": { "type": "string" },
      "name_display": { "type": "object",
          "properties": {
            "language": { "type": "string",
              "enum": ["English", "Spanish", "French"],
              "default": "English" },
            "name": { "type": "string" } },
        "type": { "type": "string",
          "enum": ["string", "integer",
            "number", "boolean"] },
        "optional": { "type": "BOOLEAN",
          "default": "FALSE" },
        "value": { "type": "string" },
        "min_value": { "type": "object",
          "properties": {
            "min_value": { "type": "number" },
            "inclusive": { "type": "BOOLEAN",
              "default": "TRUE" } },
        "max_value": { "type": "object",
          "properties": {
            "max_value": { "type": "number" },
            "inclusive": { "type": "BOOLEAN",
              "default": "TRUE" } },
        "categorical choices": { "type": "array",
          "items": { "type": "string" } },
        "description_display": { "type": "object",
          "properties": { } } } }
}]}
Example 43-2  JSON object example

The following is an JSON object example that must be passed to the registration procedure:

```json
{
  "algo_name_display" : { "English", "t1"},
  "function_language" : "R",
  "mining_function" : {
    "mining_function_name" : "CLASSIFICATION",
    "build_function" : { "function_body": "function(dat, formula, family) {

    set.seed(1234);

    mod <- glm(formula =
                formula, data=dat,
                family=
                eval(parse(text=family)));

    "score_function" : { "function_body": "function(mod, dat) {
          res <- predict(mod,
          newdata = dat,
          type='response

          res2=data.frame(1-res,
          res); res2ditract)"})

    },

    "algo_setting" : ["name" : "dbms_data_mining.odms_missing_value_treatment",

    "name_display" : { "English",
    "dbms_data_mining.odms_missing_value_treatment"},
    "type" : "string",
    "optional" : "TRUE",
    "value" :
```
"dbms_data_mining.odms_missing_value_mean_mode",
"categorical choices" : [
  "dbms_data_mining.odms_missing_value_mean_mode",
  "dbms_data_mining.odms_missing_value_auto",
  "dbms_data_mining.odms_missing_value_delete_row",
  "description" : {"English", "how to treat missing values"}
],
{"name" : "RALG_PARAMETER_FAMILY",
  "name_display" : {"English", "RALG_PARAMETER_FAMILY"},
  "type" : "string",
  "optional" : "TRUE",
  "value" : "",
  "description" : {"English", "R family parameter in build function"}
},
]

43.7.43 REGISTER_ALGORITHM Procedure

User can register a new algorithm by providing algorithm name, mining function, and all other algorithm metadata to this function.

Syntax

```
DBMS_DATA_MINING.REGISTER_ALGORITHM ( 
  algorithm_name IN VARCHAR2, 
  algorithm_metadata IN CLOB, 
  algorithm_description IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm_name</td>
<td>Name of the algorithm.</td>
</tr>
<tr>
<td>algorithm_metadata</td>
<td>Metadata of the algorithm.</td>
</tr>
<tr>
<td>algorithm_description</td>
<td>Description of the algorithm.</td>
</tr>
</tbody>
</table>

Usage Notes

The registration procedure performs the following:

- Checks whether `algorithm_metadata` has correct JSON syntax.
- Checks whether the input JSON object follows the predefined JSON schema.
- Checks whether current user has `RQADMIN` privilege.
• Checks duplicate algorithms such that the same algorithm is not registered twice.
• Checks for missing entries. For example, algorithm name, algorithm type, metadata, and build function.

Register Algorithms After the JSON Object Is Created

SQL users can register new algorithms by following the given procedure:

Create a JSON object following JSON schema and pass it to `REGISTER_ALGORITHM` procedure.

```sql
BEGIN
    DBMS_DATA_MINING.register_algorithm(
        algorithm_name                 => 't1',
        algorithm_metadata            =>
            '{
                "function_language": "R",
                "mining_function": {
                    "mining_function_name": "CLASSIFICATION",
                    "build_function": {
                        "function_body": "function(dat,
                                         formula, family) {
                           set.seed(1234); mod <- glm(formula =
                                         family=data.frame(l-res,
                                                                      res2); res2)
                        }
                    },
                    "score_function": {
                        "function_body": "function(mod, dat) {
                           res <- predict(mod,
                                          newdata = dat, type='response'); res2=data.frame(1-res,
                                          res2)
                        }
                    }
                }
            }',
        algorithm_description  => 't1');
END;
/
```

43.7.44 RANK_APPLY Procedure

This procedure ranks the results of an APPLY operation based on a top-N specification for predictive and descriptive model results.

For classification models, you can provide a cost matrix as input, and obtain the ranked results with costs applied to the predictions.

**Syntax**

```sql
DBMS_DATA_MINING.RANK_APPLY (    apply_result_table_name        IN VARCHAR2,
    case_id_column_name            IN VARCHAR2,
    score_column_name              IN VARCHAR2,
    score_criterion_column_name    IN VARCHAR2,
    ranked_apply_table_name        IN VARCHAR2,
    top_N                          IN NUMBER (38) DEFAULT 1,
    cost_matrix_table_name         IN VARCHAR2 DEFAULT NULL,
    apply_result_schema_name       IN VARCHAR2 DEFAULT NULL,
    cost_matrix_schema_name        IN VARCHAR2 DEFAULT NULL);```
Parameters

Table 43-111  RANK_APPLY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_result_table_name</td>
<td>Name of the table or view containing the results of an APPLY operation on the test data set (see Usage Notes)</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Name of the case identifier column. This must be the same as the one used for generating APPLY results.</td>
</tr>
<tr>
<td>score_column_name</td>
<td>Name of the prediction column in the apply results table</td>
</tr>
<tr>
<td>score_criterion_column_name</td>
<td>Name of the probability column in the apply results table</td>
</tr>
<tr>
<td>ranked_apply_result_tab_name</td>
<td>Name of the table containing the ranked apply results</td>
</tr>
<tr>
<td>top_N</td>
<td>Top N predictions to be considered from the APPLY results for precision recall computation</td>
</tr>
<tr>
<td>cost_matrix_table_name</td>
<td>Name of the cost matrix table</td>
</tr>
<tr>
<td>apply_result_schema_name</td>
<td>Name of the schema hosting the APPLY results table</td>
</tr>
<tr>
<td>cost_matrix_schema_name</td>
<td>Name of the schema hosting the cost matrix table</td>
</tr>
</tbody>
</table>

Usage Notes

You can use RANK_APPLY to generate ranked apply results, based on a top-N filter and also with application of cost for predictions, if the model was built with costs.

The behavior of RANK_APPLY is similar to that of APPLY with respect to other DDL-like operations such as CREATE_MODEL, DROP_MODEL, and RENAME_MODEL. The procedure does not depend on the model; the only input of relevance is the apply results generated in a fixed schema table from APPLY.

The main intended use of RANK_APPLY is for the generation of the final APPLY results against the scoring data in a production setting. You can apply the model against test data using APPLY, compute various test metrics against various cost matrix tables, and use the candidate cost matrix for RANK_APPLY.

The schema for the apply results from each of the supported algorithms is listed in subsequent sections. The case_id column will be the same case identifier column as that of the apply results.

Classification Models — NB and SVM

For numerical targets, the ranked results table will have the definition as shown:

```sql
(case_id VARCHAR2/NUMBER,
prediction NUMBER,
probability NUMBER,
cost NUMBER,
rank INTEGER)
```

For categorical targets, the ranked results table will have the following definition:
Clustering Using \textit{k}-Means or O-Cluster

Clustering is an unsupervised mining function, and hence there are no targets. The results of an \texttt{APPLY} operation contains simply the cluster identifier corresponding to a case, and the associated probability. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the cluster ids ranked by top-N.

\[
\begin{align*}
\text{(case_id: VARCHAR2/NUMBER,} \\
\text{cluster_id: NUMBER,} \\
\text{probability: NUMBER,} \\
\text{rank: INTEGER)}
\end{align*}
\]

Feature Extraction using NMF

Feature extraction is also an unsupervised mining function, and hence there are no targets. The results of an \texttt{APPLY} operation contains simply the feature identifier corresponding to a case, and the associated match quality. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the feature ids ranked by top-N.

\[
\begin{align*}
\text{(case_id: VARCHAR2/NUMBER,} \\
\text{feature_id: NUMBER,} \\
\text{match_quality: NUMBER,} \\
\text{rank: INTEGER)}
\end{align*}
\]

Examples

\[
\begin{align*}
\text{BEGIN} \\
\text{/* build a model with name census_model.} \\
\text{*(See example under CREATE_MODEL)*/} \\
\text{/* if training data was pre-processed in any manner,} \\
\text{* perform the same pre-processing steps on apply} \\
\text{* data also.} \\
\text{*(See examples in the section on DBMS_DATA_MINING_TRANSFORM)*/} \\
\text{/* apply the model to data to be scored */} \\
\text{DBMS\_DATA\_MINING.RANK\_APPLY(} \\
\text{\hspace{1cm}apply\_result\_table\_name \hspace{1cm}=> \texttt{'census\_apply\_result'},} \\
\text{\hspace{1cm}case\_id\_column\_name \hspace{1cm}=> \texttt{'person\_id'},} \\
\text{\hspace{1cm}score\_column\_name \hspace{1cm}=> \texttt{'prediction'},} \\
\text{\hspace{1cm}score\_criterion\_column\_name \hspace{1cm}=> \texttt{'probability'}} \\
\text{\hspace{1cm}ranked\_apply\_result\_tab\_name \hspace{1cm}=> \texttt{'census\_ranked\_apply\_result'},} \\
\text{\hspace{1cm}top\_N \hspace{1cm}=> \texttt{3},} \\
\text{\hspace{1cm}cost\_matrix\_table\_name \hspace{1cm}=> \texttt{'census\_cost\_matrix'}} \\
\text{);} \\
\text{END;} \\
\text{/}
\end{align*}
\]

-- View Ranked Apply Results
\[
\begin{align*}
\text{SELECT *} \\
\text{FROM census\_ranked\_apply\_result;}
\end{align*}
\]
43.7.45 REMOVE_COST_MATRIX Procedure

The **REMOVE_COST_MATRIX** procedure removes the default scoring matrix from a classification model.

### See Also:

- "ADD_COST_MATRIX Procedure"
- "REMOVE_COST_MATRIX Procedure"

### Syntax

```sql
DBMS_DATA_MINING.REMOVE_COST_MATRIX (
    model_name   IN  VARCHAR2);
```

### Parameters

**Table 43-112 Remove_Cost_Matrix Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Name of the model in the form <code>[schema_name.]model_name</code>. If you do not specify a schema, your own schema is used.</td>
</tr>
</tbody>
</table>

### Usage Notes

If the model is not in your schema, then **REMOVE_COST_MATRIX** requires the `ALTER ANY MINING MODEL` system privilege or the `ALTER` object privilege for the mining model.

### Example

The Naïve Bayes model **NB_SH_CLAS_SAMPLE** has an associated cost matrix that can be used for scoring the model.

```sql
SQL> SELECT *
FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
ORDER BY predicted, actual;
```

<table>
<thead>
<tr>
<th>ACTUAL</th>
<th>PREDICTED</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>.75</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>.25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

You can remove the cost matrix with **REMOVE_COST_MATRIX**.

```sql
SQL> EXECUTE dbms_data_mining.remove_cost_matrix('nb_sh_clas_sample');
```

```sql
SQL> SELECT *
FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
ORDER BY predicted, actual;
```
43.7.46 RENAME_MODEL Procedure

This procedure changes the name of the mining model indicated by model_name to the name that you specify as new_model_name.

If a model with new_model_name already exists, then the procedure optionally renames new_model_name to versioned_model_name before renaming model_name to new_model_name.

The model name is in the form [schema_name.model_name]. If you do not specify a schema, your own schema is used. For mining model naming restrictions, see the Usage Notes for "CREATE_MODEL Procedure".

Syntax

```sql
DBMS_DATA_MINING.RENAME_MODEL (
    model_name            IN VARCHAR2,
    new_model_name        IN VARCHAR2,
    versioned_model_name  IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>Model to be renamed.</td>
</tr>
<tr>
<td>new_model_name</td>
<td>New name for the model model_name.</td>
</tr>
<tr>
<td>versioned_model_name</td>
<td>New name for the model new_model_name if it already exists.</td>
</tr>
</tbody>
</table>

Usage Notes

If you attempt to rename a model while it is being applied, then the model will be renamed but the apply operation will return indeterminate results.

Examples

1. This example changes the name of model census_model to census_model_2012.

   ```sql
   BEGIN
   DBMS_DATA_MINING.RENAME_MODEL(
       model_name => 'census_model',
       new_model_name => 'census_model_2012');
   END;
   /
   
2. In this example, there are two classification models in the user's schema: clas_mod, the working model, and clas_mod_tst, a test model. The RENAME_MODEL procedure preserves clas_mod as clas_mod_old and makes the test model the new working model.

   ```sql
   SELECT model_name FROM user_mining_models;
   MODEL_NAME
   ---------------------------------------------------------------
   CLAS_MOD
CLAS_MOD_TST

BEGIN
    DBMS_DATA_MINING.RENAME_MODEL(
        model_name => 'clas_mod_tst',
        new_model_name => 'clas_mod',
        versioned_model_name => 'clas_mod_old');
END;
/

SELECT model_name FROM user_mining_models;
MODEL_NAME
-----------------------------
CLAS_MOD
CLAS_MOD_OLD
DBMS_DATA_MINING_TRANSFORM implements a set of transformations that are commonly used in data mining.

This chapter contains the following topics:

- Overview
- Operational Notes
- Security Model
- Datatypes
- Constants
- Summary of DBMS_DATA_MINING_TRANSFORM Subprograms

See Also:

- DBMS_DATA_MINING
- Oracle Data Mining User’s Guide

44.1 DBMS_DATA_MINING_TRANSFORM Overview

A transformation is a SQL expression that modifies the data in one or more columns. Data must typically undergo certain transformations before it can be used to build a mining model. Many data mining algorithms have specific transformation requirements.

Data that will be scored must be transformed in the same way as the data that was used to create (train) the model.

External or Embedded Transformations

DBMS_DATA_MINING_TRANSFORM offers two approaches to implementing transformations. For a given model, you can either:

- Create a list of transformation expressions and pass it to the CREATE_MODEL Procedure
  or
- Create a view that implements the transformations and pass the name of the view to the CREATE_MODEL Procedure

If you create a transformation list and pass it to CREATE_MODEL, the transformation expressions are embedded in the model and automatically implemented whenever the model is applied.
If you create a view, the transformation expressions are external to the model. You will need to re-create the transformations whenever you apply the model.

**Note:**

Embedded transformations significantly enhance the model's usability while simplifying the process of model management.

Automatic Transformations

Oracle Data Mining supports an Automatic Data Preparation (ADP) mode. When ADP is enabled, most algorithm-specific transformations are automatically embedded. Any additional transformations must be explicitly provided in an embedded transformation list or in a view.

If ADP is enabled and you create a model with a transformation list, both sets of transformations are embedded. The model will execute the user-specified transformations from the transformation list before executing the automatic transformations specified by ADP.

Within a transformation list, you can selectively disable ADP for individual attributes.

**See Also:**

"Automatic Data Preparation" in DBMS_DATA_MINING Oracle Data Mining User's Guide for a more information about ADP

"DBMS_DATA_MINING_TRANSFORM-About Transformation Lists"

Transformations in DBMS_DATA_MINING_TRANSFORM

The transformations supported by DBMS_DATA_MINING_TRANSFORM are summarized in this section.

**Binning**

Binning refers to the mapping of continuous or discrete values to discrete values of reduced cardinality.

- Supervised Binning (Categorical and Numerical)
  
  Binning is based on intrinsic relationships in the data as determined by a decision tree model.
  
  See "INSERT_BIN_SUPER Procedure".

- Top-N Frequency Categorical Binning
  
  Binning is based on the number of cases in each category.
  
  See "INSERT_BIN_CAT_FREQ Procedure"

- Equi-Width Numerical Binning
Binning is based on equal-range partitions.
See "INSERT_BIN_NUM_EQWIDTH Procedure".

- Quantile Numerical Binning
  Binning is based on quantiles computed using the SQL `NTILE` function.
  See "INSERT_BIN_NUM_QTILE Procedure".

**Linear Normalization**

Normalization is the process of scaling continuous values down to a specific range, often between zero and one. Normalization transforms each numerical value by subtracting a number (the shift) and dividing the result by another number (the scale).

\[
x_{\text{new}} = \frac{x_{\text{old}} - \text{shift}}{\text{scale}}
\]

- Min-Max Normalization
  Normalization is based on the minimum and maximum with the following shift and scale:
  \[
  \text{shift} = \text{min} \\
  \text{scale} = \text{max} - \text{min}
  \]
  See "INSERT_NORM_LIN_MINMAX Procedure".

- Scale Normalization
  Normalization is based on the minimum and maximum with the following shift and scale:
  \[
  \text{shift} = 0 \\
  \text{scale} = \max\{\abs{\text{max}}, \abs{\text{min}}\}
  \]
  See "INSERT_NORM_LIN_SCALE Procedure".

- Z-Score Normalization
  Normalization is based on the mean and standard deviation with the following shift and scale:
  \[
  \text{shift} = \text{mean} \\
  \text{scale} = \text{standard deviation}
  \]
  See "INSERT_NORM_LIN_ZSCORE Procedure".

**Outlier Treatment**

An outlier is a numerical value that is located far from the rest of the data. Outliers can artificially skew the results of data mining.

- Winsorizing
  Outliers are replaced with the nearest value that is not an outlier.
  See "INSERT_CLIP_WINSOR_TAIL Procedure"

- Trimming
  Outliers are set to `NULL`.
  See "INSERT_CLIP_TRIM_TAIL Procedure".

**Missing Value Treatment**
Missing data may indicate sparsity or it may indicate that some values are missing at random. **DBMS_DATA_MINING_TRANSFORM** supports the following transformations for minimizing the effects of missing values:

- Missing numerical values are replaced with the mean.
  
  See "INSERT_MISS_NUM_MEAN Procedure".

- Missing categorical values are replaced with the mode.
  
  See "INSERT_MISS_CAT_MODE Procedure".

**Note:**

Oracle Data Mining also has default mechanisms for handling missing data. See *Oracle Data Mining User's Guide* for details.

### 44.2 DBMS_DATA_MINING_TRANSFORM Operational Notes

The **DBMS_DATA_MINING_TRANSFORM** package offers a flexible framework for specifying data transformations. If you choose to embed transformations in the model (the preferred method), you create a **transformation list** object and pass it to the **CREATE_MODEL Procedure**. If you choose to transform the data without embedding, you create a view.

When specified in a transformation list, the transformation expressions are executed by the model. When specified in a view, the transformation expressions are executed by the view.

**Transformation Definitions**

Transformation definitions are used to generate the SQL expressions that transform the data. For example, the transformation definitions for normalizing a numeric column are the shift and scale values for that data.

With the **DBMS_DATA_MINING_TRANSFORM** package, you can call procedures to compute the transformation definitions, or you can compute them yourself, or you can do both.

**Transformation Definition Tables**

**DBMS_DATA_MINING_TRANSFORM** provides **INSERT** procedures that compute transformation definitions and insert them in transformation definition tables. You can modify the values in the transformation definition tables or populate them yourself.

**XFORM** routines use populated definition tables to transform data in external views. **STACK** routines use populated definition tables to build transformation lists.

To specify transformations based on definition tables, follow these steps:

1. Use **CREATE** routines to create transformation definition tables.

   The tables have columns to hold the transformation definitions for a given type of transformation. For example, the **CREATE_BIN_NUM Procedure** creates a
definition table that has a column for storing data values and another column for storing the associated bin identifiers.

2. Use **INSERT** routines to compute and insert transformation definitions in the tables.

   Each **INSERT** routine uses a specific technique for computing the transformation definitions. For example, the **INSERT_BIN_NUM_EQWIDTH** Procedure computes bin boundaries by identifying the minimum and maximum values then setting the bin boundaries at equal intervals.

3. Use **STACK** or **XFORM** routines to generate transformation expressions based on the information in the definition tables:
   
   - Use **STACK** routines to add the transformation expressions to a transformation list. Pass the transformation list to the **CREATE_MODEL** Procedure. The transformation expressions will be assembled into one long SQL query and embedded in the model.
   
   - Use **XFORM** routines to execute the transformation expressions within a view. The transformations will be external to the model and will need to be re-created whenever the model is applied to new data.

Transformations Without Definition Tables

**STACK** routines are not the only method for adding transformation expressions to a transformation list. You can also build a transformation list without using definition tables.

To specify transformations without using definition tables, follow these steps:

1. Write a SQL expression for transforming an attribute.
2. Write a SQL expression for reversing the transformation. (See "Reverse Transformations and Model Transparency" in "DBMS_DATA_MINING_TRANSFORM-About Transformation Lists".)
3. Determine whether or not to disable ADP for the attribute. By default ADP is enabled for the attribute if it is specified for the model. (See "Disabling Automatic Data Preparation" in "DBMS_DATA_MINING_TRANSFORM - About Transformation Lists").
4. Specify the SQL expressions and ADP instructions in a call to the **SET_TRANSFORM** Procedure, which adds the information to a transformation list.
5. Repeat steps 1 through 4 for each attribute that you wish to transform.
6. Pass the transformation list to the **CREATE_MODEL** Procedure. The transformation expressions will be assembled into one long SQL query and embedded in the model.

**Note:**

SQL expressions that you specify with **SET_TRANSFORM** must fit within a **VARCHAR2**. To specify a longer expression, you can use the **SET_EXPRESSION** Procedure. With **SET_EXPRESSION**, you can build an expression by appending rows to a **VARCHAR2** array.

About Stacking

Transformation lists are built by stacking transformation records. Transformation lists are evaluated from bottom to top. Each transformation expression depends on the result of the transformation expression below it in the stack.
Related Topics

- **CREATE_MODEL Procedure**
  This procedure creates a mining model with a given mining function.

- **DBMS_DATA_MINING_TRANSFORM — About Transformation Lists**
  The elements of a transformation list are transformation records. Each transformation record provides all the information needed by the model for managing the transformation of a single attribute.

- **DBMS_DATA_MINING_TRANSFORM — About Stacking and Stack Procedures**
  Transformation lists are built by stacking transformation records. Transformation lists are evaluated from bottom to top. Each transformation expression depends on the result of the transformation expression below it in the stack.

- **DBMS_DATA_MINING_TRANSFORM — Nested Data Transformations**
  The CREATE routines create transformation definition tables that include two columns, col and att, for identifying attributes. The column col holds the name of a column in the data table. If the data column is not nested, then att is null, and the name of the attribute is col. If the data column is nested, then att holds the name of the nested attribute, and the name of the attribute is col.att.

### 44.2.1 DBMS_DATA_MINING_TRANSFORM — About Transformation Lists

The elements of a transformation list are transformation records. Each transformation record provides all the information needed by the model for managing the transformation of a single attribute.

Each transformation record includes the following fields:

- attribute_name — Name of the column of data to be transformed
- attribute_subname — Name of the nested attribute if attribute_name is a nested column, otherwise NULL
- expression — SQL expression for transforming the attribute
- reverse_expression — SQL expression for reversing the transformation
- attribute_spec — Identifies special treatment for the attribute during the model build. See Table 44-33 for details.

**See Also:**

- Table 44-1 for details about the TRANSFORM_LIST and TRANSFORM_REC object types
- SET_TRANSFORM Procedure
- CREATE_MODEL Procedure

### Reverse Transformations and Model Transparency

An algorithm manipulates transformed attributes to train and score a model. The transformed attributes, however, may not be meaningful to an end user. For example,
if attribute \( x \) has been transformed into bins 1 — 4, the bin names 1, 2, 3, and 4 are manipulated by the algorithm, but a user is probably not interested in the model details about bins 1 — 4 or in predicting the numbers 1 — 4.

To return original attribute values in model details and predictions, you can provide a reverse expression in the transformation record for the attribute. For example, if you specify the transformation expression 
\[
\log_{10}(y)
\]
for attribute \( y \), you could specify the reverse transformation expression 
\[
\text{power}(10, y)
\].

Reverse transformations enable model transparency. They make internal processing transparent to the user.

---

**Note:**

STACK procedures automatically reverse normalization transformations, but they do not provide a mechanism for reversing binning, clipping, or missing value transformations.

You can use the DBMS_DATA_MINING.ALTER_REVERSE_EXPRESSION procedure to specify or update reverse transformations expressions for an existing model.

---

**See Also:**

Table 44-1

"ALTER_REVERSE_EXPRESSION Procedure"

"Summary of DBMS_DATA_MINING Subprograms" for links to the model details functions

---

**Disabling Automatic Data Preparation**

ADP is controlled by a model-specific setting (PREP_AUTO). The PREP_AUTO setting affects all model attributes unless you disable it for individual attributes.

If ADP is enabled and you set attribute_spec to NOPREP, only the transformations that you specify for that attribute will be evaluated. If ADP is enabled and you do not set attribute_spec to NOPREP, the automatic transformations will be evaluated after the transformations that you specify for the attribute.

If ADP is not enabled for the model, the attribute_spec field of the transformation record is ignored.

---

**See Also:**

"Automatic Data Preparation" for information about the PREP_AUTO setting
Adding Transformation Records to a Transformation List

A transformation list is a stack of transformation records. When a new transformation record is added, it is appended to the top of the stack. (See “About Stacking” for details.)

When you use `SET_TRANSFORM` to add a transformation record to a transformation list, you can specify values for all the fields in the transformation record.

When you use `STACK` procedures to add transformation records to a transformation list, only the transformation expression field is populated. For normalization transformations, the reverse transformation expression field is also populated.

You can use both `STACK` procedures and `SET_TRANSFORM` to build one transformation list. Each `STACK` procedure call adds transformation records for all the attributes in a specified transformation definition table. Each `SET_TRANSFORM` call adds a transformation record for a single attribute.

44.2.2 DBMS_DATA_MINING_TRANSFORM — About Stacking and Stack Procedures

Transformation lists are built by stacking transformation records. Transformation lists are evaluated from bottom to top. Each transformation expression depends on the result of the transformation expression below it in the stack.

Stack Procedures

`STACK` procedures create transformation records from the information in transformation definition tables. For example, `STACK_BIN_NUM` builds a transformation record for each attribute specified in a definition table for numeric binning. `STACK` procedures stack the transformation records as follows:

- If an attribute is specified in the definition table but not in the transformation list, the `STACK` procedure creates a transformation record, computes the reverse transformation (if possible), inserts the transformation and reverse transformation in the transformation record, and appends the transformation record to the top of the transformation list.
- If an attribute is specified in the transformation list but not in the definition table, the `STACK` procedure takes no action.
- If an attribute is specified in the definition table and in the transformation list, the `STACK` procedure stacks the transformation expression from the definition table on top of the transformation expression in the transformation record and updates the reverse transformation. See Table 44-1 and Example 44-4.

Example 44-1  Stacking a Clipping Transformation

This example shows how `STACK_CLIP Procedure` would add transformation records to a transformation list. Note that the clipping transformations are not reversed in `COL1` and `COL2` after stacking (as described in “Reverse Transformations and Model Transparency” in “DBMS_DATA_MINING_TRANSFORM-About Transformation Lists”).

Refer to:

- `CREATE_CLIP Procedure` — Creates the definition table
• INSERT_CLIP_TRIM_TAIL Procedure — Inserts definitions in the table
• INSERT_CLIP_WINSOR_TAIL Procedure — Inserts definitions in the table
• Table 44-1 — Describes the structure of the transformation list (TRANSFORM_LIST object)

Assume a clipping definition table populated as follows.

<table>
<thead>
<tr>
<th>col</th>
<th>att</th>
<th>lcut</th>
<th>lval</th>
<th>rcut</th>
<th>rval</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL1</td>
<td>null</td>
<td>-1.5</td>
<td>-1.5</td>
<td>4.5</td>
<td>4.5</td>
</tr>
<tr>
<td>COL2</td>
<td>null</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Assume the following transformation list before stacking.

-------------------------
transformation record #1:
-------------------------
attribute_name       = COL1
attribute_subname    = null
expression           = log(10, COL1)
reverse_expression   = power(10, COL1)
-------------------------
transformation record #2:
-------------------------
attribute_name       = COL3
attribute_subname    = null
expression           = ln(COL3)
reverse_expression   = exp(COL3)
-------------------------
transformation record #3:
-------------------------
attribute_name       = COL2
attribute_subname    = null
expression           = CASE WHEN COL2 < 0 THEN 0 WHEN COL2 > 1 THEN 1 ELSE COL2 END;
reverse_expression   = null
-------------------------

After stacking, the transformation list is as follows.

-------------------------
transformation record #1:
-------------------------
attribute_name       = COL1
attribute_subname    = null
expression           = CASE WHEN log(10, COL1) < -1.5 THEN -1.5 WHEN log(10, COL1) > 4.5 THEN 4.5 ELSE log(10, COL1) END;
reverse_expression   = power(10, COL1)
-------------------------
transformation record #2:
-------------------------
attribute_name       = COL3
attribute_subname    = null
expression           = ln(COL3)
reverse_expression   = exp(COL3)
-------------------------
transformation record #3:
-------------------------
attribute_name       = COL2
attribute_subname    = null
expression           = CASE WHEN COL2 < 0 THEN 0 WHEN COL2 > 1 THEN 1 ELSE COL2 END;
reverse_expression   = null
-------------------------
The CREATE routines create transformation definition tables that include two columns, col and att, for identifying attributes. The column col holds the name of a column in the data table. If the data column is not nested, then att is null, and the name of the attribute is col. If the data column is nested, then att holds the name of the nested attribute, and the name of the attribute is col.att.

The INSERT and XFORM routines ignore the att column in the definition tables. Neither the INSERT nor the XFORM routines support nested data.

Only the STACK procedures and SET_TRANSFORM support nested data. Nested data transformations are always embedded in the model.

Nested columns in Oracle Data Mining can have the following types:

- DM_NESTED_NUMERICALS
- DM_NESTED_CATEGORICALS
- DM_NESTED_BINARY_DOUBLES
- DM_NESTED_BINARY_FLOATS

See Also:

"Constants"

Oracle Data Mining User's Guide for details about nested attributes in Oracle Data Mining

Specifying Nested Attributes in a Transformation Record

A transformation record (TRANSFORM_REC) includes two fields, attribute_name and attribute_subname, for identifying the attribute. The field attribute_name holds the name of a column in the data table. If the data column is not nested, then attribute_subname is null, and the name of the attribute is attribute_name. If the data column is nested, then attribute_subname holds the name of the nested attribute, and the name of the attribute is attribute_name.attribute_subname.

Transforming Individual Nested Attributes

You can specify different transformations for different attributes in a nested column, and you can specify a default transformation for all the remaining attributes in the column. To specify a default nested transformation, specify null in the attribute_name field and the name of the nested column in the attribute_subname field as shown in Example 44-2. Note that the keyword VALUE is used to represent the value of a nested attribute in a transformation expression.
Example 44-2  Transforming a Nested Column

The following statement transforms two of the nested attributes in COL_N1. Attribute ATTR1 is transformed with normalization; Attribute ATTR2 is set to null, which causes attribute removal transformation (ATTR2 is not used in training the model). All the remaining attributes in COL_N1 are divided by 10.

DECLARE
  stk dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(}
    stk,'COL_N1', 'ATTR1', '(VALUE - (-1.5))/20', 'VALUE *20 + (-1.5)');
  dbms_data_mining_transform.SET_TRANSFORM(}
    stk,'COL_N1', 'ATTR2', NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM(
    stk, NULL, 'COL_N1', 'VALUE/10', 'VALUE*10');
END;
/

The following SQL is generated from this statement.

CAST(MULTISET(SELECT DM_NESTED_NUMERICAL(
      "ATTRIBUTE_NAME",
      DECODE("ATTRIBUTE_NAME",
        'ATTR1', ("VALUE" - (-1.5))/20,
        "VALUE"/10))
  FROM TABLE("COL_N1")
WHERE "ATTRIBUTE_NAME" IS NOT IN ('ATTR2'))
AS DM_NESTED_NUMERICALS)        

If transformations are not specified for COL_N1.ATTR1 and COL_N1.ATTR2, then the default transformation is used for all the attributes in COL_N1, and the resulting SQL does not include a DECODE.

CAST(MULTISET(SELECT DM_NESTED_NUMERICAL(
      "ATTRIBUTE_NAME",
      "VALUE"/10)
  FROM TABLE("COL_N1")
AS DM_NESTED_NUMERICALS)        

Since DECODE is limited to 256 arguments, multiple DECODE functions are nested to support an arbitrary number of individual nested attribute specifications.

Adding a Nested Column

You can specify a transformation that adds a nested column to the data, as shown in Example 44-3.

Example 44-3  Adding a Nested Column to a Transformation List

DECLARE
  v_xlst dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(v_xlst,
    'YOB_CREDLIM', NULL, ''dm_nested_numericals(
      dm_nested_numerical('CUST_YEAR_OF_BIRTH', cust_year_of_birth),
      dm_nested_numerical('CUST_CREDIT_LIMIT', cust_credit_limit)),
    NULL);

Stacking Nested Transformations

Example 44-4 shows how the STACK_NORM_LIN Procedure would add transformation records for nested column COL_N to a transformation list.

Refer to:

- CREATE_NORM_LIN Procedure — Creates the definition table
- INSERT_NORM_LIN_MINMAX Procedure — Inserts definitions in the table
- INSERT_NORM_LIN_SCALE Procedure — Inserts definitions in the table
- INSERT_NORM_LIN_ZSCORE Procedure — Inserts definitions in the table
- Table 44-1 — Describes the structure of the transformation list

Example 44-4  Stacking a Nested Normalization Transformation

Assume a linear normalization definition table populated as follows.

<table>
<thead>
<tr>
<th>col</th>
<th>att</th>
<th>shift</th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL_N</td>
<td>ATT2</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>null</td>
<td>COL_N</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

Assume the following transformation list before stacking.

---------------------------------
transformation record #1:        
---------------------------------
attribute_name = COL_N
attribute_subname = ATT1
expression = log(10, VALUE)
reverse_expression = power(10, VALUE)

transformation record #2:
attribute_name = null
attribute_subname = COL_N
expression = ln(VALUE)
reverse_expression = exp(VALUE)

After stacking, the transformation list is as follows.

transformation record #1:
attribute_name = COL_N
attribute_subname = ATT1
expression = (log(10, VALUE) - 0)/10
reverse_expression = power(10, VALUE*10 + 0)
transformation record #2:
attribute_name = null
attribute_subname = COL_N
expression = (ln(VALUE) - 0)/10
reverse_expression = exp(VALUE *10 + 0)
transformation record #3:
attribute_name = COL_N
attribute_subname = ATT2
expression = (ln(VALUE) - 0)/20
reverse_expression = exp(VALUE * 20 + 0)

44.3 DBMS_DATA_MINING_TRANSFORM Security Model

The DBMS_DATA_MINING_TRANSFORM package is owned by user SYS and is installed as part of
database installation. Execution privilege on the package is granted to public. The routines in
the package are run with invokers’ rights (run with the privileges of the current user).

The DBMS_DATA_MINING_TRANSFORM.INSERT_* procedures have a data_table_name parameter
that enables the user to provide the input data for transformation purposes. The value of
data_table_name can be the name of a physical table or a view. The data_table_name
parameter can also accept an inline query.

Note:

Because an inline query can be used to specify the data for transformation, Oracle
strongly recommends that the calling routine perform any necessary SQL injection
checks on the input string.
44.4 DBMS_DATA_MINING_TRANSFORM Datatypes

DBMS_DATA_MINING_TRANSFORM defines the datatypes described in the following table.

Table 44-1 Datatypes in DBMS_DATA_MINING_TRANSFORM

<table>
<thead>
<tr>
<th>List Type</th>
<th>List Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_LIST</td>
<td>VARRAY(1000) OF varchar2(32)</td>
<td>COLUMN_LIST stores quoted and non-quoted identifiers for column names. COLUMN_LIST is the datatype of the exclude_list parameter in the INSERT procedures. See &quot;INSERT_AUTOBIN_NUM_EQWIDTH Procedure&quot; for an example. See Oracle Database PL/SQL Language Reference for information about populating VARRAY structures.</td>
</tr>
<tr>
<td>DESCRIBE_LIST</td>
<td>DBMS_SQL.DESC_TAB2</td>
<td>DESCRIBE_LIST describes the columns of the data table after the transformation list has been applied. A DESCRIBE_LIST is returned by the DESCRIBE_STACK Procedure. The DESC_TAB2 and DESC_REC2 types are defined in the DBMS_SQL package. See &quot;DESC_REC2 Record Type&quot;. The col_type field of DESC_REC2 identifies the datatype of the column. The datatype is expressed as a numeric constant that represents a built-in datatype. For example, a 1 indicates a variable length character string. The codes for Oracle built-in datatypes are listed in Oracle Database SQL Language Reference. The codes for the Oracle Data Mining nested types are described in &quot;Constants&quot;. The col_name field of DESC_REC2 identifies the column name. It may be populated with a column name, an alias, or an expression. If the column name is a SELECT expression, it may be very long. If the expression is longer than 30 bytes, it cannot be used in a view unless it is given an alias.</td>
</tr>
</tbody>
</table>
Table 44-1 (Cont.) Datatypes in DBMS_DATA_MINING_TRANSFORM

<table>
<thead>
<tr>
<th>List Type</th>
<th>List Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSFORM</td>
<td>TABLE OF transform_rec</td>
<td>TRANSFORM_LIST is a list of transformations that can be embedded in a model. A TRANSFORM_LIST is accepted as an argument by the CREATE_MODEL Procedure.</td>
</tr>
<tr>
<td>LIST</td>
<td>TYPE transform_rec IS RECORD (attribute_name VARCHAR2(30), attribute_subname VARCHAR2(4000), expression EXPRESSION_REC, reverse_expression EXPRESSION_REC, attribute_spec VARCHAR2(4000));</td>
<td>Each element in a TRANSFORM_LIST is a TRANSFORM_REC that specifies how to transform a single attribute. The attribute_name is a column name. The attribute_subname is the nested attribute name if the column is nested, otherwise attribute_subname is null. The expression field holds a SQL expression for transforming the attribute. See &quot;About Transformation Lists&quot; for an explanation of reverse expressions. The attribute_spec field can be used to cause the attribute to be handled in a specific way during the model build. See Table 44-33 for details.</td>
</tr>
<tr>
<td></td>
<td>TYPE expression_rec IS RECORD (lstmt DBMS_SQL.VARCHAR2A, lb BINARY_INTEGER DEFAULT 1, ub BINARY_INTEGER DEFAULT 0);</td>
<td>The expressions in a TRANSFORM_REC have type EXPRESSION_REC. The lstmt field stores a VARCHAR2A, which is a table of VARCHAR2(32767). The VARCHAR2A datatype allows transformation expressions to be very long, as they can be broken up across multiple rows of VARCHAR2. The VARCHAR2A type is defined in the DBMS_SQL package. See &quot;VARCHAR2A Table Type&quot;. The ub (upper bound) and lb (lower bound) fields indicate how many rows there are in the VARCHAR2A table. If ub &lt; lb (default) the EXPRESSION_REC is empty; if lb=ub=1 there is one row; if lb=1 and ub=2 there are 2 rows, and so on.</td>
</tr>
<tr>
<td></td>
<td>TYPE varchar2a IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;</td>
<td></td>
</tr>
</tbody>
</table>

44.5 DBMS_DATA_MINING_TRANSFORM Constants

DBMS_DATA_MINING_TRANSFORM defines the constants described in the following table.

Table 44-2 Constants in DBMS_DATA_MINING_TRANSFORM

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEST_NUM_COL_TYPE</td>
<td>100001</td>
<td>Indicates that an attribute in the transformation list comes from a row in a column of DM_NESTED_NUMERICALS. Nested numerical attributes are defined as follows: attribute_name VARCHAR2(4000) value NUMBER</td>
</tr>
</tbody>
</table>
### Table 44-2  (Cont.) Constants in DBMS_DATA_MINING_TRANSFORM

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEST_CAT_COL_TYPE</td>
<td>100002</td>
<td>Indicates that an attribute in the transformation list comes from a row in a column of <code>DM_NESTED_CATEGORICALS</code>. Nested categorical attributes are defined as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attribute_name VARCHAR2(4000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value VARCHAR2(4000)</td>
</tr>
<tr>
<td>NEST_BD_COL_TYPE</td>
<td>100003</td>
<td>Indicates that an attribute in the transformation list comes from a row in a column of <code>DM_NESTED_BINARY_DOUBLES</code>. Nested binary double attributes are defined as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attribute_name VARCHAR2(4000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value BINARY_DOUBLE</td>
</tr>
<tr>
<td>NEST_BF_COL_TYPE</td>
<td>100004</td>
<td>Indicates that an attribute in the transformation list comes from a row in a column of <code>DM_NESTED_BINARY_FLOATS</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attribute_name VARCHAR2(4000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value BINARY_FLOAT</td>
</tr>
</tbody>
</table>

**See Also:**

*Oracle Data Mining User's Guide* for information about nested data in Oracle Data Mining

### 44.6 Summary of DBMS_DATA_MINING_TRANSFORM Subprograms

This table lists the DBMS_DATA_MINING_TRANSFORM subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_BIN_CAT Procedure</td>
<td>Creates a transformation definition table for categorical binning</td>
</tr>
<tr>
<td>CREATE_BIN_NUM Procedure</td>
<td>Creates a transformation definition table for numerical binning</td>
</tr>
<tr>
<td>CREATE_CLIP Procedure</td>
<td>Creates a transformation definition table for clipping</td>
</tr>
<tr>
<td>CREATE_COL.Rem Procedure</td>
<td>Creates a transformation definition table for column removal</td>
</tr>
<tr>
<td>CREATE_MISS_CAT Procedure</td>
<td>Creates a transformation definition table for categorical missing value treatment</td>
</tr>
<tr>
<td>CREATE_MISS_NUM Procedure</td>
<td>Creates a transformation definition table for numerical missing values treatment</td>
</tr>
</tbody>
</table>
### Table 44-3  (Cont.) DBMS_DATA_MINING_TRANSFORM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_NORM_LIN Procedure</td>
<td>Creates a transformation definition table for linear normalization</td>
</tr>
<tr>
<td>DESCRIBE_STACK Procedure</td>
<td>Describes the transformation list</td>
</tr>
<tr>
<td>GET_EXPRESSION Function</td>
<td>Returns a VARCHAR2 chunk from a transformation expression</td>
</tr>
<tr>
<td>INSERT_AUTOBIN_NUM_EQWIDTH Procedure</td>
<td>Inserts numeric automatic equi-width binning definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_BIN_CAT_FREQ Procedure</td>
<td>Inserts categorical frequency-based binning definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_BIN_NUM_EQWIDTH Procedure</td>
<td>Inserts numeric equi-width binning definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_BIN_NUM_QTILE Procedure</td>
<td>Inserts numeric quantile binning expressions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_BIN_SUPER Procedure</td>
<td>Inserts supervised binning definitions in numerical and categorical transformation definition tables</td>
</tr>
<tr>
<td>INSERT_CLIP_TRIM_TAIL Procedure</td>
<td>Inserts numerical trimming definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_CLIP_WINSOR_TAIL Procedure</td>
<td>Inserts numerical winsorizing definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_MISS_CAT_MODE Procedure</td>
<td>Inserts categorical missing value treatment definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_MISS_NUM_MEAN Procedure</td>
<td>Inserts numerical missing value treatment definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_NORM_LIN_MINMAX Procedure</td>
<td>Inserts linear min-max normalization definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_NORM_LIN_SCALE Procedure</td>
<td>Inserts linear scale normalization definitions in a transformation definition table</td>
</tr>
<tr>
<td>INSERT_NORM_LIN_ZSCORE Procedure</td>
<td>Inserts linear zscore normalization definitions in a transformation definition table</td>
</tr>
<tr>
<td>SET_EXPRESSION Procedure</td>
<td>Adds a VARCHAR2 chunk to an expression</td>
</tr>
<tr>
<td>SET_TRANSFORM Procedure</td>
<td>Adds a transformation record to a transformation list</td>
</tr>
<tr>
<td>STACK_BIN_CAT Procedure</td>
<td>Adds a categorical binning expression to a transformation list</td>
</tr>
<tr>
<td>STACK_BIN_NUM Procedure</td>
<td>Adds a numerical binning expression to a transformation list</td>
</tr>
<tr>
<td>STACK_CLIP Procedure</td>
<td>Adds a clipping expression to a transformation list</td>
</tr>
<tr>
<td>STACK_COL_REM Procedure</td>
<td>Adds a column removal expression to a transformation list</td>
</tr>
<tr>
<td>STACK_MISS_CAT Procedure</td>
<td>Adds a categorical missing value treatment expression to a transformation list</td>
</tr>
<tr>
<td>STACK_MISS_NUM Procedure</td>
<td>Adds a numerical missing value treatment expression to a transformation list</td>
</tr>
<tr>
<td>STACK_NORM_LIN Procedure</td>
<td>Adds a linear normalization expression to a transformation list</td>
</tr>
<tr>
<td>XFORM_BIN_CAT Procedure</td>
<td>Creates a view of the data table with categorical binning transformations</td>
</tr>
</tbody>
</table>
Table 44-3  (Cont.) DBMS_DATA_MINING_TRANSFORM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFORM_BIN_NUM Procedure</td>
<td>Creates a view of the data table with numerical binning transformations</td>
</tr>
<tr>
<td>XFORM_CLIP Procedure</td>
<td>Creates a view of the data table with clipping transformations</td>
</tr>
<tr>
<td>XFORM_COL_REM Procedure</td>
<td>Creates a view of the data table with column removal transformations</td>
</tr>
<tr>
<td>XFORM_EXPR_NUM Procedure</td>
<td>Creates a view of the data table with the specified numeric transformations</td>
</tr>
<tr>
<td>XFORM_EXPR_STR Procedure</td>
<td>Creates a view of the data table with the specified categorical transformations</td>
</tr>
<tr>
<td>XFORM_MISS_CAT Procedure</td>
<td>Creates a view of the data table with categorical missing value treatment</td>
</tr>
<tr>
<td>XFORM_MISS_NUM Procedure</td>
<td>Creates a view of the data table with numerical missing value treatment</td>
</tr>
<tr>
<td>XFORM_NORM_LIN Procedure</td>
<td>Creates a view of the data table with linear normalization transformations</td>
</tr>
<tr>
<td>XFORM_STACK Procedure</td>
<td>Creates a view of the transformation list</td>
</tr>
</tbody>
</table>

44.6.1 CREATE_BIN_CAT Procedure

This procedure creates a transformation definition table for categorical binning.

The columns are described in the following table.

Table 44-4  Columns in a Transformation Definition Table for Categorical Binning

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR2(30)</td>
<td>Name of a column of categorical data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the column is not nested, the column name is also the attribute name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For information about attribute names, see Oracle Data Mining User's Guide.</td>
</tr>
<tr>
<td>att</td>
<td>VARCHAR2(4000)</td>
<td>The attribute subname if col is a nested column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If col is nested, the attribute name is col.att. If col is not nested, att is null.</td>
</tr>
<tr>
<td>val</td>
<td>VARCHAR2(4000)</td>
<td>Values of the attribute</td>
</tr>
<tr>
<td>bin</td>
<td>VARCHAR2(4000)</td>
<td>Bin assignments for the values</td>
</tr>
</tbody>
</table>

Syntax

DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT (  
  bin_table_name  IN VARCHAR2,  
  bin_schema_name IN VARCHAR2 DEFAULT NULL );
Parameters

Table 44-5  CREATE_BIN_CAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table to be created</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See *Oracle Data Mining User’s Guide* for details about categorical data.
2. See “Nested Data Transformations” for information about transformation definition tables and nested data.
3. You can use the following procedures to populate the transformation definition table:
   - INSERT_BIN_CAT_FREQ Procedure — frequency-based binning
   - INSERT_BIN_SUPER Procedure — supervised binning

See Also:

“Binning” in DBMS_DATA_MINING_TRANSFORM Overview

“Operational Notes”

Examples

The following statement creates a table called `bin_cat_xtbl` in the current schema. The table has columns that can be populated with bin assignments for categorical attributes.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT('bin_cat_xtbl');
END;
/
```

```
DESCRIBE bin_cat_xtbl
Name                                      Null?    Type
----------------------------------------- -------- -----------------------------
COL                                                VARCHAR2(30)
ATT                                                VARCHAR2(4000)
VAL                                                VARCHAR2(4000)
BIN                                                VARCHAR2(4000)
```

44.6.2 CREATE_BIN_NUM Procedure

This procedure creates a transformation definition table for numerical binning. The columns are described in the following table.
Table 44-6  Columns in a Transformation Definition Table for Numerical Binning

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR2(30)</td>
<td>Name of a column of numerical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see Oracle Data Mining User's Guide.</td>
</tr>
<tr>
<td>att</td>
<td>VARCHAR2(4000)</td>
<td>The attribute subname if col is a nested column. If col is nested, the attribute name is col.att. If col is not nested, att is null.</td>
</tr>
<tr>
<td>val</td>
<td>NUMBER</td>
<td>Values of the attribute</td>
</tr>
<tr>
<td>bin</td>
<td>VARCHAR2(4000)</td>
<td>Bin assignments for the values</td>
</tr>
</tbody>
</table>

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM (  
    bin_table_name    IN VARCHAR2,  
    bin_schema_name   IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 44-7  CREATE_BIN_NUM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table to be created</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User’s Guide for details about numerical data.

2. See "Nested Data Transformations" for information about transformation definition tables and nested data.

3. You can use the following procedures to populate the transformation definition table:
   - INSERT_AUTOBIN_NUM_EQWIDTH Procedure — automatic equi-width binning
   - INSERT_BIN_NUM_EQWIDTH Procedure — user-specified equi-width binning
   - INSERT_BIN_NUM_QTILE Procedure — quantile binning
   - INSERT_BIN_SUPER Procedure — supervised binning
Examples

The following statement creates a table called `bin_num_xtbl` in the current schema. The table has columns that can be populated with bin assignments for numerical attributes.

```sql
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM('bin_num_xtbl');
END;
/
```

```sql
DESCRIBE bin_num_xtbl
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
COL                                                VARCHAR2(30)
ATT                                                VARCHAR2(4000)
VAL                                                NUMBER
BIN                                                VARCHAR2(4000)
```

44.6.3 CREATE_CLIP Procedure

This procedure creates a transformation definition table for clipping or winsorizing to minimize the effect of outliers.

The columns are described in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR2(30)</td>
<td>Name of a column of numerical data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the column is not nested, the column name is also</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the attribute name. For information about attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>names, see Oracle Data Mining User's Guide.</td>
</tr>
<tr>
<td>att</td>
<td>VARCHAR2(4000)</td>
<td>The attribute subname if <code>col</code> is a nested column of</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DM_NESTED_NUMERICALS</code>. If <code>col</code> is nested, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attribute name is <code>col.att</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If <code>col</code> is not nested, <code>att</code> is null.</td>
</tr>
<tr>
<td>lcut</td>
<td>NUMBER</td>
<td>The lowest typical value for the attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the attribute values were plotted on an <code>xy</code> axis,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>lcut</code> would be the left-most boundary of the range of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values considered typical for this attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any values to the left of <code>lcut</code> are outliers.</td>
</tr>
<tr>
<td>lval</td>
<td>NUMBER</td>
<td>Value assigned to an outlier to the left of <code>lcut</code>.</td>
</tr>
</tbody>
</table>
Table 44-8  (Cont.) Columns in a Transformation Definition Table for Clipping or Winsorizing

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcut</td>
<td>NUMBER</td>
<td>The highest typical value for the attribute. If the attribute values were plotted on an (xy) axis, (rcut) would be the right-most boundary of the range of values considered typical for this attribute. Any values to the right of (rcut) are outliers.</td>
</tr>
<tr>
<td>rval</td>
<td>NUMBER</td>
<td>Value assigned to an outlier to the right of (rcut)</td>
</tr>
</tbody>
</table>

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP (
    clip_table_name    IN VARCHAR2,
    clip_schema_name   IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 44-9  CREATE_CLIP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip_table_name</td>
<td>Name of the transformation definition table to be created</td>
</tr>
<tr>
<td>clip_schema_name</td>
<td>Schema of (clip_table_name). If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User’s Guide for details about numerical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the following procedures to populate the transformation definition table:
   - INSERT_CLIP_TRIM_TAIL Procedure — replaces outliers with nulls
   - INSERT_CLIP_WINSOR_TAIL Procedure — replaces outliers with an average value

See Also:

"Outlier Treatment" in DBMS_DATA_MINING_TRANSFORM Overview
"Operational Notes"
Examples

The following statement creates a table called `clip_xtbl` in the current schema. The table has columns that can be populated with clipping instructions for numerical attributes.

```sql
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP('clip_xtbl');
END;
/
```

```
DESCRIBE clip_xtbl
Name                                      Null?  Type
----------------------------------------- ------- ----------------------------
COL                                                VARCHAR2(30)
ATT                                                VARCHAR2(4000)
LCUT                                               NUMBER
LVAL                                               NUMBER
RCUT                                               NUMBER
RVAL                                               NUMBER
```

44.6.4 CREATE_COL_REM Procedure

This procedure creates a transformation definition table for removing columns from the data table.

The columns are described in the following table.

Table 44-10  Columns in a Transformation Definition Table for Column Removal

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR2(30)</td>
<td>Name of a column of data. If the column is not nested, the column name is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>also the attribute name. For information about attribute names, see Oracle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Mining User's Guide.</td>
</tr>
<tr>
<td>att</td>
<td>VARCHAR2(4000)</td>
<td>The attribute subname if <code>col</code> is nested (<code>DM_NESTED_NUMERIC</code> or <code>DM_NESTED_CATEGORICAL</code>). If <code>col</code> is nested, the attribute name is <code>col.att</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If <code>col</code> is not nested, <code>att</code> is null.</td>
</tr>
</tbody>
</table>

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM(
    rem_table_name VARCHAR2,
    rem_schema_name VARCHAR2 DEFAULT NULL);
```

Parameters

Table 44-11  CREATE_COL_REM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rem_table_name</td>
<td>Name of the transformation definition table to be created</td>
</tr>
<tr>
<td>rem_schema_name</td>
<td>Schema of <code>rem_table_name</code>. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>
Usage Notes

1. See "Nested Data Transformations" for information about transformation definition tables and nested data.

2. See "Operational Notes".

Examples

The following statement creates a table called `rem_att_xtbl` in the current schema. The table has columns that can be populated with the names of attributes to exclude from the data to be mined.

```sql
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM ('rem_att_xtbl');
END;
/
DESCRIBE rem_att_xtbl
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL</td>
<td>VARCHAR2(30)</td>
<td></td>
</tr>
<tr>
<td>ATT</td>
<td>VARCHAR2(4000)</td>
<td></td>
</tr>
</tbody>
</table>

44.6.5 CREATE_MISS_CAT Procedure

This procedure creates a transformation definition table for replacing categorical missing values.

The columns are described in the following table.

**Table 44-12  Columns in a Transformation Definition Table for Categorical Missing Value Treatment**

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR2(30)</td>
<td>Name of a column of categorical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see Oracle Data Mining User's Guide.</td>
</tr>
<tr>
<td>att</td>
<td>VARCHAR2(4000)</td>
<td>The attribute subname if <code>col</code> is a nested column of <code>DM_NESTED_CATEGORICALS</code>. If <code>col</code> is nested, the attribute name is <code>col.att</code>. If <code>col</code> is not nested, <code>att</code> is null.</td>
</tr>
<tr>
<td>val</td>
<td>VARCHAR2(4000)</td>
<td>Replacement for missing values in the attribute</td>
</tr>
</tbody>
</table>

**Syntax**

```sql
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT (
  miss_table_name  IN VARCHAR2,
  miss_schema_name IN VARCHAR2 DEFAULT NULL );
```
Parameters

Table 44-13  CREATE_MISS_CAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table to be created</td>
</tr>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User's Guide for details about categorical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the INSERT_MISS_CAT_MODE Procedure to populate the transformation definition table.

See Also:

"Missing Value Treatment" in DBMS_DATA_MINING_TRANSFORM Overview
"Operational Notes"

Examples

The following statement creates a table called miss_cat_xtbl in the current schema. The table has columns that can be populated with values for missing data in categorical attributes.

BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT('miss_cat_xtbl');
END;
/

DESCRIBE miss_cat_xtbl

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>ATT</td>
<td></td>
<td>VARCHAR2(4000)</td>
</tr>
<tr>
<td>VAL</td>
<td></td>
<td>VARCHAR2(4000)</td>
</tr>
</tbody>
</table>

44.6.6 CREATE_MISS_NUM Procedure

This procedure creates a transformation definition table for replacing numerical missing values.

The columns are described in Table 44-14.
Table 44-14  Columns in a Transformation Definition Table for Numerical Missing Value Treatment

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR2(30)</td>
<td>Name of a column of numerical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see Oracle Data Mining User's Guide.</td>
</tr>
<tr>
<td>att</td>
<td>VARCHAR2(4000)</td>
<td>The attribute subname if col is a nested column of DM_NESTED_NUMERICALS. If col is nested, the attribute name is col.att. If col is not nested, att is null.</td>
</tr>
<tr>
<td>val</td>
<td>NUMBER</td>
<td>Replacement for missing values in the attribute</td>
</tr>
</tbody>
</table>

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM (
    miss_table_name       IN VARCHAR2,
    miss_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 44-15  CREATE_MISS_NUM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table to be created</td>
</tr>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User's Guide for details about numerical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the INSERT_MISS_NUM_MEAN Procedure to populate the transformation definition table.

See Also:

* "Missing Value Treatment" in DBMS_DATA_MINING_TRANSFORM Overview
* "Operational Notes"
Example

The following statement creates a table called `miss_num_xtbl` in the current schema. The table has columns that can be populated with values for missing data in numerical attributes.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM('miss_num_xtbl');
END;
/
```

```
DESCRIBE miss_num_xtbl
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
COL                                                VARCHAR2(30)
ATT                                                VARCHAR2(4000)
VAL                                                NUMBER
```

44.6.7 CREATE_NORM_LIN Procedure

This procedure creates a transformation definition table for linear normalization. The columns are described in Table 44-16.

Table 44-16  Columns in a Transformation Definition Table for Linear Normalization

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR2(30)</td>
<td>Name of a column of numerical data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the column is not nested, the column name is also the attribute name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For information about attribute names, see Oracle Data Mining User's Guide.</td>
</tr>
<tr>
<td>att</td>
<td>VARCHAR2(4000)</td>
<td>The attribute subname if <code>col</code> is a nested column of DM_NESTED_NUMERICALS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If <code>col</code> is nested, the attribute name is <code>col.att</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If <code>col</code> is not nested, <code>att</code> is null.</td>
</tr>
<tr>
<td>shift</td>
<td>NUMBER</td>
<td>A constant to subtract from the attribute values</td>
</tr>
<tr>
<td>scale</td>
<td>NUMBER</td>
<td>A constant by which to divide the shifted values</td>
</tr>
</tbody>
</table>

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN (  
    norm_table_name IN VARCHAR2,  
    norm_schema_name IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 44-17  CREATE_NORM_LIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm_table_name</td>
<td>Name of the transformation definition table to be created</td>
</tr>
<tr>
<td>norm_schema_name</td>
<td>Schema of <code>norm_table_name</code>. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>
Usage Notes

1. See Oracle Data Mining User's Guide for details about numerical data.

2. See "Nested Data Transformations" for information about transformation definition tables and nested data.

3. You can use the following procedures to populate the transformation definition table:
   - INSERT_NORM_LIN_MINMAX Procedure — Uses linear min-max normalization
   - INSERT_NORM_LIN_SCALE Procedure — Uses linear scale normalization
   - INSERT_NORM_LIN_ZSCORE Procedure — Uses linear zscore normalization

See Also:

"Linear Normalization" in DBMS_DATA_MINING_TRANSFORM Overview
"Operational Notes"

Examples

The following statement creates a table called norm_xtbl in the current schema. The table has columns that can be populated with shift and scale values for normalizing numerical attributes.

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN('norm_xtbl');
END;
```

```
DESCRIBE norm_xtbl
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
COL                                                VARCHAR2(30)
ATT                                                VARCHAR2(4000)
SHIFT                                              NUMBER
SCALE                                              NUMBER
```

44.6.8 DESCRIBE_STACK Procedure

This procedure describes the columns of the data table after a list of transformations has been applied.

Only the columns that are specified in the transformation list are transformed. The remaining columns in the data table are included in the output without changes.

To create a view of the data table after the transformations have been applied, use the XFORM_STACK Procedure.
Syntax

```
DBMS_DATA_MINING_TRANSFORM.DESCRIBE_STACK (  
    xform_list           IN  TRANSFORM_LIST,  
    data_table_name      IN  VARCHAR2,  
    describe_list        OUT DESCRIBE_LIST,  
    data_schema_name     IN  VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 44-18  DESCRIBE_STACK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xform_list</td>
<td>A list of transformations. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>describe_list</td>
<td>Descriptions of the columns in the data table after the transformations specified in xform_list have been applied. See Table 44-1 for a description of the DESCRIBE_LIST object type.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes" for information about transformation lists and embedded transformations.

Examples

This example shows the column name and datatype, the column name length, and the column maximum length for the view dmuser.cust_info after the transformation list has been applied. All the transformations are user-specified. The results of DESCRIBE_STACK do not include one of the columns in the original table, because the SET_TRANSFORM procedure sets that column to NULL.

```
CREATE OR REPLACE VIEW cust_info AS
    SELECT a.cust_id, c.country_id, c.cust_year_of_birth,
    CAST(COLLECT(DM_Nested_Numerical(
        b.prod_name, 1) )
      AS DM_Nested_Numericals) custprods
    FROM sh.sales a, sh.products b, sh.customers c
    WHERE a.prod_id = b.prod_id AND
    a.cust_id=c.cust_id and
    a.cust_id between 100001 AND 105000
    GROUP BY a.cust_id, country_id, cust_year_of_birth;
```

describe cust_info

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>COUNTRY ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUST YEAR_OF_BIRTH</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUSTPRODS</td>
<td></td>
<td>SYS.DM_NESTED_NUMERICALS</td>
</tr>
</tbody>
</table>

DECLARE
    cust_stack   dbms_data_mining_transform.TRANSFORM_LIST;
```
```sql
cust_cols dbms_data_mining_transform.DESCRIBE_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'country_id', NULL, 'country_id/10', 'country_id*10');
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'cust_year_of_birth', NULL, NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'custprods', 'Mouse Pad', 'value*100', 'value/100');
  dbms_data_mining_transform.DESCRIBE_STACK(
    xform_list => cust_stack,
    data_table_name => 'cust_info',
    describe_list => cust_cols);
  dbms_output.put_line('====');
  for i in 1..cust_cols.COUNT loop
    dbms_output.put_line('COLUMN_NAME:     '||cust_cols(i).col_name);
    dbms_output.put_line('COLUMN_TYPE:     '||cust_cols(i).col_type);
    dbms_output.put_line('COLUMN_NAME_LEN: '||cust_cols(i).col_name_len);
    dbms_output.put_line('COLUMN_MAX_LEN:  '||cust_cols(i).col_max_len);
    dbms_output.put_line('====');
  END loop;
END;
/
====
COLUMN_NAME:     CUST_ID
COLUMN_TYPE:     2
COLUMN_NAME_LEN: 7
COLUMN_MAX_LEN: 22
====
COLUMN_NAME:     COUNTRY_ID
COLUMN_TYPE:     2
COLUMN_NAME_LEN: 10
COLUMN_MAX_LEN: 22
====
COLUMN_NAME:     CUSTPRODS
COLUMN_TYPE:     100001
COLUMN_NAME_LEN: 9
COLUMN_MAX_LEN: 40
====

44.6.9 GET_EXPRESSION Function

This function returns a row from a VARCHAR2 array that stores a transformation expression. The array is built by calls to the SET_EXPRESSION Procedure.

The array can be used for specifying SQL expressions that are too long to be used with the SET_TRANSFORM Procedure.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.GET_EXPRESSION ( expression IN EXPRESSION_REC,
                                             chunk_num IN PLS_INTEGER DEFAULT NULL);
RETURN VARCHAR2;
```
```
Parameters

Table 44-19  GET_EXPRESSION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>An expression record (EXPRESSION_REC) that specifies a transformation expression or a reverse transformation expression for an attribute. Each expression record includes a VARCHAR2 array and index fields for specifying upper and lower boundaries within the array. There are two EXPRESSION_REC fields within a transformation record (TRANSFORM_REC): one for the transformation expression; the other for the reverse transformation expression. See Table 44-1 for a description of the EXPRESSION_REC type.</td>
</tr>
<tr>
<td>chunk</td>
<td>A VARCHAR2 chunk (row) to be appended to expression.</td>
</tr>
</tbody>
</table>

Usage Notes

1. Chunk numbering starts with one. For chunks outside of the range, the return value is null. When a chunk number is null the whole expression is returned as a string. If the expression is too big, a VALUE_ERROR is raised.

2. See "About Transformation Lists".

3. See "Operational Notes".

Examples

See the example for the SET_EXPRESSION Procedure.

Related Topics

• SET_EXPRESSION Procedure
  This procedure appends a row to a VARCHAR2 array that stores a SQL expression.

• SET_TRANSFORM Procedure
  This procedure appends the transformation instructions for an attribute to a transformation list.

44.6.10 INSERT_AUTOBIN_NUM_EQWIDTH Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure identifies the minimum and maximum values and computes the bin boundaries at equal intervals.

INSERT_AUTOBIN_NUM_EQWIDTH computes the number of bins separately for each column. If you want to use equi-width binning with the same number of bins for each column, use the INSERT_BIN_NUM_EQWIDTH Procedure.

INSERT_AUTOBIN_NUM_EQWIDTH bins all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

DBMS_DATA_MINING_TRANSFORM.INSERT_AUTOBIN_NUM_EQWIDTH (bin_table_name IN VARCHAR2, data_table_name IN VARCHAR2,
Parameters

Table 44-20  INSERT_AUTOBIN_NUM_EQWIDTH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The following columns are required:</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>bin_num</td>
<td>Minimum number of bins. If bin_num is 0 or NULL, it is ignored. The default value of bin_num is 3.</td>
</tr>
<tr>
<td>max_bin_num</td>
<td>Maximum number of bins. If max_bin_num is 0 or NULL, it is ignored. The default value of max_bin_num is 100.</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from the binning process. If you do not specify exclude_list, all numerical columns in the data source are binned. The format of exclude_list is:</td>
</tr>
<tr>
<td>round_num</td>
<td>Specifies how to round the number in the VAL column of the transformation definition table. When round_num is positive, it specifies the most significant digits to retain. When round_num is negative, it specifies the least significant digits to remove. In both cases, the result is rounded to the specified number of digits. See the Usage Notes for an example. The default value of round_num is 6.</td>
</tr>
<tr>
<td>sample_size</td>
<td>Size of the data sample. If sample_size is less than the total number of non-NULL values in the column, then sample_size is used instead of the SQL COUNT function in computing the number of bins. If sample_size is 0 or NULL, it is ignored. See the Usage Notes. The default value of sample_size is 50,000.</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>
Table 44-20  (Cont.) INSERT_AUTOBIN_NUM_EQWIDTH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>rem_table_name</td>
<td>Name of a transformation definition table for column removal. The table must have the columns described in &quot;CREATE_COL_REM Procedure&quot;. INSERT_AUTOBIN_NUM_EQWIDTH ignores columns with all nulls or only one unique value. If you specify a value for rem_table_name, these columns are removed from the mining data. If you do not specify a value for rem_table_name, these unbinned columns remain in the data.</td>
</tr>
<tr>
<td>rem_schema_name</td>
<td>Schema of rem_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User's Guide for details about numerical data.
2. INSERT_AUTOBIN_NUM_EQWIDTH computes the number of bins for a column based on the number of non-null values \( \text{COUNT} \), the maximum \( \text{MAX} \), the minimum \( \text{MIN} \), the standard deviation \( \text{STDDEV} \), and the constant \( C=3.49/0.9 \):

\[
N = \text{floor}\left(\left(\frac{1}{3}\right)\left(\text{MAX}-\text{MIN}\right)\left(\text{C}\times\text{std}\text{dev}\right)\right)
\]

If the sample_size parameter is specified, it is used instead of \( \text{COUNT} \).

See Oracle Database SQL Language Reference for information about the COUNT, MAX, MIN, STDDEV, FLOOR, and POWER functions.

3. INSERT_AUTOBIN_NUM_EQWIDTH uses absolute values to compute the number of bins. The sign of the parameters bin_num, max_bin_num, and sample_size has no effect on the result.

4. In computing the number of bins, INSERT_AUTOBIN_NUM_EQWIDTH evaluates the following criteria in the following order:
   a. The minimum number of bins \( \text{bin num} \)
   b. The maximum number of bins \( \text{max bin num} \)
   c. The maximum number of bins for integer columns, calculated as the number of distinct values in the range \( \text{max-min}+1 \).

5. The round_num parameter controls the rounding of column values in the transformation definition table, as follows:

   For a value of 308.162:

   - when round_num = 1       result is 300
   - when round_num = 2       result is 310
   - when round_num = 3       result is 308
   - when round_num = 0       result is 308.162
   - when round_num = -1      result is 308.16
   - when round_num = -2      result is 308.2
Examples

In this example, \texttt{INSERT\_AUTOBIN\_NUM\_EQWIDTH} computes the bin boundaries for the \texttt{cust\_year\_of\_birth} column in \texttt{sh.customers} and inserts the transformations in a transformation definition table. The \texttt{STACK\_BIN\_NUM} Procedure creates a transformation list from the contents of the definition table. The \texttt{CREATE\_MODEL} Procedure embeds the transformation list in a new model called \texttt{nb\_model}.

The transformation and reverse transformation expressions embedded in \texttt{nb\_model} are returned by the \texttt{GET\_MODEL\_TRANSFORMATIONS} Function.

\begin{verbatim}
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_postal_code
    FROM sh.customers;

DESCRIBE mining_data
<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>NOT NULL</td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
</tbody>
</table>

BEGIN
    dbms_data_mining_transform.CREATE\_BIN\_NUM(
        bin_table_name   => 'bin\_tbl');
    dbms_data_mining_transform.INSERT\_AUTOBIN\_NUM\_EQWIDTH (
        bin_table_name   => 'bin\_tbl',
        data_table_name  => 'mining\_data',
        bin_num          => 3,
        max_bin_num      => 5,
        exclude_list     => dbms_data_mining_transform.COLUMN\_LIST('cust\_id'));
END;

set numwidth 4
column val off
SELECT col, val, bin FROM bin\_tbl
    ORDER BY val ASC;

<table>
<thead>
<tr>
<th>COL</th>
<th>VAL</th>
<th>BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1913</td>
<td></td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1928</td>
<td>1</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1944</td>
<td>2</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1959</td>
<td>3</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1975</td>
<td>4</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1990</td>
<td>5</td>
</tr>
</tbody>
</table>

DECLARE
    year\_birth\_xform   dbms_data_mining_transform.TRANSFORM\_LIST;
BEGIN
    dbms_data_mining_transform.STACK\_BIN\_NUM (  
        bin_table_name   => 'bin\_tbl',
        xform_list       => year\_birth\_xform);
    dbms_data_mining.CREAT\_MODEL(
        model_name   => 'nb\_model',
        mining_function => dbms_data_mining.classification,
        data_table_name => 'mining\_data',
        case_id_column_name => 'cust\_id',
        target_column_name => 'cust\_postal\_code',
    );
END;
\end{verbatim}
44.6.11 INSERT_BIN_CAT_FREQ Procedure

This procedure performs categorical binning and inserts the transformation definitions in a transformation definition table. The procedure computes the bin boundaries based on frequency.

**INSERT_BIN_CAT_FREQ** bins all the **CHAR** and **VARCHAR2** columns in the data source unless you specify a list of columns to ignore.

**Syntax**

```sql
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_CAT_FREQ (  
    bin_table_name IN VARCHAR2,  
    data_table_name IN VARCHAR2,  
    bin_num IN PLS_INTEGER DEFAULT 9,  
    exclude_list IN COLUMN_LIST DEFAULT NULL,  
    default_num IN PLS_INTEGER DEFAULT 2,  
    bin_support IN NUMBER DEFAULT NULL,  
    bin_schema_name IN VARCHAR2 DEFAULT NULL,  
    data_schema_name IN VARCHAR2 DEFAULT NULL);
```
### Parameters

**Table 44-21  INSERT_BIN_CAT_FREQ Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| bin_table_name| Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table. The following columns are required:  
  COL VARCHAR2(30)  
  VAL VARCHAR2(4000)  
  BIN VARCHAR2(4000)  
  CREATE_BIN_CAT creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_CAT_FREQ. |
| data_table_name| Name of the table containing the data to be transformed                                                                                                    |
| bin_num       | The number of bins to fill using frequency-based binning. The total number of bins will be \( \text{bin\_num}+1 \). The additional bin is the default bin. Classes that are not assigned to a frequency-based bin will be assigned to the default bin.  
  The default binning order is from highest to lowest: the most frequently occurring class is assigned to the first bin, the second most frequently occurring class is assigned to the second bin, and so on. You can reverse the binning order by specifying a negative number for \( \text{bin\_num} \). The negative sign causes the binning order to be from lowest to highest.  
  If the total number of distinct values (classes) in the column is less than \( \text{bin\_num} \), then a separate bin will be created for each value and the default bin will be empty.  
  If you specify NULL or 0 for \( \text{bin\_num} \), no binning is performed.  
  The default value of \( \text{bin\_num} \) is 9. |
| exclude_list  | List of categorical columns to be excluded from the binning process. If you do not specify exclude_list, all categorical columns in the data source are binned.  
  The format of exclude_list is:  
  `dbms_data_mining_transform.COLUMN_LIST('col1','col2',...,'coln')` |
| default_num   | The number of class occurrences (rows of the same class) required for assignment to the default bin.  
  By default, default_num is the minimum number of occurrences required for assignment to the default bin. For example, if default_num is 3 and a given class occurs only once, it will not be assigned to the default bin. You can change the occurrence requirement from minimum to maximum by specifying a negative number for default_num. For example, if default_num is -3 and a given class occurs only once, it will be assigned to the default bin, but a class that occurs four or more times will not be included.  
  If you specify NULL or 0 for default_bin, there are no requirements for assignment to the default bin.  
  The default value of default_num is 2. |
Table 44-21  (Cont.) INSERT_BIN_CAT_FREQ Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_support</td>
<td>The number of class occurrences (rows of the same class) required for assignment to a frequency-based bin. <em>bin_support</em> is expressed as a fraction of the total number of rows. By default, <em>bin_support</em> is the minimum percentage required for assignment to a frequency-based bin. For example, if there are twenty rows of data and you specify .2 for <em>bin_support</em>, then there must be four or more occurrences of a class (.2*20) in order for it to be assigned to a frequency-based bin. You can change <em>bin_support</em> from a minimum percentage to a maximum percentage by specifying a negative number for <em>bin_support</em>. For example, if there are twenty rows of data and you specify -.2 for <em>bin_support</em>, then there must be four or less occurrences of a class in order for it to be assigned to a frequency-based bin. Classes that occur less than a positive <em>bin_support</em> or more than a negative <em>bin_support</em> will be assigned to the default bin. If you specify NULL or 0 for <em>bin_support</em>, then there is no support requirement for frequency-based binning. The default value of <em>bin_support</em> is NULL.</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of <em>bin_table_name</em>. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of <em>data_table_name</em>. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User’s Guide for details about categorical data.

2. If values occur with the same frequency, INSERT_BIN_CAT_FREQ assigns them in descending order when binning is from most to least frequent, or in ascending order when binning is from least to most frequent.

Examples

1. In this example, INSERT_BIN_CAT_FREQ computes the bin boundaries for the *cust_postal_code* and *cust_city* columns in *sh.customers* and inserts the transformations in a transformation definition table. The STACK_BIN_CAT Procedure creates a transformation list from the contents of the definition table, and the CREATE_MODEL Procedure embeds the transformation list in a new model called *nb_model*.

   The transformation and reverse transformation expressions embedded in *nb_model* are returned by the GET_MODEL_TRANSFORMATIONS Function.

   ```sql
   CREATE OR REPLACE VIEW mining_data AS
   SELECT cust_id, cust_year_of_birth, cust_postal_code, cust_city
   FROM sh.customers;
   DESCRIBE mining_data
   Name                                    Null? Type
   ---------------------------------------- -------- -----------------------------
   CUST_ID                                 NOT NULL NUMBER
   CUST_YEAR_OF_BIRTH                      NOT NULL NUMBER(4)
   ```
BEGIN
  dbms_data_mining_transform.CREATE_BIN_CAT(
    bin_table_name => 'bin_tbl_1');
  dbms_data_mining_transform.INSERT_BIN_CAT_FREQ(
    bin_table_name => 'bin_tbl_1',
    data_table_name => 'mining_data',
    bin_num => 4);
END;
/

column col format a18
column val format a15
column bin format a10
SELECT col, val, bin
  FROM bin_tbl_1
  ORDER BY col ASC, bin ASC;

COL                VAL             BIN
------------------ --------------- ----------
CUST_CITY          Los Angeles     1
CUST_CITY          Greenwich       2
CUST_CITY          Killarney       3
CUST_CITY          Montara         4
CUST_CITY                          5
CUST_POSTAL_CODE   38082           1
CUST_POSTAL_CODE   63736           2
CUST_POSTAL_CODE   55787           3
CUST_POSTAL_CODE   78558           4
CUST_POSTAL_CODE                   5

DECLARE
  city_xform dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_CAT(
    bin_table_name => 'bin_tbl_1',
    xform_list => city_xform);
  dbms_data_mining.CREATE_MODEL(
    model_name => 'nb_model',
    mining_function => dbms_data_mining.classification,
    data_table_name => 'mining_data',
    case_id_column_name => 'cust_id',
    target_column_name => 'cust_city',
    settings_table_name => null,
    data_schema_name => null,
    settings_schema_name => null,
    xform_list => city_xform);
END;
/

SELECT attribute_name
  FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

ATTRIBUTE_NAME
-----------------------------------------------------------------------------
CUST_CITY
CUST_POSTAL_CODE

SELECT expression
FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

EXPRESSION
---------------------------------------------------------------
DECODE("CUST_CITY","Greenwich","2","Killarney","3","Los Angeles","1",
'Montara","4",NULL,NULL,"5")
DECODE("CUST_POSTAL_CODE","38082","1","55787","3","63736","2","78558","4",NULL,NULL,"5")

SELECT reverse_expression
FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

REVERSE_EXPRESSION
---------------------------------------------------------------
DECODE("CUST_CITY","2","'Greenwich'","3","'Killarney'","1",
'Los Angeles',"4",NULL,NULL,"5","DEFAULT'")
DECODE("CUST_POSTAL_CODE","1","'38082'","3","'55787'","2","'63736'",'4',''78558'",NULL,NULL,"5","DEFAULT'")

2. The binning order in example 1 is from most frequent to least frequent. The following example shows reverse order binning (least frequent to most frequent). The binning order is reversed by setting bin_num to -4 instead of 4.

BEGIN
    dbms_data_mining_transform.CREATE_BIN_CAT(  
        bin_table_name   => 'bin_tbl_reverse');
    dbms_data_mining_transform.INSERT_BIN_CAT_FREQ (  
        bin_table_name   => 'bin_tbl_reverse',
        data_table_name  => 'mining_data',
        bin_num          => -4);
END;
/

column col format a20
SELECT col, val, bin
FROM bin_tbl_reverse
ORDER BY col ASC, bin ASC;

<table>
<thead>
<tr>
<th>COL</th>
<th>VAL</th>
<th>BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_CITY</td>
<td>Tokyo</td>
<td>1</td>
</tr>
<tr>
<td>CUST_CITY</td>
<td>Sliedrecht</td>
<td>2</td>
</tr>
<tr>
<td>CUST_CITY</td>
<td>Haarlem</td>
<td>3</td>
</tr>
<tr>
<td>CUST_CITY</td>
<td>Diemen</td>
<td>4</td>
</tr>
<tr>
<td>CUST_CITY</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>49358</td>
<td>1</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>80563</td>
<td>2</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>74903</td>
<td>3</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>71349</td>
<td>4</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

44.6.12 INSERT_BIN_NUM_EQWIDTH Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure identifies the minimum and maximum values and computes the bin boundaries at equal intervals.

INSERT_BIN_NUM_EQWIDTH computes a specified number of bins (n) and assigns \((\text{max-min})/n\) values to each bin. The number of bins is the same for each column. If you want to use equi-width binning, but you want the number of bins to be calculated on a per-column basis, use the INSERT_AUTOBIN_NUM_EQWIDTH Procedure.
INSERT_BIN_NUM_EQWIDTH bins all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

**Syntax**

```sql
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_EQWIDTH (
    bin_table_name        IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    bin_num               IN PLS_INTEGER DEFAULT 10,
    exclude_list          IN COLUMN_LIST DEFAULT NULL,
    round_num             IN PLS_INTEGER DEFAULT 6,
    bin_schema_name       IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

**Table 44-22 INSERT_BIN_NUM_EQWIDTH Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `bin_table_name` | Name of the transformation definition table for numerical binning. You can use the `CREATE_BIN_NUM` Procedure to create the definition table. The following columns are required:  

<table>
<thead>
<tr>
<th>COL</th>
<th>VARCHAR2(30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BIN</td>
<td>VARCHAR2(4000)</td>
</tr>
</tbody>
</table>

CREATE_BIN_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_NUM_EQWIDTH. |
| `data_table_name` | Name of the table containing the data to be transformed |
| `bin_num`         | Number of bins. No binning occurs if `bin_num` is 0 or NULL. The default number of bins is 10. |
| `exclude_list`    | List of numerical columns to be excluded from the binning process. If you do not specify `exclude_list`, all numerical columns in the data source are binned.  
The format of `exclude_list` is:  

```sql
dbms_data_mining_transform.COLUMN_LIST('col1','col2',...'coln')
```

| `round_num`       | Specifies how to round the number in the VAL column of the transformation definition table.  

When `round_num` is positive, it specifies the most significant digits to retain. When `round_num` is negative, it specifies the least significant digits to remove. In both cases, the result is rounded to the specified number of digits. See the Usage Notes for an example.  
The default value of `round_num` is 6. |
| `bin_schema_name` | Schema of `bin_table_name`. If no schema is specified, the current schema is used. |
| `data_schema_name` | Schema of `data_table_name`. If no schema is specified, the current schema is used. |
Usage Notes

1. See Oracle Data Mining User's Guide for details about numerical data.

2. The \texttt{round\_num} parameter controls the rounding of column values in the transformation definition table, as follows:

   \textbf{For a value of 308.162:}
   
   - when \texttt{round\_num} = 1 \quad \text{result is 300}
   - when \texttt{round\_num} = 2 \quad \text{result is 310}
   - when \texttt{round\_num} = 3 \quad \text{result is 308}
   - when \texttt{round\_num} = 0 \quad \text{result is 308.162}
   - when \texttt{round\_num} = -1 \quad \text{result is 308.16}
   - when \texttt{round\_num} = -2 \quad \text{result is 308.2}

3. \texttt{INSERT\_BIN\_NUM\_EQWIDTH} ignores columns with all NULL values or only one unique value.

Examples

In this example, \texttt{INSERT\_BIN\_NUM\_EQWIDTH} computes the bin boundaries for the \texttt{affinity\_card} column in \texttt{mining\_data\_build} and inserts the transformations in a transformation definition table. The \texttt{STACK\_BIN\_NUM Procedure} creates a transformation list from the contents of the definition table. The \texttt{CREATE\_MODEL Procedure} embeds the transformation list in a new model called \texttt{glm\_model}.

The transformation and reverse transformation expressions embedded in \texttt{glm\_model} are returned by the \texttt{GET\_MODEL\_TRANSFORMATIONS Function}.

CREATE OR REPLACE VIEW mining\_data AS
SELECT cust\_id, cust\_income\_level, cust\_gender, affinity\_card
FROM mining\_data\_build;

DESCRIBE mining\_data
Name Null? Type
------------------------- -------- -----------------
CUST\_ID NOT NULL NUMBER
CUST\_INCOME\_LEVEL VARCHAR2(30)
CUST\_GENDER VARCHAR2(1)
AFFINITY\_CARD NUMBER(10)

BEGIN
 dbms\_data\_mining\_transform.CREATE\_BIN\_NUM(
   bin\_table\_name  => 'bin\_tbl');
 dbms\_data\_mining\_transform.INSERT\_BIN\_NUM\_EQWIDTH (
   bin\_table\_name  => 'bin\_tbl',
   data\_table\_name => 'mining\_data',
   bin\_num         => 4,
   exclude\_list    => dbms\_data\_mining\_transform.COLUMN\_LIST('cust\_id'));
 END;
/

set numwidth 10
column val off
column col format a20
column bin format a10
SELECT col, val, bin FROM bin\_tbl
   ORDER BY val ASC;

COL VAL BIN
CREATE TABLE glmsettings(
    setting_name  VARCHAR2(30),
    setting_value VARCHAR2(30));

BEGIN
    INSERT INTO glmsettings (setting_name, setting_value) VALUES
        (dbms_data_mining.algo_name,
         dbms_data_mining.algo_generalized_linear_model);
    COMMIT;
END;
/

DECLARE
    xforms   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_BIN_NUM (bin_table_name => 'bin_tbl', 
                                               xform_list =>  xforms, 
                                               literal_flag =>  TRUE);
    dbms_data_mining.CREATE_MODEL (model_name => 'glm_model', 
                                    mining_function => dbms_data_mining.regression, 
                                    data_table_name => 'mining_data',
                                    case_id_column_name => 'cust_id',
                                    target_column_name => 'affinity_card',
                                    settings_table_name => 'glmsettings',
                                    data_schema_name => null,
                                    settings_schema_name => null,
                                    xform_list => xforms);
END;
/

SELECT attribute_name
    FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));

ATTRIBUTE_NAME
------------------------
AFFINITY_CARD

SELECT expression
    FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));

EXPRESSION
--------------------------------------------------------------------------------
CASE WHEN "AFFINITY_CARD"<0 THEN NULL WHEN "AFFINITY_CARD"<=.25 THEN 1 WHEN "AFFINITY_CARD"<=.5 THEN 2 WHEN "AFFINITY_CARD"<=.75 THEN 3 WHEN "AFFINITY_CARD"<=1 THEN 4 END

SELECT reverse_expression
    FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));

REVERSE_EXPRESSION
--------------------------------------------------------------------------------
44.6.13 INSERT_BIN_NUM_QTILE Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure calls the SQL NTILE function to order the data and divide it equally into the specified number of bins (quantiles).

Insert_BIN_NUM_QTILE bins all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_QTILE (
    bin_table_name       IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    bin_num              IN PLS_INTEGER DEFAULT 10,
    exclude_list         IN COLUMN_LIST DEFAULT NULL,
    bin_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 44-23 INSERT_BIN_NUM_QTILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The following columns are required: COL VARCHAR2(30) VAL NUMBER BIN VARCHAR2(4000) CREATE_BIN_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_NUM_QTILE.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>bin_num</td>
<td>Number of bins. No binning occurs if bin_num is 0 or NULL. The default number of bins is 10.</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from the binning process. If you do not specify exclude_list, all numerical columns in the data source are binned. The format of exclude_list is: dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>
Usage Notes

1. See *Oracle Data Mining User's Guide* for details about numerical data.

2. After dividing the data into quantiles, the `NTILE` function distributes any remainder values one for each quantile, starting with the first. See *Oracle Database SQL Language Reference* for details.

3. Columns with all `NULL` values are ignored by `INSERT_BIN_NUM_QTILE`.

Examples

In this example, `INSERT_BIN_NUM_QTILE` computes the bin boundaries for the `cust_year_of_birth` and `cust_credit_limit` columns in `sh.customers` and inserts the transformations in a transformation definition table. The `STACK_BIN_NUM` Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in `STACK_VIEW`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```sql
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
FROM sh.customers;

DESCRIBE mining_data
Name                                    Null?    Type
--------------------------------------- -------- -----------------------------
CUST_ID                                 NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                      NOT NULL NUMBER(4)
CUST_CREDIT_LIMIT                       NUMBER
CUST_CITY                               NOT NULL VARCHAR2(30)
BEGIN
  dbms_data_mining_transform.CREATE_BIN_NUM(;
  bin_table_name   => 'bin_tbl');
  dbms_data_mining_transform.INSERT_BIN_NUM_QTILE (;
  bin_table_name   => 'bin_tbl',
  data_table_name  => 'mining_data',
  bin_num          => 3,
  exclude_list     => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/

set numwidth 8
column val off
column col format a20
column bin format a10
SELECT col, val, bin
FROM bin_tbl
ORDER BY col ASC, val ASC;

COL                       VAL BIN
-------------------- -------- ----------
CUST_CREDIT_LIMIT        1500
CUST_CREDIT_LIMIT        3000 1
CUST_CREDIT_LIMIT        9000 2
CUST_CREDIT_LIMIT       15000 3
CUST_YEAR_OF_BIRTH       1913
CUST_YEAR_OF_BIRTH       1949 1
```
CUST_YEAR_OF_BIRTH       1965 2
CUST_YEAR_OF_BIRTH       1990 3

DECLARE
    xforms   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_BIN_NUM (
        bin_table_name        => 'bin_tbl',
        xform_list            =>  xforms);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list            =>  xforms,
        data_table_name       => 'mining_data',
        xform_view_name       => 'stack_view');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name in 'STACK_VIEW';

TEXT

44.6.14 INSERT_BIN_SUPER Procedure

This procedure performs numerical and categorical binning and inserts the transformation definitions in transformation definition tables. The procedure computes bin boundaries based on intrinsic relationships between predictors and a target.

INSERT_BIN_SUPER uses an intelligent binning technique known as supervised binning. It builds a single-predictor decision tree and derives the bin boundaries from splits within the tree.

INSERT_BIN_SUPER bins all the VARCHAR2, CHAR, NUMBER, and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_SUPER ( 
    num_table_name        IN VARCHAR2,
    cat_table_name        IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    target_column_name    IN VARCHAR2,
    max_bin_num           IN PLS_INTEGER  DEFAULT 1000,
    exclude_list          IN COLUMN_LIST  DEFAULT NULL,
    num_schema_name       IN VARCHAR2  DEFAULT NULL,
    cat_schema_name       IN VARCHAR2  DEFAULT NULL,
    data_schema_name      IN VARCHAR2  DEFAULT NULL,
    rem_table_name        IN VARCHAR2  DEFAULT NULL,
    rem_schema_name       IN VARCHAR2  DEFAULT NULL);
Parameters

Table 44-24  INSERT_BIN_SUPER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| num_table_name    | Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The following columns are required:  
|                   | COL       VARCHAR2(30)  
|                   | VAL       VNUMBER  
|                   | BIN       VARCHAR2(4000)  
|                   | CREATE_BIN_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_SUPER.  
| cat_table_name    | Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table. The following columns are required:  
|                   | COL       VARCHAR2(30)  
|                   | VAL       VARCHAR2(4000)  
|                   | BIN       VARCHAR2(4000)  
|                   | CREATE_BIN_CAT creates an additional column, ATT, which is used for specifying nested attributes. This column is not used by INSERT_BIN_SUPER.  
| data_table_name   | Name of the table containing the data to be transformed  
| target_column_name| Name of a column to be used as the target for the decision tree models  
| max_bin_num       | The maximum number of bins. The default is 1000.  
| exclude_list      | List of columns to be excluded from the binning process. If you do not specify exclude_list, all numerical and categorical columns in the data source are binned.  
|                   | The format of exclude_list is:  
|                   | dbms_data_mining_transform.COLUMN_LIST('col1','col2',...'coln')  
| num_schema_name   | Schema of num_table_name. If no schema is specified, the current schema is used.  
| cat_schema_name   | Schema of cat_table_name. If no schema is specified, the current schema is used.  
| data_schema_name  | Schema of data_table_name. If no schema is specified, the current schema is used.  
| rem_table_name    | Name of a column removal definition table. The table must have the columns described in "CREATE_COL_REM Procedure". You can use CREATE_COL_REM to create the table. See Usage Notes.  
| rem_schema_name   | Schema of rem_table_name. If no schema is specified, the current schema is used.  

Chapter 44  Summary of DBMS_DATA_MINING_TRANSFORM Subprograms

44-46
Usage Notes

1. See Oracle Data Mining User’s Guide for details about numerical and categorical data.

2. Columns that have no significant splits are not binned. You can remove the unbinned columns from the mining data by specifying a column removal definition table. If you do not specify a column removal definition table, the unbinned columns remain in the mining data.

3. See Oracle Data Mining Concepts to learn more about decision trees in Oracle Data Mining

Examples

In this example, INSERT BIN SUPER computes the bin boundaries for predictors of cust_credit_limit and inserts the transformations in transformation definition tables. One predictor is numerical, the other is categorical. (INSERT_BIN_SUPER determines that the cust_postal_code column is not a significant predictor.) STACK procedures create transformation lists from the contents of the definition tables.

The SQL expressions that compute the transformations are shown in the views MINING_DATA_STACK_NUM and MINING_DATA_STACK_CAT. The views are for display purposes only; they cannot be used to embed the transformations in a model.

CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_marital_status,
           cust_postal_code, cust_credit_limit
    FROM sh.customers;

DESCRIBE mining_data

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>NOT NULL</td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td></td>
<td>VARCHAR2(20)</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>CUST_CREDIT_LIMIT</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

BEGIN
    dbms_data_mining_transform.CREATE_BIN_NUM(
        bin_table_name => 'bin_num_tbl');
    dbms_data_mining_transform.CREATE_BIN_CAT(
        bin_table_name => 'bin_cat_tbl');
    dbms_data_mining_transform.CREATE_COL_REM(
        rem_table_name => 'rem_tbl');
END;
/

BEGIN
    COMMIT;
    dbms_data_mining_transform.INSERT_BIN_SUPER {
        num_table_name => 'bin_num_tbl',
        cat_table_name => 'bin_cat_tbl',
        data_table_name => 'mining_data',
        target_column_name => 'cust_credit_limit',
        max_bin_num => 4,
        exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'),
        num_schema_name => 'dmuser',
        cat_schema_name => 'dmuser',
        data_schema_name => 'dmuser',
    }
END;

rem_table_name => 'rem_tbl',
rem_schema_name => 'dmuser');

COMMIT;
END;
/

set numwidth 8
column val off
SELECT col, val, bin FROM bin_num_tbl
  ORDER BY bin ASC;

<table>
<thead>
<tr>
<th>COL</th>
<th>VAL</th>
<th>BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1923.5</td>
<td>1</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1923.5</td>
<td>1</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1945.5</td>
<td>2</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1980.5</td>
<td>3</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

column val on
column val format a20
SELECT col, val, bin FROM bin_cat_tbl
  ORDER BY bin ASC;

<table>
<thead>
<tr>
<th>COL</th>
<th>VAL</th>
<th>BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>married</td>
<td>1</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>single</td>
<td>2</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>Mar-AF</td>
<td>3</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>Mabsent</td>
<td>3</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>Divorc.</td>
<td>3</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>Married</td>
<td>3</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>Widowed</td>
<td>3</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>NeverM</td>
<td>3</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>Separ.</td>
<td>3</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>divorced</td>
<td>4</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>widow</td>
<td>4</td>
</tr>
</tbody>
</table>

SELECT col from rem_tbl;

<table>
<thead>
<tr>
<th>COL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_POSTAL_CODE</td>
</tr>
</tbody>
</table>

DECLARE
  xforms_num   dbms_data_mining_transform.TRANSFORM_LIST;
  xforms_cat   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_NUM (
    bin_table_name => 'bin_num_tbl',
    xform_list    => xforms_num);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list    => xforms_num,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_num');
  dbms_data_mining_transform.STACK_BIN_CAT (
    bin_table_name => 'bin_cat_tbl',
    xform_list    => xforms_cat);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list    => xforms_cat,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_cat');
END;
44.6.15 INSERT_CLIP_TRIM_TAIL Procedure

This procedure replaces numeric outliers with nulls and inserts the transformation definitions in a transformation definition table.

INSERT_CLIP_TRIM_TAIL computes the boundaries of the data based on a specified percentage. It removes the values that fall outside the boundaries (tail values) from the data. If you wish to replace the tail values instead of removing them, use the INSERT_CLIP_WINSOR_TAIL Procedure.

INSERT_CLIP_TRIM_TAIL clips all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_TRIM_TAIL (
  clip_table_name IN VARCHAR2,
  data_table_name IN VARCHAR2,
  tail_frac IN NUMBER DEFAULT 0.025,
  exclude_list IN COLUMN_LIST DEFAULT NULL,
  clip_schema_name IN VARCHAR2 DEFAULT NULL,
  data_schema_name IN VARCHAR2 DEFAULT NULL);

Chapter 44

Summary of DBMS_DATA_MINING_TRANSFORM Subprograms
Parameters

### Table 44-25 INSERT_CLIP_TRIM_TAIL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip_table_name</td>
<td>Name of the transformation definition table for numerical clipping. You can use the CREATE_CLIP Procedure to create the definition table. The following columns are required: COL VARCHAR2(30) LCUT NUMBER LVAL NUMBER RCUUT NUMBER RVAL NUMBER</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>tail_frac</td>
<td>The percentage of non-null values to be designated as outliers at each end of the data. For example, if tail_frac is .01, then 1% of the data at the low end and 1% of the data at the high end will be treated as outliers. If tail_frac is greater than or equal to .5, no clipping occurs. The default value of tail_frac is 0.025.</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from the clipping process. If you do not specify exclude_list, all numerical columns in the data are clipped. The format of exclude_list is: dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')</td>
</tr>
<tr>
<td>clip_schema_name</td>
<td>Schema of clip_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

**Usage Notes**

1. See *Oracle Data Mining User's Guide* for details about numerical data.
2. The DBMS_DATA_MINING_TRANSFORM package provides two clipping procedures: INSERT_CLIP_TRIM_TAIL and INSERT_CLIP_WINSOR_TAIL. Both procedures compute the boundaries as follows:
   - Count the number of non-null values, $n$, and sort them in ascending order
   - Calculate the number of outliers, $t$, as $n \times \text{tail_frac}$
   - Define the lower boundary $lcut$ as the value at position $1 + \text{floor}(t)$
   - Define the upper boundary $rcut$ as the value at position $n - \text{floor}(t)$
   (The SQL FLOOR function returns the largest integer less than or equal to $t$.)
• All values that are <= \( lcut \) or => \( rcut \) are designated as outliers.

\[ \text{INSERT_CLIP_TRIM_TAIL} \] replaces the outliers with nulls, effectively removing them from the data.

\[ \text{INSERT_CLIP_WINSOR_TAIL} \] assigns \( lcut \) to the low outliers and \( rcut \) to the high outliers.

Examples

In this example, \[ \text{INSERT_CLIP_TRIM_TAIL} \] trims 10% of the data in two columns (5% from the high end and 5% from the low end) and inserts the transformations in a transformation definition table. The \[ \text{STACK_CLIP} \] Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the trimming is shown in the view \[ \text{MINING_DATA_STACK} \]. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
  FROM sh.customers;

DESCRIBE mining_data
Name                            Null?    Type
------------------------------- -------- -------------------
CUST_ID                         NOT NULL NUMBER
CUST_YEAR_OF_BIRTH              NOT NULL NUMBER(4)
CUST_CREDIT_LIMIT                        NUMBER
CUST_CITY                       NOT NULL VARCHAR2(30)
BEGIN
  dbms_data_mining_transform.CREATE_CLIP(
    clip_table_name => 'clip_tbl');
  dbms_data_mining_transform.INSERT_CLIP_TRIM_TAIL(
    clip_table_name => 'clip_tbl',
    data_table_name => 'mining_data',
    tail_frac => 0.05,
    exclude_list => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/
```

```
SELECT col, lcut, lval, rcut, rval
FROM clip_tbl
ORDER BY col ASC;
```

```
COL                      LCUT     LVAL     RCUT     RVAL
-------------------- -------- -------- -------- --------
CUST_CREDIT_LIMIT        1500             11000
CUST_YEAR_OF_BIRTH       1934              1982
```

```
DECLARE
  xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_CLIP (
    clip_table_name => 'clip_tbl',
    xform_list => xforms);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list => xforms,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack');
END;
```

Chapter 44
Summary of DBMS_DATA_MINING_TRANSFORM Subprograms
44-51
set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

---

SELECT "CUST_ID", CASE WHEN "CUST_YEAR_OF_BIRTH" < 1934 THEN NULL WHEN "CUST_YEAR_OF_BIRTH" > 1982 THEN NULL ELSE "CUST_YEAR_OF_BIRTH" END "CUST_YEAR_OF_BIRTH", CASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN NULL WHEN "CUST_CREDIT_LIMIT" > 11000 THEN NULL ELSE "CUST_CREDIT_LIMIT" END "CUST_CREDIT_LIMIT", "CUST_CITY" FROM mining_data

44.6.16 INSERT_CLIP_WINSOR_TAIL Procedure

This procedure replaces numeric outliers with the upper or lower boundary values. It inserts the transformation definitions in a transformation definition table.

INSERT_CLIP_WINSOR_TAIL computes the boundaries of the data based on a specified percentage. It replaces the values that fall outside the boundaries (tail values) with the related boundary value. If you wish to set tail values to null, use the INSERT_CLIP_TRIM_TAIL Procedure.

INSERT_CLIP_WINSOR_TAIL clips all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_WINSOR_TAIL (  
    clip_table_name    IN VARCHAR2,  
    data_table_name    IN VARCHAR2,  
    tail_frac          IN NUMBER DEFAULT 0.025,  
    exclude_list       IN COLUMN_LIST DEFAULT NULL,  
    clip_schema_name   IN VARCHAR2 DEFAULT NULL,  
    data_schema_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 44-26 INSERT_CLIP_WINSOR_TAIL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip_table_name</td>
<td>Name of the transformation definition table for numerical clipping. You can use the CREATE_CLIP Procedure to create the definition table. The following columns are required: COL VARCHAR2(30), LCUT NUMBER, LVAL NUMBER, RCUT NUMBER, RVAL NUMBER. CREATE_CLIP creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_CLIP_WINSOR_TAIL.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
</tbody>
</table>
Table 44-26 (Cont.) INSERT_CLIP_WINSOR_TAIL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tail_frac</td>
<td>The percentage of non-null values to be designated as outliers at each end of the data. For example, if tail_frac is .01, then 1% of the data at the low end and 1% of the data at the high end will be treated as outliers. If tail_frac is greater than or equal to .5, no clipping occurs. The default value of tail_frac is 0.025.</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from the clipping process. If you do not specify exclude_list, all numerical columns in the data are clipped. The format of exclude_list is: dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')</td>
</tr>
<tr>
<td>clip_schema_name</td>
<td>Schema of clip_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User’s Guide for details about numerical data.

2. The DBMS_DATA_MINING_TRANSFORM package provides two clipping procedures: INSERT_CLIP_WINSOR_TAIL and INSERT_CLIP_TRIM_TAIL. Both procedures compute the boundaries as follows:
   - Count the number of non-null values, \( n \), and sort them in ascending order
   - Calculate the number of outliers, \( t \), as \( n \times \text{tail}_\text{frac} \)
   - Define the lower boundary \( lcut \) as the value at position \( 1 + \text{floor}(t) \)
   - Define the upper boundary \( rcut \) as the value at position \( n - \text{floor}(t) \)
   (The SQL FLOOR function returns the largest integer less than or equal to \( t \).)
   - All values that are \( \leq lcut \) or \( \geq rcut \) are designated as outliers.

   INSERT_CLIP_WINSOR_TAIL assigns \( lcut \) to the low outliers and \( rcut \) to the high outliers.
   INSERT_CLIP_TRIM_TAIL replaces the outliers with nulls, effectively removing them from the data.

Examples

In this example, INSERT_CLIP_WINSOR_TAIL winsorizes 10% of the data in two columns (5% from the high end, and 5% from the low end) and inserts the transformations in a transformation definition table. The STACK_CLIP Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
FROM sh.customers;

describe mining_data
Name                   Null?    Type
---------------------- -------- ---------------
CUST_ID                NOT NULL NUMBER
CUST_YEAR_OF_BIRTH     NOT NULL NUMBER(4)
CUST_CREDIT_LIMIT      NUMBER
CUST_CITY              NOT NULL VARCHAR2(30)

BEGIN
  dbms_data_mining_transform.CREATE_CLIP(
    clip_table_name => 'clip_tbl');
  dbms_data_mining_transform.INSERT_CLIP_WINSOR_TAIL(
    clip_table_name => 'clip_tbl',
    data_table_name => 'mining_data',
    tail_frac       => 0.05,
    exclude_list    => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/

SELECT col, lcut, lval, rcut, rval FROM clip_tbl
ORDER BY col ASC;
COL                                LCUT     LVAL     RCUT     RVAL
------------------------------ -------- -------- -------- --------
CUST_CREDIT_LIMIT                  1500     1500    11000    11000
CUST_YEAR_OF_BIRTH                 1934     1934     1982     1982

DECLARE
  xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_CLIP (
    clip_table_name    => 'clip_tbl',
    xform_list        => xforms);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list         => xforms,
    data_table_name    => 'mining_data',
    xform_view_name    => 'mining_data_stack');
END;
/

set long 3000
SQL> SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
-------------------------------------------------------------------
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH" < 1934 THEN 1934 WHEN "CUST_YEAR_OF_BIRTH" > 1982 THEN 1982 ELSE "CUST_YEAR_OF_BIRTH" END "CUST_YEAR_OF_BIRTH",CASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN 1500 WHEN "CUST_CREDIT_LIMIT" > 11000 THEN 11000 ELSE "CUST_CREDIT_LIMIT" END "CUST_CREDIT_LIMIT","CUST_CITY" FROM mining_data
44.6.17 INSERT_MISS_CAT_MODE Procedure

This procedure replaces missing categorical values with the value that occurs most frequently in the column (the mode). It inserts the transformation definitions in a transformation definition table.

INSERT_MISS_CAT_MODE replaces missing values in all VARCHAR2 and CHAR columns in the data source unless you specify a list of columns to ignore.

Syntax

DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE (miss_table_name IN VARCHAR2,
data_table_name IN VARCHAR2,
exclude_list IN COLUMN_LIST DEFAULT NULL,
miss_schema_name IN VARCHAR2 DEFAULT NULL,
data_schema_name IN VARCHAR2 DEFAULT NULL);

Parameters

Table 44-27 INSERT_MISS_CAT_MODE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table for categorical missing value treatment. You can use the CREATE_MISS_CAT Procedure to create the definition table. The following columns are required: CREATE_MISS_CAT creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_MISS_CAT_MODE.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of categorical columns to be excluded from missing value treatment. If you do not specify exclude_list, all categorical columns are transformed. The format of exclude_list is:</td>
</tr>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User's Guide for details about categorical data.
2. If you wish to replace categorical missing values with a value other than the mode, you can edit the transformation definition table.
Example

In this example, INSERT_MISS_CAT_MODE computes missing value treatment for cust_city and inserts the transformation in a transformation definition table. The STACK_MISS_CAT Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_city
    FROM sh.customers;

describe mining_data
Name                             Null?    Type
-------------------------------- -------- ----------------
CUST_ID                          NOT NULL NUMBER
CUST_YEAR_OF_BIRTH               NOT NULL NUMBER(4)
CUST_CITY                        NOT NULL VARCHAR2(30)

BEGIN
    dbms_data_mining_transform.create_miss_cat(
        miss_table_name => 'missc_tbl');
    dbms_data_mining_transform.insert_miss_cat_mode(
        miss_table_name => 'missc_tbl',
        data_table_name => 'mining_data');
END;
/

SELECT stats_mode(cust_city) FROM mining_data;

STATS_MODE(CUST_CITY)
---------------------
Los Angeles

SELECT col, val
    FROM missc_tbl;

COL                             VAL
----------------------------------------
CUST_CITY                       Los Angeles

DECLARE
    xforms     dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_MISS_CAT (
        miss_table_name => 'missc_tbl',
        xform_list => xforms);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list => xforms,
        data_table_name => 'mining_data',
xform_view_name  => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

44.6.18 INSERT_MISS_NUM_MEAN Procedure

This procedure replaces missing numerical values with the average (the mean) and inserts the transformation definitions in a transformation definition table.

INSERT_MISS_NUM_MEAN replaces missing values in all NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_NUM_MEAN (  
    miss_table_name IN VARCHAR2,  
    data_table_name IN VARCHAR2,  
    exclude_list IN COLUMN_LIST DEFAULT NULL,  
    round_num IN PLS_INTEGER DEFAULT 6,  
    miss_schema_name IN VARCHAR2 DEFAULT NULL,  
    data_schema_name IN VARCHAR2 DEFAULT NULL);

Parameters

Table 44-28  INSERT_MISS_NUM_MEAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table for numerical missing value treatment. You can use the CREATE_MISS_NUM Procedure to create the definition table. The following columns are required by INSERT_MISS_NUM_MEAN: COL VARCHAR2(30) VAL NUMBER</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from missing value treatment. If you do not specify exclude_list, all numerical columns are transformed. The format of exclude_list is: dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')</td>
</tr>
<tr>
<td>round_num</td>
<td>The number of significant digits to use for the mean. The default number is 6.</td>
</tr>
</tbody>
</table>
Table 44-28  (Cont.) INSERT_MISS_NUM_MEAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See Oracle Data Mining User’s Guide for details about numerical data.
2. If you wish to replace numerical missing values with a value other than the mean, you can edit the transformation definition table.

Example

In this example, INSERT_MISS_NUM_MEAN computes missing value treatment for cust_year_of_birth and inserts the transformation in a transformation definition table. The STACK_MISS_NUM Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_year_of_birth, cust_city
FROM sh.customers;

DESCRIPTION mining_data
Name           Null?   Type
----------------  ----------  -------------------
CUST_ID         NOT NULL  NUMBER
CUST_YEAR_OF_BIRTH NOT NULL  NUMBER(4)
CUST_CITY       NOT NULL  VARCHAR2(30)

BEGIN
  dbms_data_mining_transform.create_miss_num(
    miss_table_name => 'missn_tbl');
  dbms_data_mining_transform.insert_miss_num_mean(
    miss_table_name => 'missn_tbl',
    data_table_name => 'mining_data',
    exclude_list    => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/

set numwidth 4
column val off
SELECT col, val
FROM missn_tbl;

<table>
<thead>
<tr>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1957</td>
</tr>
</tbody>
</table>

SELECT avg(cust_year_of_birth) FROM mining_data;

| AVG(CUST_YEAR_OF_BIRTH) | 1957    |

DECLARE
xforms dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_MISS_NUM (miss_table_name => 'missn_tbl',
                                          xform_list => xforms);
  dbms_data_mining_transform.XFORM_STACK (xform_list => xforms,
                                          data_table_name => 'mining_data',
                                          xform_view_name => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT

44.6.19 INSERT_NORM_LIN_MINMAX Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table.

INSERT_NORM_LIN_MINMAX computes the minimum and maximum values from the data and sets the value of shift and scale as follows:

\[
\begin{align*}
shift &= \text{min} \\
\text{scale} &= \text{max} - \text{min}
\end{align*}
\]

Normalization is computed as:

\[
x_{\text{new}} = \frac{(x_{\text{old}} - \text{shift})}{\text{scale}}
\]

INSERT_NORM_LIN_MINMAX rounds the value of scale to a specified number of significant digits before storing it in the transformation definition table.

INSERT_NORM_LIN_MINMAX normalizes all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_MINMAX (norm_table_name IN VARCHAR2, data_table_name IN VARCHAR2,
```
Parameters

Table 44-29  INSERT_NORM_LIN_MINMAX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm_table_name</td>
<td>Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The following columns are required: COL VARCHAR2(30), SHIFT NUMBER, SCALE NUMBER. CREATE_NORM_LIN creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_NORM_LIN_MINMAX.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from normalization. If you do not specify exclude_list, all numerical columns are transformed. The format of exclude_list is: dbms_data_mining_transform.COLUMN_LIST('col1','col2',...'coln')</td>
</tr>
<tr>
<td>round_num</td>
<td>The number of significant digits to use for the minimum and maximum. The default number is 6.</td>
</tr>
<tr>
<td>norm_schema_name</td>
<td>Schema of norm_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See Oracle Data Mining User's Guide for details about numerical data.

Examples

In this example, INSERT_NORM_LIN_MINMAX normalizes the cust_year_of_birth column and inserts the transformation in a transformation definition table. The STACK_NORM_LIN Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_gender, cust_year_of_birth
FROM sh.customers;
```

describe mining_data
### 44.6.20 INSERT_NORM_LIN_SCALE Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table.

**INSERT_NORM_LIN_SCALE** computes the minimum and maximum values from the data and sets the value of *shift* and *scale* as follows:

- \( \text{shift} = 0 \)
- \( \text{scale} = \max(\text{abs}(\text{max}), \text{abs}(\text{min})) \)

Normalization is computed as:

\[ x_{\text{new}} = \frac{x_{\text{old}}}{\text{scale}} \]
**INSERT_NORM_LIN_SCALE** rounds the value of *scale* to a specified number of significant digits before storing it in the transformation definition table.

**INSERT_NORM_LIN_SCALE** normalizes all the *NUMBER* and *FLOAT* columns in the data source unless you specify a list of columns to ignore.

**Syntax**

```sql
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_SCALE (
   norm_table_name     IN VARCHAR2,
   data_table_name     IN VARCHAR2,
   exclude_list        IN COLUMN_LIST DEFAULT NULL,
   round_num           IN PLS_INTEGER DEFAULT 6,
   norm_schema_name    IN VARCHAR2 DEFAULT NULL,
   data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm_table_name</td>
<td>Name of the transformation definition table for linear normalization. You can use the <strong>CREATE_NORM_LIN Procedure</strong> to create the definition table. The following columns are required:</td>
</tr>
<tr>
<td></td>
<td>COL       VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td>SHIFT     NUMBER</td>
</tr>
<tr>
<td></td>
<td>SCALE     NUMBER</td>
</tr>
<tr>
<td></td>
<td><strong>CREATE_NORM_LIN</strong> creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by <strong>INSERT_NORM_LIN_SCALE</strong>.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from normalization. If you do not specify <strong>exclude_list</strong>, all numerical columns are transformed.</td>
</tr>
<tr>
<td></td>
<td>The format of <strong>exclude_list</strong> is:</td>
</tr>
<tr>
<td></td>
<td><code>dbms_data_mining_transform.COLUMN_LIST('col1','col2',...,'coln')</code></td>
</tr>
<tr>
<td>round_num</td>
<td>The number of significant digits to use for <em>scale</em>. The default number is 6.</td>
</tr>
<tr>
<td>norm_schema_name</td>
<td>Schema of <strong>norm_table_name</strong>. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of <strong>data_table_name</strong>. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

**Usage Notes**

See *Oracle Data Mining User's Guide* for details about numerical data.

**Examples**

In this example, **INSERT_NORM_LIN_SCALE** normalizes the *cust_year_of_birth* column and inserts the transformation in a transformation definition table. The
STACK_NORM_LIN Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_gender, cust_year_of_birth
    FROM sh.customers;
DESCRIBE mining_data
Name                  Null?    Type
--------------------- -------- ------------------
CUST_ID               NOT NULL NUMBER
CUST_GENDER           NOT NULL CHAR(1)
CUST_YEAR_OF_BIRTH    NOT NULL NUMBER(4)
BEGIN
    dbms_data_mining_transform.CREATE_NORM_LIN(
        norm_table_name => 'norm_tbl');
    dbms_data_mining_transform.INSERT_NORM_LIN_SCALE(
        norm_table_name => 'norm_tbl',
        data_table_name => 'mining_data',
        exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'),
        round_num => 3);
END;
/
SELECT col, shift, scale FROM norm_tbl;

<table>
<thead>
<tr>
<th>COL</th>
<th>SHIFT</th>
<th>SCALE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>0</td>
<td>1990</td>
</tr>
</tbody>
</table>
DECLARE
    xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_NORM_LIN(
        norm_table_name => 'norm_tbl',
        xform_list => xforms);
    dbms_data_mining_transform.XFORM_STACK(
        xform_list => xforms,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack');
END;
/
set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_GENDER",("CUST_YEAR_OF_BIRTH"-0)/1990 "CUST_YEAR_OF_BIRTH" FROM mining_data
44.6.21 INSERT_NORM_LIN_ZSCORE Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table.

**INSERT_NORM_LIN_ZSCORE** computes the mean and the standard deviation from the data and sets the value of shift and scale as follows:

\[
\begin{align*}
\text{shift} & = \text{mean} \\
\text{scale} & = \text{stddev}
\end{align*}
\]

Normalization is computed as:

\[
x_{\text{new}} = \frac{x_{\text{old}} - \text{shift}}{\text{scale}}
\]

**INSERT_NORM_LIN_ZSCORE** rounds the value of scale to a specified number of significant digits before storing it in the transformation definition table.

**INSERT_NORM_LIN_ZSCORE** normalizes all the NUMBER and FLOAT columns in the data unless you specify a list of columns to ignore.

**Syntax**

```sql
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_ZSCORE (
    norm_table_name     IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    exclude_list        IN COLUMN_LIST DEFAULT NULL,
    round_num           IN PLS_INTEGER DEFAULT 6,
    norm_schema_name    IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

**Table 44-31  INSERT_NORM_LIN_ZSCORE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm_table_name</td>
<td>Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The following columns are required:</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to be excluded from normalization. If you do not specify exclude_list, all numerical columns are transformed. The format of exclude_list is:</td>
</tr>
<tr>
<td>round_num</td>
<td>Specifies the number of significant digits to which the scale value is rounded.</td>
</tr>
<tr>
<td>norm_schema_name</td>
<td>Name of the schema containing the transformation definition table.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Name of the schema containing the data table.</td>
</tr>
</tbody>
</table>

**CREATE_NORM_LIN** creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by **INSERT_NORM_LIN_ZSCORE**.
Table 44-31  (Cont.) INSERT_NORM_LIN_ZSCORE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>round_num</td>
<td>The number of significant digits to use for scale. The default number is 6.</td>
</tr>
<tr>
<td>norm_schema_name</td>
<td>Schema of norm_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See Oracle Data Mining User's Guide for details about numerical data.

Examples

In this example, INSERT_NORM_LIN_ZSCORE normalizes the cust_year_of_birth column and inserts the transformation in a transformation definition table. The STACK_NORM_LIN Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_gender, cust_year_of_birth
FROM sh.customers;

DESCRIBE mining_data

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUST_GENDER</td>
<td>NOT NULL</td>
<td>CHAR(1)</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>NOT NULL</td>
<td>NUMBER(4)</td>
</tr>
</tbody>
</table>

BEGIN
  dbms_data_mining_transform.CREATE_NORM_LIN(
    norm_table_name => 'norm_tbl');
  dbms_data_mining_transform.INSERT_NORM_LIN_ZSCORE(
    norm_table_name => 'norm_tbl',
    data_table_name => 'mining_data',
    exclude_list     => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
    round_num        => 3);
END;
/

SELECT col, shift, scale FROM norm_tbl;

<table>
<thead>
<tr>
<th>COL</th>
<th>SHIFT</th>
<th>SCALE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1960</td>
<td>15</td>
</tr>
</tbody>
</table>

DECLARE
  xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_NORM_LIN (
    norm_table_name => 'norm_tbl',
    ...)
44.6.22 SET_EXPRESSION Procedure

This procedure appends a row to a VARCHAR2 array that stores a SQL expression.

The array can be used for specifying a transformation expression that is too long to be used with the SET_TRANSFORM Procedure.

The GET_EXPRESSION Function returns a row in the array.

When you use SET_EXPRESSION to build a transformation expression, you must build a corresponding reverse transformation expression, create a transformation record, and add the transformation record to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.SET_EXPRESSION (
    expression    IN OUT NOCOPY EXPRESSION_REC,
    chunk                       VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>An expression record (EXPRESSION_REC) that specifies a transformation expression or a reverse transformation expression for an attribute. Each expression record includes a VARCHAR2 array and index fields for specifying upper and lower boundaries within the array. There are two EXPRESSION_REC fields within a transformation record (TRANSFORM_REC): one for the transformation expression; the other for the reverse transformation expression. See Table 44-1 for a description of the EXPRESSION_REC type.</td>
</tr>
<tr>
<td>chunk</td>
<td>A VARCHAR2 chunk (row) to be appended to expression.</td>
</tr>
</tbody>
</table>

Notes

1. You can pass NULL in the chunk argument to SET_EXPRESSION to clear the previous chunk. The default value of chunk is NULL.

2. See "About Transformation Lists".
3. See "Operational Notes".

Examples

In this example, two calls to SET_EXPRESSION construct a transformation expression and two calls construct the reverse transformation.

**Note:**

This example is for illustration purposes only. It shows how SET_EXPRESSION appends the text provided in chunk to the text that already exists in expression. The SET_EXPRESSION procedure is meant for constructing very long transformation expressions that cannot be specified in a VARCHAR2 argument to SET_TRANSFORM.

Similarly while transformation lists are intended for embedding in a model, the transformation list v_xlst is shown in an external view for illustration purposes.

```sql
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_postal_code, cust_credit_limit
  FROM sh.customers;

DECLARE
  v_expr dbms_data_mining_transform.EXPRESSION_REC;
  v_rexp dbms_data_mining_transform.EXPRESSION_REC;
  v_xrec dbms_data_mining_transform.TRANSFORM_REC;
  v_xlst dbms_data_mining_transform.TRANSFORM_LIST :=
    dbms_data_mining_transform.TRANSFORM_LIST(NULL);
BEGIN
  dbms_data_mining_transform.SET_EXPRESSION(
    EXPRESSION  => v_expr,
    CHUNK       => '("CUST_YEAR_OF_BIRTH"-1910)');
  dbms_data_mining_transform.SET_EXPRESSION(
    EXPRESSION  => v_expr,
    CHUNK       => '/77');
  dbms_data_mining_transform.SET_EXPRESSION(
    EXPRESSION  => v_rexp,
    CHUNK       => '"CUST_YEAR_OF_BIRTH"*77');
  dbms_data_mining_transform.SET_EXPRESSION(
    EXPRESSION  => v_rexp,
    CHUNK       => '+1910');
  v_xrec := null;
  v_xrec.attribute_name := 'CUST_YEAR_OF_BIRTH';
  v_xrec.expression := v_expr;
  v_xrec.reverse_expression := v_rexp;
  v_xlst.TRIM;
  v_xlst.extend(1);
  v_xlst(1) := v_xrec;
  dbms_data_mining_transform.XFORM_STACK(
    xform_list => v_xlst,
    data_table_name => 'mining_data',
    xform_view_name => 'v_xlst_view');
  dbms_output.put_line('====');
  FOR i IN 1..v_xlst.count LOOP
    dbms_output.put_line('ATTR: '||v_xlst(i).attribute_name);
  END LOOP;
END;
```
44.6.23 SET_TRANSFORM Procedure

This procedure appends the transformation instructions for an attribute to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.SET_TRANSFORM (  
    xform_list               IN OUT NOCOPY TRANSFORM_LIST,  
    attribute_name           VARCHAR2,  
    attribute_subname        VARCHAR2,  
    expression               VARCHAR2,  
    reverse_expression       VARCHAR2,  
    attribute_spec           VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Name of the attribute to be transformed</td>
</tr>
<tr>
<td>attribute_subname</td>
<td>Name of the nested attribute if attribute_name is a nested column, otherwise NULL.</td>
</tr>
<tr>
<td>expression</td>
<td>A SQL expression that specifies the transformation of the attribute.</td>
</tr>
<tr>
<td>reverse_expression</td>
<td>A SQL expression that reverses the transformation for readability in model details and in the target of a supervised model (if the attribute is a target)</td>
</tr>
</tbody>
</table>
Table 44-33  (Cont.) SET_TRANSFORM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_spec</td>
<td>One or more keywords that identify special treatment for the attribute</td>
</tr>
<tr>
<td></td>
<td>during model build. Values are:</td>
</tr>
<tr>
<td></td>
<td>• NOPREP — When ADP is on, prevents automatic transformation of the attribute.</td>
</tr>
<tr>
<td></td>
<td>• TEXT — Causes the attribute to be treated as unstructured text data</td>
</tr>
<tr>
<td></td>
<td>• FORCE_IN — Forces the inclusion of the attribute in the model build. Applies only to GLM models with feature selection enabled (ftr_selection_enable = yes). Feature selection is disabled by default.</td>
</tr>
<tr>
<td></td>
<td>If the model is not using GLM with feature selection, this value has no effect.</td>
</tr>
<tr>
<td></td>
<td>See “Specifying Transformation Instructions for an Attribute” in Oracle Data Mining User’s Guide for more information about attribute_spec.</td>
</tr>
</tbody>
</table>

Usage Notes

1. See the following relevant sections in "Operational Notes":
   • About Transformation Lists
   • Nested Data Transformations

2. As shown in the following example, you can eliminate an attribute by specifying a null transformation expression and reverse expression. You can also use the STACK interface to remove a column (CREATE_COL_REM Procedure and STACK_COL_REM Procedure).

44.6.24 STACK_BIN_CAT Procedure

This procedure adds categorical binning transformations to a transformation list.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.STACK_BIN_CAT (  
  bin_table_name     IN            VARCHAR2,  
  xform_list         IN OUT NOCOPY TRANSFORM_LIST,  
  literal_flag       IN            BOOLEAN  DEFAULT FALSE,  
  bin_schema_name    IN            VARCHAR2 DEFAULT NULL);  
```
Parameters

Table 44-34  STACK_BIN_CAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call STACK_BIN_CAT. To populate the table, you can use one of the INSERT procedures for categorical binning or you can write your own SQL. See Table 44-4.</td>
</tr>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>literal_flag</td>
<td>Indicates whether the values in the bin column in the transformation definition table are valid SQL literals. When literal_flag is FALSE (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set literal_flag to TRUE if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See &quot;INSERT_BIN_NUM_EQWIDTH Procedure&quot; for an example.</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how a binning transformation for the categorical column cust_postal_code could be added to a stack called mining_data_stack.

```
CREATE or REPLACE VIEW mining_data AS
SELECT cust_id, cust_postal_code, cust_credit_limit
FROM sh.customers
```

Note:

This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. XFORM_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE_MODEL in the xform_list parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.
WHERE cust_id BETWEEN 100050 AND 100100;
BEGIN
    dbms_data_mining_transform.CREATE_BIN_CAT ('bin_cat_tbl');
    dbms_data_mining_transform.INSERT_BIN_CAT_FREQ ( 
        bin_table_name => 'bin_cat_tbl',
        data_table_name => 'mining_data',
        bin_num => 3);
END;
/
DECLARE
    MINING_DATA_STACK dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_BIN_CAT ( 
        bin_table_name       => 'bin_cat_tbl',
        xform_list           => mining_data_stack);
    dbms_data_mining_transform.XFORM_STACK ( 
        xform_list           => mining_data_stack,
        data_table_name      => 'mining_data',
        xform_view_name      => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT * from mining_data
WHERE cust_id BETWEEN 100050 AND 100053
ORDER BY cust_id;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>CUST_POSTAL_CODE</th>
<th>CUST_CREDIT_LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>100050</td>
<td>76486</td>
<td>1500</td>
</tr>
<tr>
<td>100051</td>
<td>73216</td>
<td>9000</td>
</tr>
<tr>
<td>100052</td>
<td>69499</td>
<td>5000</td>
</tr>
<tr>
<td>100053</td>
<td>45704</td>
<td>7000</td>
</tr>
</tbody>
</table>

-- After transformation
SELECT * FROM mining_data_stack_view
WHERE cust_id BETWEEN 100050 AND 100053
ORDER BY cust_id;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>CUST_POSTAL_CODE</th>
<th>CUST_CREDIT_LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>100050</td>
<td>4</td>
<td>1500</td>
</tr>
<tr>
<td>100051</td>
<td>1</td>
<td>9000</td>
</tr>
<tr>
<td>100052</td>
<td>4</td>
<td>5000</td>
</tr>
<tr>
<td>100053</td>
<td>4</td>
<td>7000</td>
</tr>
</tbody>
</table>

### 44.6.25 STACK_BIN_NUM Procedure

This procedure adds numerical binning transformations to a transformation list.

**Syntax**

```sql
DBMS_DATA_MINING_TRANSFORM.STACK_BIN_NUM ( 
    bin_table_name IN VARCHAR2,
    xform_list IN OUT NOCOPY TRANSFORM_LIST,
    literal_flag IN BOOLEAN DEFAULT FALSE,
    bin_schema_name IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 44-35  STACK_BIN_NUM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call STACK_BIN_NUM. To populate the table, you can use one of the INSERT procedures for numerical binning or you can write your own SQL. See Table 44-6.</td>
</tr>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>literal_flag</td>
<td>Indicates whether the values in the bin column in the transformation definition table are valid SQL literals. When literal_flag is FALSE (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set literal_flag to TRUE if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See &quot;INSERT_BIN_NUM_EQWIDTH Procedure&quot; for an example.</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how a binning transformation for the numerical column cust_credit_limit could be added to a stack called mining_data_stack.

Note:

This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. XFORM_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE_MODEL in the xform_list parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_postal_code, cust_credit_limit
FROM sh.customers
WHERE cust_id BETWEEN 100050 and 100100;
BEGIN
  dbms_data_mining_transform.create_bin_num ('bin_num_tbl');
  dbms_data_mining_transform.insert_bin_num_qtile (
    bin_table_name => 'bin_num_tbl',
    data_table_name => 'mining_data',
    bin_num => 5,
    exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/
DECLARE
  MINING_DATA_STACK dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_CAT (
    bin_table_name => 'bin_num_tbl',
    xform_list => mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list => mining_data_stack,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT cust_id, cust_postal_code, ROUND(cust_credit_limit) FROM mining_data
WHERE cust_id BETWEEN 100050 AND 100055
ORDER BY cust_id;
CUST_ID  CUST_POSTAL_CODE  ROUND(CUST_CREDIT_LIMIT)
-------  -----------------  -------------------------
100050   76486               1500
100051   73216               9000
100052   69499               5000
100053   45704               7000
100055   74673               11000
100055   74673               11000
-- After transformation
SELECT cust_id, cust_postal_code, ROUND(cust_credit_limit)
FROM mining_data_stack_view
WHERE cust_id BETWEEN 100050 AND 100055
ORDER BY cust_id;
CUST_ID  CUST_POSTAL_CODE  ROUND(CUST_CREDIT_LIMIT)
-------  ----------------  -------------------------
100050   76486                2
100051   73216                1
100052   69499                3
100053   45704                3
100054   88021                3
100055   74673                3

44.6.26 STACK_CLIP Procedure

This procedure adds clipping transformations to a transformation list.

Syntax

DBMS_DATA_MINING_TRANSFORM.STACK_CLIP {
  clip_table_name IN VARCHAR2,
  xform_list IN OUT NOCOPY TRANSFORM_LIST,
  clip_schema_name IN VARCHAR2 DEFAULT NULL);
Parameters

Table 44-36  STACK_CLIP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip_table_name</td>
<td>Name of the transformation definition table for clipping. You can use the CREATE_CLIP Procedure to create the definition table. The table must be populated with transformation definitions before you call STACK_CLIP. To populate the table, you can use one of the INSERT procedures for clipping or you can write your own SQL. See Table 44-8</td>
</tr>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>clip_schema_name</td>
<td>Schema of clip_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See DBMS_DATA_MINING_TRANSFORM Operational Notes. The following sections are especially relevant:

- “About Transformation Lists”
- “About Stacking”
- “Nested Data Transformations”

Examples

This example shows how a clipping transformation for the numerical column cust_credit_limit could be added to a stack called mining_data_stack.

```
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_postal_code, cust_credit_limit
FROM sh.customers
WHERE cust_id BETWEEN 100050 AND 100100;
BEGIN
    dbms_data_mining_transform.create_clip ('clip_tbl');
    dbms_data_mining_transform.insert_clip_winsor_tail (
        clip_table_name => 'clip_tbl',
        data_table_name => 'mining_data',
        tail_frac       => 0.25,
        exclude_list    => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
```

Note:

This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. XFORM_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE_MODEL in the xform_list parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.
DECLARE
    MINING_DATA_STACK dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_CLIP (
        clip_table_name => 'clip_tbl',
        xform_list => mining_data_stack);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list => mining_data_stack,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack_view');
END;
/

-- Before transformation
SELECT cust_id, custPostal_code, round(cust_credit_limit)
FROM mining_data
WHERE cust_id BETWEEN 100050 AND 100054
ORDER BY cust_id;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>CUST_POSTAL_CODE</th>
<th>ROUND(CUST_CREDIT_LIMIT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100050</td>
<td>76486</td>
<td>1500</td>
</tr>
<tr>
<td>100051</td>
<td>73216</td>
<td>9000</td>
</tr>
<tr>
<td>100052</td>
<td>69499</td>
<td>5000</td>
</tr>
<tr>
<td>100053</td>
<td>45704</td>
<td>7000</td>
</tr>
<tr>
<td>100054</td>
<td>88021</td>
<td>11000</td>
</tr>
</tbody>
</table>

-- After transformation
SELECT cust_id, custPostal_code, round(cust_credit_limit)
FROM mining_data_stack_view
WHERE cust_id BETWEEN 100050 AND 100054
ORDER BY cust_id;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>CUST_POSTAL_CODE</th>
<th>ROUND(CUST_CREDIT_LIMIT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100050</td>
<td>76486</td>
<td>5000</td>
</tr>
<tr>
<td>100051</td>
<td>73216</td>
<td>9000</td>
</tr>
<tr>
<td>100052</td>
<td>69499</td>
<td>5000</td>
</tr>
<tr>
<td>100053</td>
<td>45704</td>
<td>7000</td>
</tr>
<tr>
<td>100054</td>
<td>88021</td>
<td>11000</td>
</tr>
</tbody>
</table>

44.6.27 STACK_COL_REM Procedure

This procedure adds column removal transformations to a transformation list.

Syntax

DBMS_DATA_MINING_TRANSFORM.STACK_COL_REM (rem_table_name IN VARCHAR2,
                                          xform_list IN OUT NOCOPY TRANSFORM_LIST,
                                          rem_schema_name IN VARCHAR2 DEFAULT NULL);
Parameters

Table 44-37  STACK_COL_REM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rem_table_name</td>
<td>Name of the transformation definition table for column removal. You can use the CREATE_COL_REM Procedure to create the definition table. See Table 44-10. The table must be populated with column names before you call STACK_COL_REM. The INSERT_BIN_SUPER Procedure and the INSERT_AUTOBIN_NUM_EQWIDTH Procedure can optionally be used to populate the table. You can also use SQL INSERT statements.</td>
</tr>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>rem_schema_name</td>
<td>Schema of rem_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how the column cust_credit_limit could be removed in a transformation list called mining_data_stack.

Note:

This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. XFORM_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE_MODEL in the xform list parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```sql
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, country_id, cust_postal_code, cust_credit_limit
FROM sh.customers;
BEGIN
    dbms_data_mining_transform.create_col_rem ('rem_tbl');
END;
/

INSERT into rem_tbl VALUES (upper('cust_postal_code'), null);
DECLARE
```
MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
   dbms_data_mining_transform.stack_col_rem (
      rem_table_name     => 'rem_tbl',
      xform_list         => mining_data_stack);
   dbms_data_mining_transform.XFORM_STACK (
      xform_list         => mining_data_stack,
      data_table_name    => 'mining_data',
      xform_view_name    => 'mining_data_stack_view');
END;
/

SELECT *  FROM mining_data
   WHERE cust_id BETWEEN 100050 AND 100051
   ORDER BY cust_id;

CUST_ID   COUNTRY_ID   CUST_POSTAL_CODE   CUST_CREDIT_LIMIT
-------   ----------   ----------------   -----------------
100050         52773   76486                          1500
100051         52790   73216                          9000

SELECT *  FROM mining_data_stack_view
   WHERE cust_id BETWEEN 100050 AND 100051
   ORDER BY cust_id;

CUST_ID   COUNTRY_ID   CUST_CREDIT_LIMIT
-------   ----------   -----------------
100050         52773              1500
100051         52790              9000

44.6.28 STACK_MISS_CAT Procedure

This procedure adds categorical missing value transformations to a transformation list.

Syntax

DBMS_DATA_MINING_TRANSFORM.STACK_MISS_CAT (  
   miss_table_name     IN       VARCHAR2,
   xform_list          IN OUT   NOCOPY TRANSFORM_LIST,
   miss_schema_name    IN       VARCHAR2 DEFAULT NULL);

Parameters

Table 44-38  STACK_MISS_CAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table for categorical missing value treatment. You can use the CREATE_MISS_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call STACK_MISS_CAT. To populate the table, you can use the INSERT_MISS_CAT_MODE Procedure or you can write your own SQL. See Table 44-12.</td>
</tr>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>
Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how the missing values in the column cust_marital_status could be replaced with the mode in a transformation list called mining_data_stack.

This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. XFORM_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE_MODEL in the xform_list parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, country_id, cust_marital_status
    FROM sh.customers
    where cust_id BETWEEN 1 AND 10;
BEGIN
    dbms_data_mining_transform.create_miss_cat ('miss_cat_tbl');
    dbms_data_mining_transform.insert_miss_cat_mode ('miss_cat_tbl',
    'mining_data');
END;
/

DECLARE
    MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.stack_miss_cat (
        miss_table_name   => 'miss_cat_tbl',
        xform_list        => mining_data_stack);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list        =>  mining_data_stack,
        data_table_name   => 'mining_data',
        xform_view_name   => 'mining_data_stack_view');
END;
/
SELECT * FROM mining_data
ORDER BY cust_id;
```

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>COUNTRY_ID</th>
<th>CUST_MARITAL_STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>52789</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>52778</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>52770</td>
<td></td>
</tr>
</tbody>
</table>
44.6.29 STACK_MISS_NUM Procedure

This procedure adds numeric missing value transformations to a transformation list.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.STACK_MISS_NUM (  
  miss_table_name IN VARCHAR2,  
  xform_list OUT NOCOPY TRANSFORM_LIST,  
  miss_schema_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table for numerical missing value treatment. You can use the CREATE_MISS_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call STACK_MISS_NUM. To populate the table, you can use the INSERT_MISS_NUM_MEAN Procedure or you can write your own SQL. See Table 44-14.</td>
</tr>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
Examples

This example shows how the missing values in the column `cust_credit_limit` could be replaced with the mean in a transformation list called `mining_data_stack`.

---

### Note:

This example invokes the `XFORM_STACK Procedure` to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See `INSERT_BIN_NUM_EQWIDTH Procedure` for an example.

---

```
describe mining_data
Name                  Null?    Type
--------------------- -------- ----- 
CUST_ID               NOT NULL NUMBER
CUST_CREDIT_LIMIT                              NUMBER

BEGIN
  dbms_data_mining_transform.create_miss_num ('miss_num_tbl');
  dbms_data_mining_transform.insert_miss_num_mean ('miss_num_tbl','mining_data');
END;
/
SELECT * FROM miss_num_tbl;

COL                  ATT      VAL
-------------------- ----- -----
CUST_ID                       5.5
CUST_CREDIT_LIMIT          185.71

DECLARE
  MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_MISS_NUM (miss_table_name   => 'miss_num_tbl',
                                            xform_list        => mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (xform_list        =>  mining_data_stack,
                                           data_table_name   => 'mining_data',
                                           xform_view_name   => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT * FROM mining_data
ORDER BY cust_id;
CUST_ID CUST_CREDIT_LIMIT
------- -----------------
1       100
2
3       200
```
44.6.30 STACK_NORM_LIN Procedure

This procedure adds linear normalization transformations to a transformation list.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.STACK_NORM_LIN (  
  norm_table_name     IN       VARCHAR2,  
  xform_list          IN OUT   NOCOPY TRANSFORM_LIST,  
  norm_schema_name    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm_table_name</td>
<td>Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The table must be populated with transformation definitions before you call STACK_NORM_LIN. To populate the table, you can use one of the INSERT procedures for normalization or you can write your own SQL. See Table 44-16.</td>
</tr>
<tr>
<td>xform_list</td>
<td>A transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>norm_schema_name</td>
<td>Schema of norm_table_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
Examples

This example shows how the column `cust_credit_limit` could be normalized in a transformation list called `mining_data_stack`.

**Note:**

This example invokes the `XFORM_STACK` Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See `INSERT_BIN_NUM_EQWIDTH Procedure` for an example.

```sql
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, country_id, cust_postal_code, cust_credit_limit
    FROM sh.customers;
BEGIN
    dbms_data_mining_transform.create_norm_lin ('norm_lin_tbl');
    dbms_data_mining_transform.insert_norm_lin_minmax (
        norm_table_name => 'norm_lin_tbl',
        data_table_name => 'mining_data',
        exclude_list    => dbms_data_mining_transform.COLUMN_LIST('cust_id',
                                                                  'country_id'));
END;
/
SELECT * FROM norm_lin_tbl;
COL                  ATT    SHIFT  SCALE
-------------------- ----- ------ ----------
CUST_CREDIT_LIMIT            1500  13500
DECLARE
    MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.stack_norm_lin (
        norm_table_name => 'norm_lin_tbl',
        xform_list      => mining_data_stack);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list      => mining_data_stack,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack_view');
END;
/
SELECT * FROM mining_data
WHERE cust_id between 1 and 10
ORDER BY cust_id;
CUST_ID COUNTRY_ID CUST_POSTAL_CODE CUST_CREDIT_LIMIT
------- ---------- -------------------- -----------------
1      52789 30828                             9000
2      52778 86319                            10000
3      52770 88666                             1500
4      52770 87551                             1500
5      52789 59200                             1500
```
44.6.31 XFORM_BIN_CAT Procedure

This procedure creates a view that implements the categorical binning transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_CAT (
    bin_table_name       IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    xform_view_name      IN VARCHAR2,
    literal_flag         IN BOOLEAN DEFAULT FALSE,
    bin_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL,
    xform_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 44-41 XFORM_BIN_CAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call XFORM_BIN_CAT. To populate the table, you can use one of the INSERT procedures for categorical binning or you can write your own SQL. See Table 44-4.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed.</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in bin_table_name.</td>
</tr>
</tbody>
</table>
Table 44-41  (Cont.) XFORM_BIN_CAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>literal_flag</td>
<td>Indicates whether the values in the bin column in the transformation definition table are valid SQL literals. When literal_flag is FALSE (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set literal_flag to TRUE if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See &quot;INSERT_BIN_NUM_EQWIDTH Procedure&quot; for an example.</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes".

Examples

This example creates a view that bins the cust_postal_code column. The data source consists of three columns from sh.customer.

describe mining_data
Name                                      Null?    Type
---------------------------------------- -------- ------------------------
CUST_ID                                  NOT NULL NUMBER
CUST_POSTAL_CODE                         NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT                        NUMBER

SELECT * FROM mining_data WHERE cust_id between 104066 and 104069;

  CUST_ID CUST_POSTAL_CODE
--------- --------------------
 104066  69776
 7000
 104067  52602
 9000
 104068  55787
11000
 104069  55977
 5000

BEGIN
  dbms_data_mining_transform.create_bin_cat(
    bin_table_name => 'bin_cat_tbl');
  dbms_data_mining_transform.insert_bin_cat_freq(
    bin_table_name => 'bin_cat_tbl',
    data_table_name => 'mining_data',
    bin_num          => 10);
  dbms_data_mining_transform.xform_bin_cat(  

...
bin_table_name => 'bin_cat_tbl',
data_table_name => 'mining_data',
xform_view_name => 'bin_cat_view');

END;
/

SELECT * FROM bin_cat_view WHERE cust_id between 104066 and 104069;

CUST_ID CUST_POSTAL_CODE CUST_CREDIT_LIMIT
---------- ----------------------- ------------------------
          6 7000
104066 11 9000
104067 3 11000
104068 11 5000

SELECT text FROM user_views WHERE view_name IN 'BIN_CAT_VIEW';

TEXT
--------------------------------------------------------------------------------
SELECT
"CUST_ID",DECODE("CUST_POSTAL_CODE","38082',"1","45704","9","48346","5","55787","3","63736","2","67843","7","69776","6","72860","10","78558","4","80841","8",NULL,NULL,"11") "CUST_POSTAL_CODE","CUST_CREDIT_LIMIT" FROM mining_data

44.6.32 XFORM_BIN_NUM Procedure

This procedure creates a view that implements the numerical binning transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_NUM ( bin_table_name IN VARCHAR2,
data_table_name IN VARCHAR2,
xform_view_name IN VARCHAR2,
literal_flag IN BOOLEAN DEFAULT FALSE,
bin_schema_name IN VARCHAR2 DEFAULT NULL,
data_schema_name IN VARCHAR2 DEFAULT NULL,
xform_schema_name IN VARCHAR2 DEFAULT NULL);
Parameters

Table 44-42  XFORM_BIN_NUM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_table_name</td>
<td>Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call XFORM_BIN_NUM. To populate the table, you can use one of the INSERT procedures for numerical binning or you can write your own SQL. See &quot;Table 44-6&quot;.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed.</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in bin_table_name.</td>
</tr>
<tr>
<td>literal_flag</td>
<td>Indicates whether the values in the bin column in the transformation definition table are valid SQL literals. When literal_flag is FALSE (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set literal_flag to TRUE if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See &quot;INSERT_BIN_NUM_EQWIDTH Procedure&quot; for an example.</td>
</tr>
<tr>
<td>bin_schema_name</td>
<td>Schema of bin_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes".

Examples

This example creates a view that bins the cust_credit_limit column. The data source consists of three columns from sh.customer.

describe mining_data
Name Null? Type
----------------------------  -----  ------------------------
CUST_ID                      NOT NULL NUMBER
CUST_POSTAL_CODE             NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT            NUMBER

column cust_credit_limit off
SELECT * FROM mining_data WHERE cust_id between 104066 and 104069;

    CUST_ID CUST_POSTAL_CODE CUST_CREDIT_LIMIT
    ---------  ------------------  --------------
BEGIN
    dbms_data_mining_transform.create_bin_num(
        bin_table_name => 'bin_num_tbl',
    );
    dbms_data_mining_transform.insert_autobin_num_eqwidth(
        bin_table_name => 'bin_num_tbl',
        data_table_name => 'mining_data',
        bin_num => 5,
        max_bin_num => 10,
        exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
    dbms_data_mining_transform.xform_bin_num(
        bin_table_name => 'bin_num_tbl',
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_view');
END;
/

describe mining_data_view

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>CUST_CREDIT_LIMIT</td>
<td>NOT NULL</td>
<td>VARCHAR2(2)</td>
</tr>
</tbody>
</table>

col cust_credit_limit on
col cust_credit_limit format a25
SELECT * FROM mining_data_view WHERE cust_id between 104066 and 104069;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>CUST_POSTAL_CODE</th>
<th>CUST_CREDIT_LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>104066</td>
<td>69776</td>
<td>5</td>
</tr>
<tr>
<td>104067</td>
<td>52602</td>
<td>6</td>
</tr>
<tr>
<td>104068</td>
<td>55787</td>
<td>8</td>
</tr>
<tr>
<td>104069</td>
<td>55977</td>
<td>3</td>
</tr>
</tbody>
</table>

set long 2000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_VIEW';

TEXT

---------------------------------------
SELECT "CUST_ID","CUST_POSTAL_CODE",CASE WHEN "CUST_CREDIT_LIMIT"<1500 THEN NULL WHEN "CUST_CREDIT_LIMIT"<=2850 THEN '1' WHEN "CUST_CREDIT_LIMIT"<=4200 THEN '2' WHEN "CUST_CREDIT_LIMIT"<=5550 THEN '3' WHEN "CUST_CREDIT_LIMIT"<=6900 THEN
WHEN "CUST_CREDIT_LIMIT"<=8250 THEN '5' WHEN "CUST_CREDIT_LIMIT"<=9600 THEN '6' WHEN "CUST_CREDIT_LIMIT"<=10950 THEN '7' WHEN "CUST_CREDIT_LIMIT"<=12300 THEN '8' WHEN "CUST_CREDIT_LIMIT"<=13650 THEN '9' WHEN "CUST_CREDIT_LIMIT"<=15000 THEN '10' END "CUST_CREDIT_LIMIT" FROM mining_data

44.6.33 XFORM_CLIP Procedure

This procedure creates a view that implements the clipping transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

DBMS_DATA_MINING_TRANSFORM.XFORM_CLIP (clip_table_name IN VARCHAR2, data_table_name IN VARCHAR2, xform_view_name IN VARCHAR2, clip_schema_name IN VARCHAR2 DEFAULT NULL, data_schema_name IN VARCHAR2, DEFAULT NULL, xform_schema_name IN VARCHAR2, DEFAULT NULL);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip_table_name</td>
<td>Name of the transformation definition table for clipping. You can use the CREATE_CLIP Procedure to create the definition table. The table must be populated with transformation definitions before you call XFORM_CLIP. To populate the table, you can use one of the INSERT procedures for clipping you can write your own SQL. See Table 44-8.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in clip_table_name.</td>
</tr>
<tr>
<td>clip_schema_name</td>
<td>Schema of clip_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Examples

This example creates a view that clips the cust_credit_limit column. The data source consists of three columns from sh.customer.
describe mining_data
Name                           Null?    Type
------------------------------ -------- -------------------------
CUST_ID                        NOT NULL NUMBER
CUST_POSTAL_CODE               NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT                        NUMBER

BEGIN
    dbms_data_mining_transform.create_clip(
        clip_table_name => 'clip_tbl');
    dbms_data_mining_transform.insert_clip_trim_tail(
        clip_table_name => 'clip_tbl',
        data_table_name => 'mining_data',
        tail_frac => 0.05,
        exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
    dbms_data_mining_transform.xform_clip(
        clip_table_name => 'clip_tbl',
        data_table_name => 'mining_data',
        xform_view_name => 'clip_view');
END;
/

describe clip_view
Name                           Null?    Type
------------------------------ -------- -------------------------
CUST_ID                        NOT NULL NUMBER
CUST_POSTAL_CODE               NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT                        NUMBER

SELECT MIN(cust_credit_limit), MAX(cust_credit_limit) FROM mining_data;

MIN(CUST_CREDIT_LIMIT) MAX(CUST_CREDIT_LIMIT)
---------------------- ----------------------
    1500                  15000

SELECT MIN(cust_credit_limit), MAX(cust_credit_limit) FROM clip_view;

MIN(CUST_CREDIT_LIMIT) MAX(CUST_CREDIT_LIMIT)
---------------------- ----------------------
    1500                  11000

set long 2000
SELECT text FROM user_views WHERE view_name IN 'CLIP_VIEW';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_POSTAL_CODE",CASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN NU LL WHEN "CUST_CREDIT_LIMIT" > 11000 THEN NULL ELSE "CUST_CREDIT_LIMIT" END "CUST _CREDIT_LIMIT" FROM mining_data

44.6.34 XFORM_COL_REM Procedure

This procedure creates a view that implements the column removal transformations specified in a definition table. Only the columns that are specified in the definition table are removed; the remaining columns from the data table are present in the view.

Syntax

DBMS_DATA_MINING_TRANSFORM.XFORM_COL_REM (rem_table_name IN VARCHAR2,
data_table_name IN VARCHAR2,
Parameters

Table 44-44 XFORM_COL_REM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rem_table_name</td>
<td>Name of the transformation definition table for column removal. You can use the CREATE_COL_REM Procedure to create the definition table. See Table 44-10. The table must be populated with column names before you call XFORM_COL_REM. The INSERT_BIN_SUPER Procedure and the INSERT_AUTOBIN_NUM_EQWIDTH Procedure can optionally be used to populate the table. You can also use SQL INSERT statements.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents the columns in data_table_name that are not specified in rem_table_name.</td>
</tr>
<tr>
<td>rem_schema_name</td>
<td>Schema of rem_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes".

Examples

This example creates a view that includes all but one column from the table customers in the current schema.

describe customers
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
CUST_ID                                   NOT NULL NUMBER
CUST_MARITAL_STATUS                                VARCHAR2(20)
OCCUPATION                                         VARCHAR2(21)
AGE                                                NUMBER
YRS_RESIDENCE                                      NUMBER

BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM ('colrem_xtbl');
END;
/
INSERT INTO colrem_xtbl VALUES('CUST_MARITAL_STATUS', null);

NOTE: This currently doesn't work. See bug 9310319
BEGIN
    DBMS_DATA_MINING_TRANSFORM.XFORM_COL_REM (
        rem_table_name        => 'colrem_xtbl',
        data_table_name       => 'customers',
        xform_view_name       => 'colrem_view');
END;
/

describe colrem_view

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>OCCUPATION</td>
<td></td>
<td>VARCHAR2(21)</td>
</tr>
<tr>
<td>AGE</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>YRS_RESIDENCE</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

**44.6.35 XFORM_EXPR_NUM Procedure**

This procedure creates a view that implements the specified numeric transformations. Only the columns that you specify are transformed; the remaining columns from the data table are present in the view, but they are not changed.

**Syntax**

```
DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_NUM (
    expr_pattern       IN       VARCHAR2,
    data_table_name    IN       VARCHAR2,
    xform_view_name    IN       VARCHAR2,
    exclude_list       IN       COLUMN_LIST DEFAULT NULL,
    include_list       IN       COLUMN_LIST DEFAULT NULL,
    col_pattern        IN       VARCHAR2 DEFAULT ':col',
    data_schema_name   IN       VARCHAR2 DEFAULT NULL,
    xform_schema_name  IN       VARCHAR2 DEFAULT NULL);
```

**Parameters**

**Table 44-45  XFORM_EXPR_NUM Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr_pattern</td>
<td>A numeric transformation expression</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in expr_pattern and col_pattern.</td>
</tr>
<tr>
<td>exclude_list</td>
<td>List of numerical columns to exclude. If NULL, no numerical columns are excluded.</td>
</tr>
</tbody>
</table>

**The format of exclude_list is:**

```
dbms_data_mininig_transform.COLUMN_LIST('col1','col2', ...
'coln')
```
### Table 44-45 (Cont.) XFORM_EXPR_NUM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_list</td>
<td>List of numeric columns to include. If NULL, all numeric columns are included. The format of <code>include_list</code> is: <code>dbms_data_mining_transform.COLUMN_LIST('col1','col2','...','coln')</code></td>
</tr>
<tr>
<td>col_pattern</td>
<td>The value within <code>expr_pattern</code> that will be replaced with a column name. The value of <code>col_pattern</code> is case-sensitive. The default value of <code>col_pattern</code> is <code>:col</code></td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of <code>data_table_name</code>. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of <code>xform_view_name</code>. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

**Usage Notes**

1. The `XFORM_EXPR_NUM` procedure constructs numeric transformation expressions from the specified expression pattern (`expr_pattern`) by replacing every occurrence of the specified column pattern (`col_pattern`) with an actual column name.

   `XFORM_EXPR_NUM` uses the SQL `REPLACE` function to construct the transformation expressions.

   ```sql
   REPLACE (expr_pattern, col_pattern, '"column_name") || '"column_name"
   ```

   If there is a column match, then the replacement is made in the transformation expression; if there is not a match, then the column is used without transformation.

   **See:**

   *Oracle Database SQL Language Reference* for information about the `REPLACE` function

2. Because of the include and exclude list parameters, the `XFORM_EXPR_NUM` and `XFORM_EXPR_STR` procedures allow you to easily specify individual columns for transformation within large data sets. The other `XFORM_*` procedures support an exclude list only. In these procedures, you must enumerate every column that you do not want to transform.

3. See "Operational Notes"

**Examples**

This example creates a view that transforms the datatype of numeric columns.

```
describe customers
Name                                Null?    Type
----------------------------------- -------- ------------------------
```
### 44.6.36 XFORM_EXPR_STR Procedure

This procedure creates a view that implements the specified categorical transformations. Only the columns that you specify are transformed; the remaining columns from the data table are present in the view, but they are not changed.

**Syntax**

```sql
DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_STR (
   expr_pattern IN VARCHAR2,
   data_table_name IN VARCHAR2,
   xform_view_name IN VARCHAR2,
   exclude_list IN COLUMN_LIST DEFAULT NULL,
   include_list IN COLUMN_LIST DEFAULT NULL,
   col_pattern IN VARCHAR2 DEFAULT ':col',
   data_schema_name IN VARCHAR2 DEFAULT NULL,
   xform_schema_name IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

**Table 44-46 XFORM_EXPR_STR Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr_pattern</td>
<td>A character transformation expression</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in expr_pattern and col_pattern.</td>
</tr>
</tbody>
</table>
Table 44-6  (Cont.) XFORM_EXPR_STR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclude_list</td>
<td>List of categorical columns to exclude. If NULL, no categorical columns are excluded. The format of exclude_list is: <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', 'coln')</code></td>
</tr>
<tr>
<td>include_list</td>
<td>List of character columns to include. If NULL, all character columns are included. The format of include_list is: <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', 'coln')</code></td>
</tr>
<tr>
<td>col_pattern</td>
<td>The value within expr_pattern that will be replaced with a column name. The value of col_pattern is case-sensitive. The default value of col_pattern is ':col'</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

1. The XFORM_EXPR_STR procedure constructs character transformation expressions from the specified expression pattern (expr_pattern) by replacing every occurrence of the specified column pattern (col_pattern) with an actual column name.

   XFORM_EXPR_STR uses the SQL REPLACE function to construct the transformation expressions.

   `REPLACE (expr_pattern, col_pattern, ""column_name"") || ""column_name"'

   If there is a column match, then the replacement is made in the transformation expression; if there is not a match, then the column is used without transformation.

   See: Oracle Database SQL Language Reference for information about the REPLACE function

2. Because of the include and exclude list parameters, the XFORM_EXPR_STR and XFORM_EXPR_NUM procedures allow you to easily specify individual columns for transformation within large data sets. The other XFORM_* procedures support an exclude list only. In these procedures, you must enumerate every column that you do not want to transform.

3. See "Operational Notes"
Examples

This example creates a view that transforms character columns to upper case.

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_STR(  
    expr_pattern  => 'upper(:col)',  
    data_table_name => 'customers',  
    xform_view_name => 'cust_upcase_view');
END;
/
```

```
describe cust_upcase_view
Name                          Null?    Type
----------------------------- -------- -------------------
CUST_ID                       NOT NULL NUMBER
CUST_MARITAL_STATUS                    VARCHAR2(20)
OCCUPATION                             VARCHAR2(21)
AGE                                    NUMBER
YRS_RESIDENCE                          NUMBER
```

```
SELECT cust_id,  cust_marital_status, occupation FROM cust_upcase_view 
  WHERE cust_id > 102995
ORDER BY cust_id desc;
```

```
CUST_ID CUST_MARITAL_STATUS  OCCUPATION
------- -------------------- ---------------------
103000 DIVORC.              CLERIC.
102999 MARRIED              CLERIC.
102998 MARRIED              EXEC.
102997 MARRIED              EXEC.
102996 NEVERM               OTHER
```

44.6.37 XFORM_MISS_CAT Procedure

This procedure creates a view that implements the categorical missing value treatment transformations specified in a definition table. Only the columns that are specified in the
definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT (  
    miss_table_name       IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    xform_view_name       IN VARCHAR2,
    miss_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2 DEFAULT NULL,
    xform_schema_name     IN VARCHAR2 DEFAULT NULL;
)
```

Parameters

**Table 44-47  XFORM_MISS_CAT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table for categorical missing value treatment. You can use the CREATE_MISS_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call XFORM_MISS_CAT. To populate the table, you can use the INSERT_MISS_CAT_MODE Procedure or you can write your own SQL. See Table 44-12.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in miss_table_name.</td>
</tr>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes".

Examples

This example creates a view that replaces missing categorical values with the mode.

```sql
SELECT * FROM geog;
```

<table>
<thead>
<tr>
<th>REG_ID</th>
<th>REGION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NE</td>
</tr>
<tr>
<td>2</td>
<td>SW</td>
</tr>
<tr>
<td>3</td>
<td>SE</td>
</tr>
<tr>
<td>4</td>
<td>SW</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>NE</td>
</tr>
<tr>
<td>7</td>
<td>NW</td>
</tr>
<tr>
<td>8</td>
<td>NW</td>
</tr>
</tbody>
</table>
SELECT STATS_MODE(region) FROM geog;

STATS_MODE(Region)
-------------------
SE

BEGIN
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT('misscat_xtbl');
DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE(
   miss_table_name => 'misscat_xtbl',
   data_table_name => 'geog');
END;
/

SELECT col, val FROM misscat_xtbl;

<table>
<thead>
<tr>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGION</td>
<td>SE</td>
</tr>
</tbody>
</table>

BEGIN
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT(
   miss_table_name => 'misscat_xtbl',
   data_table_name => 'geog',
   xform_view_name => 'geogxf_view');
END;
/

SELECT * FROM geogxf_view;

<table>
<thead>
<tr>
<th>REG_ID</th>
<th>REGION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NE</td>
</tr>
<tr>
<td>1</td>
<td>SW</td>
</tr>
<tr>
<td>2</td>
<td>SE</td>
</tr>
<tr>
<td>3</td>
<td>NW</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>NE</td>
</tr>
<tr>
<td>7</td>
<td>NW</td>
</tr>
<tr>
<td>8</td>
<td>NW</td>
</tr>
<tr>
<td>9</td>
<td>SE</td>
</tr>
<tr>
<td>10</td>
<td>SE</td>
</tr>
<tr>
<td>11</td>
<td>SE</td>
</tr>
<tr>
<td>12</td>
<td>NW</td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>SE</td>
</tr>
<tr>
<td>15</td>
<td>SE</td>
</tr>
</tbody>
</table>
44.6.38 XFORM_MISS_NUM Procedure

This procedure creates a view that implements the numerical missing value treatment transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM (
    miss_table_name IN VARCHAR2,
    data_table_name IN VARCHAR2,
    xform_view_name IN VARCHAR2,
    miss_schema_name IN VARCHAR2 DEFAULT NULL,
    data_schema_name IN VARCHAR2 DEFAULT NULL,
    xform_schema_name IN VARCHAR2 DEFAULT NULL;
)
```

Parameters

Table 44-48  XFORM_MISS_NUM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss_table_name</td>
<td>Name of the transformation definition table for numerical missing value treatment. You can use the CREATE_MISS_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call XFORM_MISS_NUM. To populate the table, you can use the INSERT_MISS_NUM_MEAN Procedure or you can write your own SQL. See Table 44-14.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in miss_table_name.</td>
</tr>
<tr>
<td>miss_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes".

Examples

This example creates a view that replaces missing numerical values with the mean.

```sql
SELECT * FROM items;
```

<table>
<thead>
<tr>
<th>ITEM_ID</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>200</td>
</tr>
<tr>
<td>bb</td>
<td>200</td>
</tr>
</tbody>
</table>
SELECT AVG(qty) FROM items;

AVG(QTY)
--------
  200

BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM('missnum_xtbl');
    DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_NUM_MEAN (
        miss_table_name => 'missnum_xtbl',
        data_table_name => 'items',
    );
END;
/

SELECT col, val FROM missnum_xtbl;

COL  VAL
----- ----
QTY  200

BEGIN
    DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM (
        miss_table_name => 'missnum_xtbl',
        data_table_name => 'items',
        xform_view_name => 'items_view');
END;
/

SELECT * FROM items_view;

ITEM_ID   QTY
-------- ----
aa        200
bb        200
cc        250
dd        200
ee        200
ff        100
gg        250
hh        200
ii        200
jj        200

44.6.39 XFORM_NORM_LIN Procedure

This procedure creates a view that implements the linear normalization transformations specified in a definition table. Only the columns that are specified in the definition table are
transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```sql
DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN (
    norm_table_name      IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    xform_view_name      IN VARCHAR2,
    norm_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL,
    xform_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 44-49  XFORM_NORM_LIN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm_table_name</td>
<td>Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The table must be populated with transformation definitions before you call XFORM_NORM_LIN. To populate the table, you can use one of the INSERT procedures for normalization or you can write your own SQL. See Table 44-12.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view presents columns in data_table_name with the transformations specified in miss_table_name.</td>
</tr>
<tr>
<td>norm_schema_name</td>
<td>Schema of miss_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes".

Examples

This example creates a view that normalizes the cust_year_of_birth and cust_credit_limit columns. The data source consists of three columns from sh.customer.

```sql
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_credit_limit
    FROM sh.customers;
```

describe mining_data

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>NOT NULL</td>
<td>NUMBER(4)</td>
</tr>
</tbody>
</table>
CUST_CREDIT_LIMIT          NUMBER

SELECT * FROM mining_data WHERE cust_id > 104495
ORDER BY cust_year_of_birth;

CUST_ID  CUST_YEAR_OF_BIRTH  CUST_CREDIT_LIMIT
---------  ------------------  -----------------
104496    1947               3000
104498    1954               10000
104500    1962               15000
104499    1970               3000
104497    1976               3000

BEGIN
  dbms_data_mining_transform.CREATE_NORM_LIN(
    norm_table_name => 'normx_tbl');
  dbms_data_mining_transform.INSERT_NORM_LIN_MINMAX(
    norm_table_name => 'normx_tbl',
    data_table_name => 'mining_data',
    exclude_list    => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
    round_num       => 3);
END;
/

SELECT col, shift, scale FROM normx_tbl;

<table>
<thead>
<tr>
<th>COL</th>
<th>SHIFT</th>
<th>SCALE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1910</td>
<td>77</td>
</tr>
<tr>
<td>CUST_CREDIT_LIMIT</td>
<td>1500</td>
<td>13500</td>
</tr>
</tbody>
</table>

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN(
    norm_table_name => 'normx_tbl',
    data_table_name => 'mining_data',
    xform_view_name => 'norm_view');
END;
/

SELECT * FROM norm_view WHERE cust_id > 104495
ORDER BY cust_year_of_birth;

CUST_ID  CUST_YEAR_OF_BIRTH  CUST_CREDIT_LIMIT
---------  ------------------  -----------------
104496    .4605195           .1111111
104498    .5714286           .6296296
104500    .6753247           1
104499    .7792208           .1111111
104497    .8571429           .1111111

set long 2000
SQL> SELECT text FROM user_views WHERE view_name IN 'NORM_VIEW';

TEXT

---------------------------------------------------------------------------------
SELECT "CUST_ID",("CUST_YEAR_OF_BIRTH"-1910)/77 "CUST_YEAR_OF_BIRTH",("CUST_CREDIT_LIMIT"-1500)/13500 "CUST_CREDIT_LIMIT" FROM mining_data
44.6.40 XFORM_STACK Procedure

This procedure creates a view that implements the transformations specified by the stack. Only the columns and nested attributes that are specified in the stack are transformed. Any remaining columns and nested attributes from the data table appear in the view without changes.

To create a list of objects that describe the transformed columns, use the DESCRIBE_STACK Procedure.

See Also:

"Overview"

Oracle Data Mining User's Guide for more information about data mining attributes

Syntax

DBMS_DATA_MINING_TRANSFORM.XFORM_STACK (  
    xform_list IN TRANSFORM_list,  
    data_table_name IN VARCHAR2,  
    xform_view_name IN VARCHAR2,  
    data_schema_name IN VARCHAR2 DEFAULT NULL,  
    xform_schema_name IN VARCHAR2 DEFAULT NULL);

Parameters

Table 44-50  XFORM_STACK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xform_list</td>
<td>The transformation list. See Table 44-1 for a description of the TRANSFORM_LIST object type.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be transformed</td>
</tr>
<tr>
<td>xform_view_name</td>
<td>Name of the view to be created. The view applies the transformations in xform_list to data_table_name.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Schema of data_table_name. If no schema is specified, the current schema is used.</td>
</tr>
<tr>
<td>xform_schema_name</td>
<td>Schema of xform_view_name. If no schema is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"
Examples

This example applies a transformation list to the view dmuser.cust_info and shows how the data is transformed. The `CREATE` statement for cust_info is shown in "DESCRIBE_STACK Procedure".

BEGIN
    dbms_data_mining_transform.CREATE_BIN_NUM ('birth_yr_bins');
    dbms_data_mining_transform.INSERT_BIN_NUM_QTILE (
        bin_table_name => 'birth_yr_bins',
        data_table_name => 'cust_info',
        bin_num => 6,
        exclude_list => dbms_data_mining_transform.column_list('cust_id','country_id'));
END;
/
SELECT * FROM birth_yr_bins;

<table>
<thead>
<tr>
<th>COL</th>
<th>ATT</th>
<th>VAL BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1922</td>
<td></td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1951</td>
<td>1</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1959</td>
<td>2</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1966</td>
<td>3</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1973</td>
<td>4</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1979</td>
<td>5</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>1986</td>
<td>6</td>
</tr>
</tbody>
</table>

DECLARE
    cust_stack   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
        'country_id', NULL, 'country_id/10', 'country_id*10');
    dbms_data_mining_transform.STACK_BIN_NUM ('birth_yr_bins',
        cust_stack);
    dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
        'custprods', 'Mouse Pad', 'value*100', 'value/100');
    dbms_data_mining_transform.XFORM_STACK(
        xform_list => cust_stack,
        data_table_name => 'cust_info',
        xform_view_name => 'cust_xform_view');
END;
/
-- Two rows of data without transformations
SELECT * from cust_info WHERE cust_id BETWEEN 100010 AND 100011;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>COUNTRY_ID</th>
<th>CUST_YEAR_OF_BIRTH</th>
<th>CUSTPRODS(ATTRIBUTE_NAME, VALUE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100010</td>
<td>52790</td>
<td>1975</td>
<td>DM_NESTED_NUMERICALS(</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERICAL(</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'18&quot; Flat Panel Graphics Monitor', 1),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERICAL(</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'SIMM- 16MB PCMCIAI card', 1))</td>
</tr>
<tr>
<td>100011</td>
<td>52775</td>
<td>1972</td>
<td>DM_NESTED_NUMERICALS(</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERICAL(</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'External 8X CD-ROM', 1),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERICAL(</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'Mouse Pad', 1),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERICAL(</td>
</tr>
</tbody>
</table>
```
SIMM- 16MB PCMCIAII card', 1),
DM_NESTED_NUMERIC(  
  'Keyboard Wrist Rest', 1),
DM_NESTED_NUMERIC(  
  '18" Flat Panel Graphics Monitor', 1),
DM_NESTED_NUMERIC(  
  'O/S Documentation Set - English', 1))

-- Same two rows of data with transformations
SELECT * FROM cust_xform_view WHERE cust_id BETWEEN 100010 AND 100011;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>COUNTRY_ID</th>
<th>C</th>
<th>CUSTPRODS(ATTRIBUTE_NAME, VALUE)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100010</td>
<td>5279</td>
<td>5</td>
<td>DM_NESTED_NUMERICS(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'18&quot; Flat Panel Graphics Monitor', 1),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'SIMM- 16MB PCMCIAII card', 1))</td>
<td></td>
</tr>
<tr>
<td>100011</td>
<td>5277.5</td>
<td>4</td>
<td>DM_NESTED_NUMERICS(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'External 8X CD-ROM', 1),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'Mouse Pad', 100),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'SIMM- 16MB PCMCIAII card', 1),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'Keyboard Wrist Rest', 1),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'18&quot; Flat Panel Graphics Monitor', 1),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DM_NESTED_NUMERIC(</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'O/S Documentation Set - English', 1))</td>
<td></td>
</tr>
</tbody>
</table>
```
The DBMS_DATAPUMP package is used to move all, or part of, a database between databases, including both data and metadata.

See Also:
Oracle Database Utilities for more information on the concepts behind the DBMS_DATAPUMP API, how it works, and how it is implemented in the Data Pump Export and Import utilities.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Data Structures
- Summary of DBMS_DATAPUMP Subprograms

45.1 DBMS_DATAPUMP Overview

DBMS_DATAPUMP provides the following support and functionality.

- The source and target databases can have different hardware, operating systems, character sets, time zones, and versions.
- All object types and datatypes existing in Oracle Database 11g and higher are supported.
- Data and metadata can be transferred between databases without using any intermediary files.
- A subset of a database can be moved based upon object type and names of objects.
- Schema names, data file names, tablespace names, and data can be transformed at import time.
- Previously aborted export and import jobs can be restarted without duplicating or omitting any data or metadata from the original job.
- The resources applied to an export or import job can be modified.
- Data in an Oracle proprietary format can be unloaded and loaded.
45.2 DBMS_DATAPUMP Security Model

Security for the DBMS_DATAPUMP package is implemented through roles.

The DATAPUMP_EXP_FULL_DATABASE and DATAPUMP_IMP_FULL_DATABASE roles allow privileged users to take full advantage of the API. The Data Pump API will use these roles to determine whether privileged application roles should be assigned to the processes comprising the job.

- **DATAPUMP_EXP_FULL_DATABASE**
  The DATAPUMP_EXP_FULL_DATABASE role affects only Export operations. It allows users running these operations to do the following:
  - Perform the operation outside of the scope of their schema
  - Monitor jobs that were initiated by another user
  - Export objects (for example, TABLESPACE definitions) that unprivileged users cannot reference

  Although the SYS schema does not have the DATAPUMP_EXP_FULL_DATABASE role assigned to it, all security checks performed by Data Pump that require the DATAPUMP_EXP_FULL_DATABASE role will also grant access to the SYS schema.

- **DATAPUMP_IMP_FULL_DATABASE**
  The DATAPUMP_IMP_FULL_DATABASE role affects only Import and SQL_FILE operations. It allows users running these operations to do the following:
  - Perform the operation outside of the scope of their schema
  - Monitor jobs that were initiated by another user
  - Import objects (for example, DIRECTORY definitions) that unprivileged users cannot create

  Although the SYS schema does not have the DATAPUMP_IMP_FULL_DATABASE role assigned to it, all security checks performed by Data Pump that require the DATAPUMP_IMP_FULL_DATABASE role will also grant access to the SYS schema.

45.3 DBMS_DATAPUMP Constants

There are several public constants defined for use with the DBMS_DATAPUMP.GET_STATUS procedure. All such constants are defined as part of the DBMS_DATAPUMP package. Any references to these constants must be prefixed by DBMS_DATAPUMP, and followed by the symbols in the following lists:

**Mask Bit Definitions**

The following mask bit definitions are used for controlling the return of data through the DBMS_DATAPUMP.GET_STATUS procedure.

- **KU$_STATUS_WIP** CONSTANT BINARY_INTEGER := 1;
- **KU$_STATUS_JOB_DESC** CONSTANT BINARY_INTEGER := 2;
- **KU$_STATUS_JOB_STATUS** CONSTANT BINARY_INTEGER := 4;
- **KU$_STATUS_JOB_ERROR** CONSTANT BINARY_INTEGER := 8;
Dump File Type Definitions

The following definitions are used for identifying types of dump files returned through the DBMS_DATAPUMP.GET_STATUS procedure.

- KU$_DUMPFILE_TYPE_DISK CONSTANT BINARY_INTEGER := 0;
- KU$_DUMPFILE_TYPE_TEMPLATE CONSTANT BINARY_INTEGER := 3;

45.4 DBMS_DATAPUMP Data Structures

The DBMS_DATAPUMP package defines OBJECT types. The types described in this section are defined in the SYS schema for use by the GET_STATUS function. The way in which these types are defined and used may be different than what you are accustomed to. Be sure to read this section carefully.

The collection of types defined for use with the GET_STATUS procedure are version-specific and include version information in the names of the types. Once introduced, these types will always be provided and supported in future versions of Oracle Database and will not change. However, in future releases of Oracle Database, new versions of these types might be created that provide new or different information. The new versions of these types will have different version information embedded in the type names.

For example, in Oracle Database 12c, Release 1 (12.1), there is a sys.ku$_Status1210 type, and in the next Oracle Database release, there could be a sys.ku$_Status1310 type defined. Both types could be used with the GET_STATUS procedure.

Public synonyms have been defined for each of the types used with the GET_STATUS procedure. This makes it easier to use the types and means that you do not have to be concerned with changes to the actual type names or schemas where they reside. Oracle recommends that you use these synonyms whenever possible.

For each of the types, there is a version-specific synonym and a generic synonym. For example, the version-specific synonym ku$_Status1210 is defined for the sys.ku$_Status1210 type.

The generic synonym always describes the latest version of that type. For example, in Oracle Database 12c, Release 1 (12.1), the generic synonym ku$_Status is defined as ku$_Status1210. In a future release, there might be a ku$_Status1310 synonym for sys.ku$Status1310. Because the ku$_Status generic synonym always points to the latest definition, it would then point to ku$_Status1310 rather than to ku$_Status1210.

The choice of whether to use version-specific synonyms or generic synonyms makes a significant difference in how you work. Using version-specific names protects your code from changes in future releases of Oracle Database because those types will continue to exist and be supported. However, access to new information will require code changes to use new synonym names for each of the types. Using the generic names implies that you always want the latest definition of the types and are prepared to deal with changes in different releases of Oracle Database.

When the version of Oracle Database that you are using changes, any C code that accesses types through generic synonym names will need to be recompiled.
Data Structures — Object Types

The `DBMS_DATAPUMP` package defines the following kinds of `OBJECT` types:

- Worker Status Types
- Log Entry and Error Types
- Job Status Types
- Job Description Types
- Status Types

Worker Status Types

The worker status types describe what each worker process in a job is doing. The schema, object name, and object type of an object being processed will be provided. For workers processing user data, the partition name for a partitioned table (if any), the number of bytes processed in the partition, and the number of rows processed in the partition are also returned. Workers processing metadata provide status on the last object that was processed. No status for idle threads is returned.

The `percent_done` refers to the amount completed for the current data item being processed. It is not updated for metadata objects.

The worker status types are defined as follows:

```plaintext
CREATE TYPE sys.ku$WorkerStatus1010 AS OBJECT
{
    worker_number NUMBER,       -- Worker process identifier
    process_name  VARCHAR2(30), -- Worker process name
    state         VARCHAR2(30), -- Worker process state
    schema        VARCHAR2(30), -- Schema name
    name          VARCHAR2(4000),-- Object name
    object_type   VARCHAR2(200),-- Object type
    partition     VARCHAR2(30), -- Partition name
    completed_objects NUMBER,   -- Completed number of objects
    total_objects  NUMBER,       -- Total number of objects
    completed_rows NUMBER,       -- Number of rows completed
    completed_bytes NUMBER,      -- Number of bytes completed
    percent_done   NUMBER         -- Percent done current object
}

CREATE OR REPLACE PUBLIC SYNONYM ku$WorkerStatus1010
FOR sys.ku$WorkerStatus1010;
```
CREATE TYPE sys.ku$_WorkerStatus1020 AS OBJECT
(
    worker_number NUMBER, -- Worker process identifier
    process_name VARCHAR2(30), -- Worker process name
    state VARCHAR2(30), -- Worker process state
    schema VARCHAR2(30), -- Schema name
    name VARCHAR2(4000), -- Object name
    object_type VARCHAR2(200), -- Object type
    partition VARCHAR2(30), -- Partition name
    completed_objects NUMBER, -- Completed number of objects
    total_objects NUMBER, -- Total number of objects
    completed_rows NUMBER, -- Number of rows completed
    completed_bytes NUMBER, -- Number of bytes completed
    percent_done NUMBER, -- Percent done current object
    degree NUMBER, -- Degree of parallelism
)

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1020 FOR sys.ku$_WorkerStatus1020;

CREATE TYPE sys.ku$_WorkerStatus1120 AS OBJECT
(
    worker_number NUMBER, -- Worker process identifier
    process_name VARCHAR2(30), -- Worker process name
    state VARCHAR2(30), -- Worker process state
    schema VARCHAR2(30), -- Schema name
    name VARCHAR2(4000), -- Object name
    object_type VARCHAR2(200), -- Object type
    partition VARCHAR2(30), -- Partition name
    completed_objects NUMBER, -- Completed number of objects
    total_objects NUMBER, -- Total number of objects
    completed_rows NUMBER, -- Number of rows completed
    completed_bytes NUMBER, -- Number of bytes completed
    percent_done NUMBER, -- Percent done current object
    degree NUMBER, -- Degree of parallelism
    instance_id NUMBER, -- Instance ID where running
    instance_name VARCHAR2(60), -- Instance Name where running
    host_name VARCHAR2(64) -- Host name where running
)

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1120 FOR sys.ku$_WorkerStatus1120;

CREATE TYPE sys.ku$_WorkerStatus1210 AS OBJECT
(
    worker_number NUMBER, -- Worker process identifier
    process_name VARCHAR2(30), -- Worker process name
    state VARCHAR2(30), -- Worker process state
    schema VARCHAR2(30), -- Schema name
    name VARCHAR2(4000), -- Object name
    object_type VARCHAR2(200), -- Object type
    partition VARCHAR2(30), -- Partition name
    completed_objects NUMBER, -- Completed number of objects
    total_objects NUMBER, -- Total number of objects
    completed_rows NUMBER, -- Number of rows completed
    completed_bytes NUMBER, -- Number of bytes completed
    percent_done NUMBER, -- Percent done current object
    degree NUMBER, -- Degree of parallelism
    instance_id NUMBER, -- Instance ID where running
    instance_name VARCHAR2(60), -- Instance Name where running
    host_name VARCHAR2(64) -- Host name where running
)
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1210
FOR sys.ku$_WorkerStatus1210;

CREATE TYPE sys.ku$_WorkerStatus1220 AS OBJECT
{
    worker_number NUMBER, -- Worker process identifier
    process_name VARCHAR2(128), -- Worker process name
    state VARCHAR2(30), -- Worker process state
    schema VARCHAR2(128), -- Schema name
    name VARCHAR2(4000), -- Object name
    object_type VARCHAR2(200), -- Object type
    partition VARCHAR2(128), -- Partition name
    completed_objects NUMBER, -- Completed number of objects
    total_objects NUMBER, -- Total number of objects
    completed_rows NUMBER, -- Number of rows completed
    completed_bytes NUMBER, -- Number of bytes completed
    percent_done NUMBER, -- Percent done current object
    degree NUMBER, -- Degree of parallelism
    instance_id NUMBER, -- Instance ID where running
    instance_name VARCHAR2(60), -- Instance Name where running
    host_name VARCHAR2(101), -- Host name where running
    access_method VARCHAR2(16), -- Access Method of object
    obj_start_time DATE, -- Object start time
    obj_status DATE -- Object status at current time
}

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1220
FOR sys.ku$_WorkerStatus1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus FOR ku$_WorkerStatus1220;

CREATE TYPE sys.ku$_WorkerStatusList1010 AS TABLE OF sys.ku$_WorkerStatus1010
CREATE TYPE sys.ku$_WorkerStatusList1020 AS TABLE OF sys.ku$_WorkerStatus1020
CREATE TYPE sys.ku$_WorkerStatusList1120 AS TABLE OF sys.ku$_WorkerStatus1120
CREATE TYPE sys.ku$_WorkerStatusList1210 AS TABLE OF sys.ku$_WorkerStatus1210
CREATE TYPE sys.ku$_WorkerStatusList1220 AS TABLE OF sys.ku$_WorkerStatus1220

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1010
FOR sys.ku$_WorkerStatusList1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1020
FOR sys.ku$_WorkerStatusList1020;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1120
FOR sys.ku$_WorkerStatusList1120;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1210
FOR sys.ku$_WorkerStatusList1210;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1220
FOR sys.ku$_WorkerStatusList1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList
FOR ku$_WorkerStatusList1220;

Log Entry and Error Types

These types provide informational and error text to attached clients and the log stream. The ku$LogLine.errorNumber type is set to NULL for informational messages but is specified for error messages. Each log entry may contain several lines of text messages.

The log entry and error types are defined as follows:
CREATE TYPE sys.ku$ _LogLine1010 AS OBJECT {
  logLineNumber NUMBER,
  errorNumber NUMBER,
  LogText VARCHAR2(2000)}

CREATE OR REPLACE PUBLIC SYNONYM ku$ _LogLine1010 FOR sys.ku$ _LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$ _LogLine1020 FOR sys.ku$ _LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$ _LogLine FOR ku$ _LogLine1010;
CREATE TYPE sys.ku$_LogEntry1010 AS TABLE OF sys.ku$_LogLine1010

CREATE OR REPLACE PUBLIC SYNONYM ku$ _LogEntry1010 FOR sys.ku$ _LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$ _LogEntry1020 FOR sys.ku$ _LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$ _LogEntry FOR ku$ _LogEntry1010;

Job Status Types

The job status type returns status about a job. Usually, the status concerns a running job, but it could also be about a stopped job when a client attaches. It is typically requested at attach time, when the client explicitly requests status from interactive mode and every N seconds when the client has requested status periodically.

The job status types are defined as follows (percent_done applies to data only):

CREATE TYPE sys.ku$ _JobStatus1010 IS OBJECT {
  job_name VARCHAR2(30), -- Name of the job
  operation VARCHAR2(30), -- Current operation
  job_mode VARCHAR2(30), -- Current mode
  bytes_processed NUMBER, -- Bytes so far
  total_bytes NUMBER, -- Total bytes for job
  percent_done NUMBER, -- Percent done
  degree NUMBER, -- Of job parallelism
  error_count NUMBER, -- #errors so far
  state VARCHAR2(30), -- Current job state
  phase NUMBER, -- Job phase
  restart_count NUMBER, -- #Job restarts
  worker_status_list ku$ _WorkerStatusList1010, -- For (non-idle)
                  -- job worker processes
  files ku$ _DumpFileSet1010 -- Dump file info
}

CREATE OR REPLACE PUBLIC SYNONYM ku$ _JobStatus1010 FOR sys.ku$ _JobStatus1010;

CREATE TYPE sys.ku$ _JobStatus1020 IS OBJECT {
  job_name VARCHAR2(30), -- Name of the job
  operation VARCHAR2(30), -- Current operation
  job_mode VARCHAR2(30), -- Current mode
  bytes_processed NUMBER, -- Bytes so far
  total_bytes NUMBER, -- Total bytes for job
  percent_done NUMBER, -- Percent done
  degree NUMBER, -- Of job parallelism
  error_count NUMBER, -- #errors so far
  state VARCHAR2(30), -- Current job state
  phase NUMBER, -- Job phase
  restart_count NUMBER, -- #Job restarts
  worker_status_list ku$ _WorkerStatusList1020, -- For (non-idle)
                  -- job worker processes
  files ku$ _DumpFileSet1010 -- Dump file info
}
CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1020 FOR sys.ku$_JobStatus1020;

CREATE TYPE sys.ku$_JobStatus1120 IS OBJECT
{
    job_name VARCHAR2(30),            -- Name of the job
    operation VARCHAR2(30),           -- Current operation
    job_mode VARCHAR2(30),            -- Current mode
    bytes_processed NUMBER,            -- Bytes so far
    total_bytes NUMBER,                -- Total bytes for job
    percent_done NUMBER,               -- Percent done
    degree NUMBER,                     -- Of job parallelism
    error_count NUMBER,                -- #errors so far
    state VARCHAR2(30),                -- Current job state
    phase NUMBER,                      -- Job phase
    restart_count NUMBER,              -- #Job restarts
    worker_status_list ku$_WorkerStatusList1120, -- For (non-idle)
    -- job worker processes
    files ku$_DumpFileSet1010           -- Dump file info
}

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1120 FOR sys.ku$_JobStatus1120;

CREATE TYPE sys.ku$_JobStatus1210 IS OBJECT
{
    job_name VARCHAR2(30),            -- Name of the job
    operation VARCHAR2(30),           -- Current operation
    job_mode VARCHAR2(30),            -- Current mode
    bytes_processed NUMBER,            -- Bytes so far
    total_bytes NUMBER,                -- Total bytes for job
    percent_done NUMBER,               -- Percent done
    degree NUMBER,                     -- Of job parallelism
    error_count NUMBER,                -- #errors so far
    state VARCHAR2(30),                -- Current job state
    phase NUMBER,                      -- Job phase
    restart_count NUMBER,              -- #Job restarts
    worker_status_list ku$_WorkerStatusList1210, -- For (non-idle)
    -- job worker processes
    files ku$_DumpFileSet1010           -- Dump file info
}

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1210 FOR sys.ku$_JobStatus1210;

CREATE TYPE sys.ku$_JobStatus1220 IS OBJECT
{
    job_name VARCHAR2(128),            -- Name of the job
    operation VARCHAR2(30),           -- Current operation
    job_mode VARCHAR2(30),            -- Current mode
    bytes_processed NUMBER,            -- Bytes so far
    total_bytes NUMBER,                -- Total bytes for job
    percent_done NUMBER,               -- Percent done
    degree NUMBER,                     -- Of job parallelism
    error_count NUMBER,                -- #errors so far
    state VARCHAR2(30),                -- Current job state
    phase NUMBER,                      -- Job phase
    restart_count NUMBER,              -- #Job restarts
    heartbeat NUMBER,                  -- Job heartbeat
    worker_status_list ku$_WorkerStatusList1220, -- For (non-idle)
    -- job worker processes
    files ku$_DumpFileSet1010           -- Dump file info
}
CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1220 FOR sys.ku$_JobStatus1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus FOR ku$_JobStatus1220;

**Job Description Types**

The job description type holds all the environmental information about the job such as parameter settings and dump file set members. There are a couple of subordinate types required as well.

The job description types are defined as follows:

```
CREATE TYPE sys.ku$_JobDesc1010 IS OBJECT
{
  job_name        VARCHAR2(30),           -- The job name
  guid            RAW(16),                -- The job GUID
  operation       VARCHAR2(30),           -- Current operation
  job_mode        VARCHAR2(30),           -- Current mode
  remote_link     VARCHAR2(4000),         -- DB link, if any
  owner           VARCHAR2(30),           -- Job owner
  instance        VARCHAR2(16),           -- The instance name
  db_version      VARCHAR2(30),           -- Version of objects
  creator_privs   VARCHAR2(30),           -- Privs of job
  start_time      DATE,                   -- This job start time
  max_degree      NUMBER,                 -- Max. parallelism
  log_file        VARCHAR2(4000),         -- Log file name
  sql_file        VARCHAR2(4000),         -- SQL file name
  params          ku$_ParamValues1010     -- Parameter list
}
```

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1010 FOR sys.ku$_JobDesc1010;

CREATE TYPE sys.ku$_JobDesc1020 IS OBJECT
{
  job_name        VARCHAR2(30),           -- The job name
  guid            RAW(16),                -- The job GUID
  operation       VARCHAR2(30),           -- Current operation
  job_mode        VARCHAR2(30),           -- Current mode
  remote_link     VARCHAR2(4000),         -- DB link, if any
  owner           VARCHAR2(30),           -- Job owner
  platform        VARCHAR2(101),          -- Current job platform
  exp_platform    VARCHAR2(101),          -- Export platform
  global_name     VARCHAR2(4000),         -- Current global name
  exp_global_name VARCHAR2(4000),         -- Export global name
  instance        VARCHAR2(16),           -- The instance name
  db_version      VARCHAR2(30),           -- Version of objects
  exp_db_version  VARCHAR2(30),           -- Export version
  scn             NUMBER,                 -- Job SCN
  creator_privs   VARCHAR2(30),           -- Privs of job
  start_time      DATE,                   -- This job start time
  exp_start_time  DATE,                   -- Export start time
  term_reason     NUMBER,                 -- Job termination code
  max_degree      NUMBER,                 -- Max. parallelism
  log_file        VARCHAR2(4000),         -- Log file name
  sql_file        VARCHAR2(4000),         -- SQL file name
  params          ku$_ParamValues1010     -- Parameter list
}
```
CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1020 FOR sys.ku$_JobDesc1020;

CREATE TYPE sys.ku$_JobDesc1210 IS OBJECT
{
  job_name VARCHAR2(30), -- The job name
  guid RAW(16), -- The job GUID
  operation VARCHAR2(30), -- Current operation
  job_mode VARCHAR2(30), -- Current mode
  remote_link VARCHAR2(4000), -- DB link, if any
  owner VARCHAR2(30), -- Job owner
  platform VARCHAR2(101), -- Current job platform
  exp_platform VARCHAR2(101), -- Export platform
  global_name VARCHAR2(4000), -- Current global name
  exp_global_name VARCHAR2(4000), -- Export global name
  instance VARCHAR2(16), -- The instance name
  db_version VARCHAR2(30), -- Cur. server software
  version exp_db_version VARCHAR2(30), -- Export svr. software
  version job_version VARCHAR2(30), -- Negotiated data
  version scn NUMBER, -- Job SCN
  creator_privs VARCHAR2(30), -- Privs of job
  start_time DATE, -- This job start time
  exp_start_time DATE, -- Export start time
  term_reason NUMBER, -- Job termination code
  max_degree NUMBER, -- Max. parallelism
  timezone VARCHAR2(64), -- Cur. server timezone
  exp_timezone VARCHAR2(64), -- Exp. server timezone
  tstz_version NUMBER, -- Cur. server timezone
  version exp_tstz_version NUMBER, -- Exp. server timezone
  endianness VARCHAR2(16), -- Cur. platform's
  version endianness VARCHAR2(16), -- Exp. platform's
  version -- endianness is 'BIG' or 'LITTLE'
  charset VARCHAR2(28), -- Cur. server charset
  exp_charset VARCHAR2(28), -- Exp. server charset
  ncharset VARCHAR2(28), -- Cur. server national
  exp_ncharset VARCHAR2(28), -- Exp. server national
  charset log_file VARCHAR2(4000), -- Log file name
  sql_file VARCHAR2(4000), -- SQL file name
  params ku$_ParamValues1010 -- Parameter list
}

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1210 FOR sys.ku$_JobDesc1210;

CREATE TYPE sys.ku$_JobDesc1220 IS OBJECT
{
  job_name VARCHAR2(128), -- The job name
  guid RAW(16), -- The job GUID
  operation VARCHAR2(30), -- Current operation
  job_mode VARCHAR2(30), -- Current mode
  remote_link VARCHAR2(4000), -- DB link, if any
  owner VARCHAR2(30), -- Job owner
  platform VARCHAR2(101), -- Current job platform
  exp_platform VARCHAR2(101), -- Export platform
  version
Status Types

The status type is an aggregate of some the previous types defined and is the return value for the GET_STATUS call. The mask attribute indicates which types of information are being returned to the caller. It is created by a client's shadow process from information it retrieves off the status queue or directly from the master table.

For errors, the ku$_LogEntry that is returned has already had its log lines ordered for proper output. That is, the original ku$_LogEntry objects have been ordered from outermost context to innermost.

The status types are defined as follows:

CREATE TYPE sys.ku$_Status1010 IS OBJECT
{
  mask            NUMBER,           -- Status types present
  wip             ku$_LogEntry1010, -- Work in progress
  job_description ku$_JobDesc1010, -- Complete job description
  job_status      ku$_JobStatus1010,-- Detailed job status
  error           ku$_LogEntry1010 -- Multi-level context errors
}

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1010 FOR sys.ku$_Status1010;

CREATE TYPE sys.ku$_Status1020 IS OBJECT
{
  mask            NUMBER,           -- Status types present
  wip             ku$_LogEntry1010, -- Work in progress
  job_description ku$_JobDesc1020, -- Complete job description
  job_status      ku$_JobStatus1020,-- Detailed job status
  error           ku$_LogEntry1010 -- Multi-level context errors
}

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1020 FOR sys.ku$_Status1020;
45.5 Summary of DBMS_DATAPUMP Subprograms

This table lists the DBMS_DATAPUMP subprograms in alphabetical order and briefly describes them.

Table 45-1   DBMS_DATAPUMP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_FILE Procedure</td>
<td>Adds dump files to the dump file set for an Export, Import, or SQL_FILE operation. In addition to dump files, other types of files can also be added by using the FILETYPE parameter provided with this procedure.</td>
</tr>
<tr>
<td>ATTACH Function</td>
<td>Used to gain access to a Data Pump job that is in the Defining, Executing, Idling, or Stopped state.</td>
</tr>
<tr>
<td>DATA_FILTER Procedures</td>
<td>Specifies restrictions on the rows that are to be retrieved.</td>
</tr>
</tbody>
</table>

---

Chapter 45
Summary of DBMS_DATAPUMP Subprograms
### Table 45-1 (Cont.) DBMS_DATAPUMP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_REMAP Procedure</td>
<td>Specifies transformations to be applied to column data as it is exported from, or imported into, a database.</td>
</tr>
<tr>
<td>DETACH Procedure</td>
<td>Specifies that the user has no further interest in using the handle</td>
</tr>
<tr>
<td>GET_DUMPFILE_INFO Procedure</td>
<td>Retrieves information about a specified dump file</td>
</tr>
<tr>
<td>GET_STATUS Procedure</td>
<td>Monitors the status of a job or waits for the completion of a job or for more details on API errors</td>
</tr>
<tr>
<td>LOG_ENTRY Procedure</td>
<td>Inserts a message into the log file</td>
</tr>
<tr>
<td>METADATA_FILTER Procedure</td>
<td>Provides filters that allow you to restrict the items that are included in a job</td>
</tr>
<tr>
<td>METADATA_REMAP Procedure</td>
<td>Specifies a remapping to be applied to objects as they are processed in the specified job</td>
</tr>
<tr>
<td>METADATA_TRANSFORM Procedure</td>
<td>Specifies transformations to be applied to objects as they are processed in the specified job</td>
</tr>
<tr>
<td>OPEN Function</td>
<td>Declares a new job using the Data Pump API, the handle returned being used as a parameter for calls to all other procedures (but not to the ATTACH function)</td>
</tr>
<tr>
<td>SET_PARALLEL Procedure</td>
<td>Adjusts the degree of parallelism within a job</td>
</tr>
<tr>
<td>SET_PARAMETER Procedures</td>
<td>Specifies job-processing options</td>
</tr>
<tr>
<td>START_JOB Procedure</td>
<td>Begins or resumes execution of a job</td>
</tr>
<tr>
<td>STOP_JOB Procedure</td>
<td>Terminates a job, but optionally, preserves the state of the job</td>
</tr>
<tr>
<td>WAIT_FOR_JOB Procedure</td>
<td>Runs a job until it either completes normally or stops for some other reason</td>
</tr>
</tbody>
</table>

### 45.5.1 ADD_FILE Procedure

This procedure adds files to the dump file set for an Export, Import, or SQL_FILE operation or specifies the log file or the output file for a SQL_FILE operation.

**Syntax**

```sql
DBMS_DATAPUMP.ADD_FILE (  
    handle    IN NUMBER,  
    filename  IN VARCHAR2,  
    directory IN VARCHAR2,  
    filesize  IN VARCHAR2 DEFAULT NULL,  
    filetype  IN NUMBER DEFAULT DBMS_DATAPUMP.KU$_FILE_TYPE_DUMP_FILE,  
    reusefile IN NUMBER DEFAULT NULL);
```
Parameters

Table 45-2  ADD_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.</td>
</tr>
<tr>
<td>filename</td>
<td>The name of the file being added. filename must be a simple filename without any directory path information. For dump files, the filename can include a substitution variable. See the following table for a description of available substitution variables.</td>
</tr>
<tr>
<td>directory</td>
<td>The name of a directory object within the database that is used to locate filename. A directory must be specified.</td>
</tr>
<tr>
<td>filesize</td>
<td>The size of the dump file that is being added. It may be specified as the number of bytes, number of kilobytes (if followed by K), number of megabytes (if followed by M), number of gigabytes (if followed by G) or number of terabytes (if followed by T). An Export operation will write no more than the specified number of bytes to the file. Once the file is full, it will be closed. If there is insufficient space on the device to write the specified number of bytes, the Export operation will fail, but it can be restarted. If not specified, filesize will default to an unlimited size. For Import and SQL_FILE operations, filesize is ignored. The minimum value for filesize is ten times the default Data Pump block size, which is 4 kilobytes. A filesize can only be specified for dump files.</td>
</tr>
<tr>
<td>filetype</td>
<td>The type of the file to be added. The legal values are as follows and must be preceded by DBMS_DATAPUMP.:</td>
</tr>
<tr>
<td></td>
<td>- KU$_FILE_TYPE_DUMP_FILE (dump file for a job)</td>
</tr>
<tr>
<td></td>
<td>- KU$_FILE_TYPE_LOG_FILE (log file for a job)</td>
</tr>
<tr>
<td></td>
<td>- KU$_FILE_TYPE_SQL_FILE (output for SQL_FILE job)</td>
</tr>
<tr>
<td>reusefile</td>
<td>If 0, a preexisting file will cause an error. If 1, a preexisting file will be overwritten. If NULL, the default action for the file type will be applied (that is, dump files will not be overwritten). This parameter should only be non-NULL for dump files. The reusefile parameter is restricted to export jobs.</td>
</tr>
</tbody>
</table>

Substitution Variables

Table 45-3  Substitution Variables Available for the Filename Parameter on DBMS_DATAPUMP.ADD_FILE

<table>
<thead>
<tr>
<th>Substitution Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%U</td>
<td>The %U is expanded in the resulting file names into a two-character, fixed-width, incrementing integer starting at 01. For example, the dump filename of export%U would cause export01, export02, export03, and so on, to be created depending on how many files are needed to perform the export. For filenames containing the % character, the % must be represented as %% to avoid ambiguity.</td>
</tr>
<tr>
<td>%d, %D</td>
<td>Specifies the current day of the month from the Gregorian calendar in format DD. Note: This substitution variable cannot be used in an import file name.</td>
</tr>
</tbody>
</table>
Table 45-3  (Cont.) Substitution Variables Available for the Filename Parameter on DBMS_DATAPUMP.ADD_FILE

<table>
<thead>
<tr>
<th>Substitution Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%m, %M</td>
<td>Specifies the month in the Gregorian calendar in format MM. Note: This substitution variable cannot be used in an import file name.</td>
</tr>
<tr>
<td>%t, %T</td>
<td>Specifies the year, month, and day in the Gregorian calendar in this format: YYYYMMDD. Note: This substitution variable cannot be used in an import file name.</td>
</tr>
<tr>
<td>%l, %L</td>
<td>Specifies a system-generated unique file name. The file names can contain a substitution variable (%L), which implies that multiple files may be generated. The substitution variable is expanded in the resulting file names into a 2-digit, fixed-width, incrementing integer starting at 01 and ending at 99 which is the same as (%U). In addition, the substitution variable is expanded in the resulting file names into a 3-digit to 10-digit, variable-width, incrementing integers starting at 100 and ending at 2147483646. The width field is determined by the number of digits in the integer. For example if the current integer was 1, exp%Uaa%U.dmp would resolve to exp01aa01.dmp exp02aa02.dmp and so forth up until 99. Then, the next file name would have 3 digits substituted: exp100aa100.dmp exp101aa101.dmp and so forth up until 999 where the next file name would have 4 digits substituted. The substitution will continue up to the largest number substitution allowed, which is 2147483646.</td>
</tr>
<tr>
<td>%y, %Y</td>
<td>Specifies the year in this format: YYYY. Note: This substitution variable cannot be used in an import file name.</td>
</tr>
</tbody>
</table>

Exceptions

- **INVALID_HANDLE.** The specified handle is not attached to a Data Pump job.
- **INVALID_ARGVAL.** An invalid value was supplied for an input parameter.
- **INVALID_STATE.** The job is completing, or the job is past the defining state for an import or SQL_FILE job or is past the defining state for LOG and SQL files.
- **INVALID_OPERATION.** A dump file was specified for a Network Import or ESTIMATE_ONLY export operation.
- **SUCCESS_WITH_INFO.** The procedure succeeded, but further information is available through the GET_STATUS procedure.
- **NO_SUCH_JOB.** The specified job does not exist.
Usage Notes

- Adds files to a Data Pump job. Three types of files may be added to jobs: Dump files to contain the data that is being moved, log files to record the messages associated with an operation, and SQL files to record the output of a SQL_FILE operation. Log and SQL files will overwrite previously existing files. Dump files will never overwrite previously existing files. Instead, an error will be generated.

- Import and SQL_FILE operations require that all dump files be specified during the definition phase of the job. For Export operations, dump files can be added at any time. For example, if the user ascertains that the file space is running low during an Export, additional dump files may be added through this API. If the specified dump file already exists for an Export operation and reusefile is not set to 1, an error will be returned.

- For Export operations, the parallelism setting should be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, the job will not be able to maximize parallelism to the degree specified by the SET_PARALLEL procedure.

- For Import operations, the parallelism setting should also be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, the performance will not be optimal as multiple threads of execution try to access the same dump file.

- If the substitution variable (\%U) is included in a filename, multiple dump files may be specified through a single call to ADD_FILE. For Export operations, the new dump files will be created as they are needed. Enough dump files will be created to allow all of the processes specified by the current SET_PARALLEL value to be active. If one of the dump files fills, it will be closed and a new dump file (with a new generated name) will be created to take its place. If multiple ADD_FILES with substitution variables have been specified for dump files in a job, they will be used to generate dump files in a round robin fashion. For example, if expa\%U, expb\%U and expc\%U were all specified for a job having a parallelism of 6, the initial dump files created would look like: expa01, expb01, expc01, expa02, expb02, and expc02.

- If presented with dump file specifications, expa\%U, expb\%U and expc\%U, an Import or SQL_FILE operation will begin by attempting to open the dump files, expa01, expb01, and expc01. If the dump file containing the master table is not found in this set, the operation will expand its search for dump files by incrementing the substitution variable and looking up the new filenames (for example, expa02, expb02, and expc02). The DataPump API will keep expanding the search until it locates the dump file containing the master table. If the DataPump API determines that the dump file does not exist or is not part of the current dump set at any iteration, the DataPump API will stop incrementing the substitution variable for the dump file specification that was in error. Once the master table is found, the master table will be used to ascertain when all of dump files in the dump file set have been located.
45.5.2 ATTACH Function

This function provides access to a previously created job.

Syntax

```sql
DBMS_DATAPUMP.ATTACH(
    job_name    IN VARCHAR2 DEFAULT NULL,
    job_owner   IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of the job. The default is the job name owned by the user who is specified in the job_owner parameter (assuming that user has only one job in the Defining, Executing, or Idling states).</td>
</tr>
<tr>
<td>job_owner</td>
<td>The user who originally started the job. If NULL, then the value defaults to the owner of the current session. To specify a job owner other than yourself, you must have either the DATAPUMP_EXP_FULL_DATABASE role (for export operations) or the DATAPUMP_IMP_FULL_DATABASE role (for import and SQL_FILE operations). Being a privileged user allows you to monitor another user's job, but you cannot restart another user's job.</td>
</tr>
</tbody>
</table>

Return Values

An opaque handle for the job. This handle is used as input to the following procedures: ADD_FILE, DATA_FILTER, DETACH, GET_STATUS, LOG_ENTRY, METADATA_FILTER, METADATA_REMAP, METADATA_TRANSFORM, SET_PARALLEL, SET_PARAMETER, START_JOB, STOP_JOB, and WAIT_FOR_JOB.

Exceptions

- INVALID_ARGVAL. An invalid value was supplied for an input parameter.
- OBJECT_NOT_FOUND. The specified job no longer exists or the user specified a job owned by another schema, but the user did not have the DATAPUMP_EXP_FULL_DATABASE or DATAPUMP_IMP_FULL_DATABASE role.
- SUCCESS_WITH_INFO. The function succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- If the job was in the Stopped state, then the job is placed into the Idling state. After the ATTACH succeeds, you can monitor the progress of the job or control the job. The stream of KU$_STATUS_WIP and KU$_STATUS_JOB_ERROR messages returned through the GET_STATUS procedure will be returned to the newly attached job starting at the approximate time of the client's attachment. There will be no repeating of status and error messages that were processed before the client attached to a job.
• If you want to perform a second attach to a job, then you must do so from a different session.
• If the ATTACH fails, then use a null handle in a subsequent call to GET_STATUS for more information about the failure.

45.5.3 DATA_FILTER Procedures

This procedure specifies restrictions on the rows that are to be retrieved.

Syntax

```sql
DBMS_DATAPUMP.DATA_FILTER (  
    handle IN NUMBER,  
    name IN VARCHAR2,  
    value IN NUMBER,  
    table_name IN VARCHAR2 DEFAULT NULL,  
    schema_name IN VARCHAR2 DEFAULT NULL);

DBMS_DATAPUMP.DATA_FILTER(  
    handle IN NUMBER,  
    name IN VARCHAR2,  
    value IN VARCHAR2,  
    table_name IN VARCHAR2 DEFAULT NULL,  
    schema_name IN VARCHAR2 DEFAULT NULL);

DBMS_DATAPUMP.DATA_FILTER(  
    handle IN NUMBER,  
    name IN VARCHAR2,  
    value IN CLOB,  
    table_name IN VARCHAR2 DEFAULT NULL,  
    schema_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle that is returned from the OPEN function</td>
</tr>
<tr>
<td>name</td>
<td>The name of the filter</td>
</tr>
<tr>
<td>value</td>
<td>The value of the filter</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table on which the data filter is applied. If no table name is supplied, the filter applies to all tables in the job.</td>
</tr>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the table on which the filter is applied. If no schema name is specified, the filter applies to all schemas in the job. If you supply a schema name you must also supply a table name.</td>
</tr>
</tbody>
</table>

Exceptions

• INVALID_ARGVAL. There can be several reasons for this message:
  – A bad filter name is specified
  – The mode is TRANSPORTABLE, which does not support data filters
  – The specified table does not exist
The filter has already been set for the specified values of `schema_name` and `table_name`.

- **INVALID_STATE.** The user called `DATA_FILTER` when the job was not in the Defining state.
- **INCONSISTENT_ARGS.** The `value` parameter is missing or its datatype does not match the filter name. Or a schema name was supplied, but not a table name.
- **PRIVILEGE_ERROR.** A schema name was supplied, but the user did not have the `DATAPUMP_EXP_FULL_DATABASE` or `DATAPUMP_IMP_FULL_DATABASE` role.
- **SUCCESS_WITH_INFO.** The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- **NO_SUCH_JOB.** The specified job does not exist.

**Usage Notes**

- Each data filter can only appear once in each table (for example, you cannot supply multiple `SUBQUERY` filters to a table) or once in each job. If different filters using the same name are applied to both a particular table and to the whole job, the filter parameter supplied for the specific table will take precedence.

  With the exception of the `INCLUDE_ROWS` filter, data filters are not supported on tables having nested tables or domain indexes defined upon them. Data filters are not supported in jobs performed in Transportable Tablespace mode.

  The available data filters are described in Table 45-6.

**Table 45-6 Data Filters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Operations that Support Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDE_ROWS</td>
<td>NUMBER</td>
<td>EXPORT, IMPORT</td>
<td>If nonzero, this filter specifies that user data for the specified table should be included in the job. The default is 1.</td>
</tr>
</tbody>
</table>
Table 45-6  (Cont.) Data Filters

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Operations that Support Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION_EXPR</td>
<td>TEXT</td>
<td>EXPORT, IMPORT</td>
<td>For Export jobs, these filters specify which partitions are unloaded from the database. For Import jobs, they specify which table partitions are loaded into the database. Partition names are included in the job if their names satisfy the specified expression (for PARTITION_EXPR) or are included in the list (for PARTITION_LIST). Whereas the expression version of the filter offers more flexibility, the list version provides for full validation of the partition names. Double quotation marks around partition names are required only if the partition names contain special characters. PARTITION_EXPR is not supported on jobs across a network link. Default=All partitions are processed.</td>
</tr>
<tr>
<td>PARTITION_LIST</td>
<td>TEXT</td>
<td>EXPORT, IMPORT</td>
<td>For Export jobs, specifies a percentage for sampling the data blocks to be moved. This filter allows subsets of large tables to be extracted for testing purposes.</td>
</tr>
<tr>
<td>SAMPLE</td>
<td>NUMBER</td>
<td>EXPORT, IMPORT</td>
<td>For Export jobs, specifies a percentage for sampling the data blocks to be moved. This filter allows subsets of large tables to be extracted for testing purposes.</td>
</tr>
<tr>
<td>SUBQUERY</td>
<td>TEXT</td>
<td>EXPORT, IMPORT</td>
<td>Specifies a subquery that is added to the end of the SELECT statement for the table. If you specify a WHERE clause in the subquery, you can restrict the rows that are selected. Specifying an ORDER BY clause orders the rows dumped in the export which improves performance when migrating from heap-organized tables to index-organized tables.</td>
</tr>
</tbody>
</table>

45.5.4 DATA_REMAP Procedure

This procedure specifies transformations to be applied to column data as it is exported from, or imported into, a database.

**Syntax**

```sql
DBMS_DATAPUMP.DATA_REMAP(
    handle IN NUMBER,
    name IN VARCHAR2,
    table_name IN VARCHAR2,
)```

**Note:**

In this description, the information about partitions also applies to subpartitions.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of the current job. The current session must have previously attached to the handle through a call to an OPEN function.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the remap</td>
</tr>
<tr>
<td>table_name</td>
<td>The table containing the column to be remapped</td>
</tr>
<tr>
<td>column</td>
<td>The name of the column to be remapped</td>
</tr>
<tr>
<td>remap_function</td>
<td>The meaning of remap_function is dependent upon the value of name. See Table 45-8 for a list of possible names.</td>
</tr>
<tr>
<td>schema</td>
<td>The schema containing the column to be remapped. If NULL, the remapping applies to all schemas moved in the job that contain the specified table.</td>
</tr>
</tbody>
</table>

Exceptions

- INVALID_ARGVAL. The mode is transportable (which does not support data modifications) or it has specified that no data to be included in the job. An invalid remap name was supplied.
- INVALID_OPERATION. Data remaps are only supported for Export and Import operations.
- INVALID_STATE. The DATA_REMAP procedure was called after the job started (that is, it was not in the defining state).
- NO_SUCH_JOB. The job handle is no longer valid.

Usage Notes

- The DATA_REMAP procedure is only supported for Export and Import operations. It allows you to manipulate user data being exported or imported. The name of the remap determines the remap operation to be performed.
- For export operations, you might wish to define a data remap to obscure sensitive data such as credit card numbers from a dump file, but leave the remainder of the data so that it can be read. To accomplish this, the remapping should convert each unique source number into a distinct generated number. So that the mapping is consistent across the dump file set, the same function should be called for every column that contains the credit card number.
- For import operations, you might wish to define a data remap to reset the primary key when data is being merged into an existing table that contains colliding primary keys. A single remapping function should be provided for all columns defining or referencing the primary key to ensure that remapping is consistent.
Note:

If the called function uses package state variables, then to ensure that remapping is performed consistently across all tables, the job should be run with a SET_PARALLEL value of 1 and no restart operations should be performed.

The Data Remap functions are listed in Table 45-8.

Table 45-8 Names of Data Remap Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning of remap_function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_FUNCTION</td>
<td>String having the format: [schema.]package.function</td>
<td>The name parameter references a PL/SQL package function which is called to modify the data for the specified column. The function accepts a single parameter, which has the same datatype as the remapped column, and returns a value having the same datatype as the remapped column. Note that the default for the schema is the schema of the user performing the export.</td>
</tr>
</tbody>
</table>

45.5.5 DETACH Procedure

This procedure specifies that the user has no further interest in using the handle.

Syntax

DBMS_DATAPUMP.DETACH(
    handle  IN NUMBER);

Parameters

Table 45-9 DETACH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of the job. The current session must have previously attached to the handle through a call to either an OPEN or ATTACH function.</td>
</tr>
</tbody>
</table>

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
• **NO_SUCH_JOB.** The specified job does not exist.

**Usage Notes**

• Through this call, you specify that you have no further interest in using the handle. Resources associated with a completed job cannot be reclaimed until all users are detached from the job. An implicit detach from a handle is performed when the user’s session is exited or aborted. An implicit detach from a handle is also performed upon the expiration of the timeout associated with a **STOP_JOB** that was applied to the job referenced by the handle. All previously allocated **DBMS_DATAPUMP** handles are released when an instance is restarted.

### 45.5.6 GET_DUMPFILE_INFO Procedure

This procedure retrieves information about a specified dump file.

**Syntax**

```sql
DBMS_DATAPUMP.GET_DUMPFILE_INFO(
    filename    IN VARCHAR2,
    directory   IN VARCHAR2,
    info_table  OUT ku$_dumpfile_info,
    filetype    OUT NUMBER);
```

**Parameters**

**Table 45-10**  GET_DUMPFILE_INFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>A simple filename with no directory path information</td>
</tr>
<tr>
<td>directory</td>
<td>A directory object that specifies where the file can be found</td>
</tr>
<tr>
<td>info_table</td>
<td>A PL/SQL table for storing information about the dump file</td>
</tr>
<tr>
<td>filetype</td>
<td>The type of file (Data Pump dump file, original Export dump file, external tables dump file, or unknown)</td>
</tr>
</tbody>
</table>

**Exceptions**

The **GET_DUMPFILE_INFO** procedure is a utility routine that operates outside the context of any Data Pump job. Exceptions are handled differently for this procedure than for procedures associated in some way with a Data Pump job. A full exception stack should be available directly, without the need to call the **GET_STATUS** procedure to retrieve the detailed information. The exception for this procedure is as follows:

• **NO_DUMPFILE_INFO.** Unable to retrieve dump file information as specified.

**Usage Notes**

You can use the **GET_DUMPFILE_INFO** procedure to request information about a specific file. If the file is not recognized as any type of dump file, then a filetype of 0 (zero) is returned and the dump file info_table remains empty.

A filetype value of 1 indicates a Data Pump dump file. A filetype value of 2 indicates an original Export dump file. A filetype value of 3 indicates an external tables dump file. In all cases, the dump file info_table will be populated with information retrieved from the dump file.
header. Rows of this table consist of item code and value pairs, where the item code indicates the type of information and the value column is a VARCHAR2 containing the actual data (converted to a string in some cases). The table is defined as follows:

```
CREATE TYPE sys.ku$_dumpfile_item IS OBJECT (  
  item_code       NUMBER,           -- Identifies header item  
  value           VARCHAR2(2048)    -- Text string value)/
GRANT EXECUTE ON sys.ku$_dumpfile_item TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_item FOR sys.ku$_dumpfile_item;

CREATE TYPE sys.ku$_dumpfile_info AS TABLE OF sys.ku$_dumpfile_item/
GRANT EXECUTE ON sys.ku$_dumpfile_info TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_info FOR sys.ku$_dumpfile_info;
```

The item codes, which can easily be extended to provide more information as needed, are currently defined as shown in Table 45-11 (prepended with the package name, DBMS_DATAPUMP). Assume the following with regard to these item codes:

- Unless otherwise stated, all item codes may be returned only for Oracle Data Pump and external tables dump files (filetypes 1 and 3).
- Unless otherwise stated, all item codes have been available since Oracle Database 10g Release 2 (10.2).

Table 45-11  Item Codes For the DBMS_DATAPUMP.GET_DUMPFILE_INFO Procedure

<table>
<thead>
<tr>
<th>Item Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KU$_DFHDR_FILE_VERSION</td>
<td>The internal file version of the dump file.</td>
</tr>
<tr>
<td>KU$_DFHDR_MASTER_PRESENT</td>
<td>If the Data Pump master table is present in the dump file, then the value for this item code is 1; otherwise the value is 0. Returned only for filetype 1.</td>
</tr>
<tr>
<td>KU$_DFHDR_GUID</td>
<td>A unique identifier assigned to the Data Pump export job or the external tables unload job that produced the dump file. For a multifile dump set, each file in the set has the same value for this item code.</td>
</tr>
<tr>
<td>KU$_DFHDR_FILE_NUMBER</td>
<td>A numeric identifier assigned to the dump file. Each dump file in a multifile dump set has its own identifier, unique only within the dump set.</td>
</tr>
<tr>
<td>KU$_DFHDR_CHARSET_ID</td>
<td>A numeric code that represents the character set in use at the source system when the dump file was created. Returned for all filetypes.</td>
</tr>
<tr>
<td>KU$_DFHDR_CREATION_DATE</td>
<td>The date and time that the dump file was created.</td>
</tr>
<tr>
<td>KU$_DFHDR_FLAGS</td>
<td>Internal flag values.</td>
</tr>
<tr>
<td>KU$_DFHDR_JOB_NAME</td>
<td>The name assigned to the export job that created the dump file. Returned only for filetype 1.</td>
</tr>
<tr>
<td>KU$_DFHDR_PLATFORM</td>
<td>The operating system name of the source system on which the dump file was created.</td>
</tr>
<tr>
<td>KU$_DFHDR_INSTANCE</td>
<td>The instance name of the source system on which the dump file was created.</td>
</tr>
<tr>
<td>KU$_DFHDR_LANGUAGE</td>
<td>The language name that corresponds to the character set of the source system where the export dump file was created.</td>
</tr>
<tr>
<td>KU$_DFHDR_BLOCKSIZE</td>
<td>The blocksize, in bytes, of the dump file.</td>
</tr>
<tr>
<td>Item Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KU$_DFHDR_DIRPATH</td>
<td>If direct path mode was used when the dump file was created, then the value for this item code is 1, otherwise the value is 0.</td>
</tr>
<tr>
<td></td>
<td>Returned only for filetype 2.</td>
</tr>
<tr>
<td>KU$_DFHDR_METADATA_COMPRESSED</td>
<td>If the system metadata is stored in the dump file in compressed format, then the value for this item code is 1, otherwise the value is 0.</td>
</tr>
<tr>
<td></td>
<td>Returned only for filetype 1.</td>
</tr>
<tr>
<td>KU$_DFHDR_DB_VERSION</td>
<td>The database job version used to create the dump file.</td>
</tr>
<tr>
<td></td>
<td>Returned for all filetypes.</td>
</tr>
<tr>
<td>KU$_DFHDR_MASTER_PIECE_COUNT</td>
<td>The Data Pump master table may be split into multiple pieces and written to multiple dump files in the set, one piece per file. The value returned for this item code indicates the number of dump files that contain pieces of the master table. The value for this item code is only meaningful if the Data Pump master table is present in the dump file, as indicated by the item code KU$_DFHDR_MASTER_PRESENT.</td>
</tr>
<tr>
<td></td>
<td>Returned only for filetype 1.</td>
</tr>
<tr>
<td></td>
<td>Only available since Oracle Database 11g Release 1 (11.1).</td>
</tr>
<tr>
<td>KU$_DFHDR_MASTER_PIECE_NUMBER</td>
<td>The Data Pump master table may be split into multiple pieces and written to multiple dump files in the set, one piece per file. The value returned for this item code indicates which master table piece is contained in the dump file. The value for this item code is only meaningful if the Data Pump master table is present in the dump file, as indicated by the item code KU$_DFHDR_MASTER_PRESENT.</td>
</tr>
<tr>
<td></td>
<td>Returned only for filetype 1.</td>
</tr>
<tr>
<td></td>
<td>Only available since Oracle Database 11g Release 1 (11.1).</td>
</tr>
<tr>
<td>KU$_DFHDR_DATA_COMPRESSED</td>
<td>If the table data is stored in the dump file in compressed format, then the value for this item code is 1, otherwise the value is 0.</td>
</tr>
<tr>
<td></td>
<td>Only available since Oracle Database 11g Release 1 (11.1).</td>
</tr>
<tr>
<td>KU$_DFHDR_METADATA_ENCRYPTED</td>
<td>If the system metadata is stored in the dump file in encrypted format, then the value for this item code is 1, otherwise the value is 0.</td>
</tr>
<tr>
<td></td>
<td>Returned only for filetype 1.</td>
</tr>
<tr>
<td></td>
<td>Only available since Oracle Database 11g Release 1 (11.1).</td>
</tr>
<tr>
<td>KU$_DFHDR_DATA_ENCRYPTED</td>
<td>If the table data is stored in the dump file in encrypted format, then the value for this item code is 1, otherwise the value is 0.</td>
</tr>
<tr>
<td></td>
<td>Only available since Oracle Database 11g Release 1 (11.1).</td>
</tr>
<tr>
<td>KU$_DFHDR_COLUMNS_ENCRYPTED</td>
<td>If encrypted column data is stored in the dump file in encrypted format, then the value for this item code is 1, otherwise the value is 0.</td>
</tr>
<tr>
<td></td>
<td>Returned only for filetype 1.</td>
</tr>
<tr>
<td></td>
<td>Only available since Oracle Database 11g Release 1 (11.1).</td>
</tr>
</tbody>
</table>
Table 45-11 (Cont.) Item Codes For the DBMS_DATAPUMP.GET_DUMPFILE_INFO Procedure

<table>
<thead>
<tr>
<th>Item Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KU$_DFHDR_ENCRYPTION_MODE</td>
<td>The encryption mode indicates whether a user-provided password or the Oracle Encryption Wallet was used to encrypt data written to the dump file. The possible values returned for this item code are:</td>
</tr>
<tr>
<td>• KU$_DFHDR_ENCMODE_NONE</td>
<td>No data was written to the dump file in encrypted format.</td>
</tr>
<tr>
<td>• KU$_DFHDR_ENCMODE_PASSWORD</td>
<td>Data was written to the dump file in encrypted format using a provided password.</td>
</tr>
<tr>
<td>• KU$_DFHDR_ENCMODE_DUAL</td>
<td>Data was written to the dump file in encrypted format using both a provided password as well as an Oracle Encryption Wallet.</td>
</tr>
<tr>
<td>• KU$_DFHDR_ENCMODE_TRANS</td>
<td>Data was written to the dump file in encrypted format transparently using an Oracle Encryption Wallet.</td>
</tr>
<tr>
<td>Only available since Oracle Database 11g Release 1 (11.1).</td>
<td></td>
</tr>
<tr>
<td>KU$_DFHDR_COMPRESSION_ALG</td>
<td>The compression algorithm used when writing system metadata and/or table data to the dump file in compressed format. The possible values returned for this item code are:</td>
</tr>
<tr>
<td>• KU$_DFHDR_CMPALG_NONE</td>
<td>No data was written to the dump file in compressed format.</td>
</tr>
<tr>
<td>• KU$_DFHDR_CMPALG_BASIC</td>
<td>Data was written to the dump file in compressed format using an internal algorithm. This is the default algorithm used since Oracle Database 10g Release 2 (10.2).</td>
</tr>
<tr>
<td>• KU$_DFHDR_CMPALG_LOW</td>
<td>Data was written to the dump file in compressed format using the LOW algorithm.</td>
</tr>
<tr>
<td>• KU$_DFHDR_CMPALG_MEDIUM</td>
<td>Data was written to the dump file in compressed format using the MEDIUM algorithm.</td>
</tr>
<tr>
<td>• KU$_DFHDR_CMPALG_HIGH</td>
<td>Data was written to the dump file in compressed format using the HIGH algorithm.</td>
</tr>
<tr>
<td>Only available since Oracle Database 12c Release 1 (12.1).</td>
<td></td>
</tr>
</tbody>
</table>

45.5.7 GET_STATUS Procedure

This procedure monitors the status of a job or waits for the completion of a job.

Syntax

```sql
DBMS_DATAPUMP.GET_STATUS(
    handle    IN NUMBER,
    mask      IN BINARY_INTEGER,
    timeout   IN NUMBER DEFAULT NULL,
    job_state OUT VARCHAR2,
    status    OUT ku$_Status);
```
Parameters

Table 45-12  GET_STATUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function. A null handle can be used to retrieve error information after OPEN and ATTACH failures.</td>
</tr>
<tr>
<td>mask</td>
<td>A bit mask that indicates which of four types of information to return:</td>
</tr>
<tr>
<td></td>
<td>· KU$_STATUS_WIP</td>
</tr>
<tr>
<td></td>
<td>· KU$_STATUS_JOB_DESC</td>
</tr>
<tr>
<td></td>
<td>· KU$_STATUS_JOB_STATUS</td>
</tr>
<tr>
<td></td>
<td>· KU$_STATUS_JOB_ERROR</td>
</tr>
<tr>
<td></td>
<td>Each status has a numerical value. You can request multiple types of information by adding together different combinations of values.</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum number of seconds to wait before returning to the user. A value of 0 requests an immediate return. A value of -1 requests an infinite wait. If KU$_STATUS_WIP or KU$_STATUS_JOB_ERROR information is requested and becomes available during the timeout period, then the procedure returns before the timeout period is over.</td>
</tr>
<tr>
<td>job_state</td>
<td>Current state of the job. If only the job state is needed, it is much more efficient to use this parameter than to retrieve the full ku$_Status structure.</td>
</tr>
<tr>
<td>status</td>
<td>A ku$_Status is returned. The ku$_Status mask indicates what kind of information is included. This could be none if only KU$_STATUS_WIP or KU$_STATUS_JOB_ERROR information is requested and the timeout period expires.</td>
</tr>
</tbody>
</table>

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_VALUE. The mask or timeout contains an illegal value.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

The GET_STATUS procedure is used to monitor the progress of an ongoing job and to receive error notification. You can request various type of information using the mask parameter. The KU$_STATUS_JOB_DESC and KU$_STATUS_JOB_STATUS values are classified as synchronous information because the information resides in the master table. The KU$_STATUS_WIP and KU$_STATUS_JOB_ERROR values are classified as asynchronous because the messages that embody these types of information can be generated at any time by various layers in the Data Pump architecture.

- If synchronous information only is requested, the interface will ignore the timeout parameter and simply return the requested information.
- If asynchronous information is requested, the interface will wait a maximum of timeout seconds before returning to the client. If a message of the requested asynchronous information type is received, the call will complete prior to timeout seconds. If
synchronous information was also requested, it will be returned whenever the
procedure returns.
• If the \texttt{job\_state} returned by \texttt{GET\_STATUS} does not indicate a terminating job, it is
possible that the job could still terminate before the next call to \texttt{GET\_STATUS}. This
would result in an \texttt{INVALID\_HANDLE} exception. Alternatively, the job could terminate
during the call to \texttt{GET\_STATUS}, which would result in a \texttt{NO\_SUCH\_JOB} exception.
Callers should be prepared to handle these cases.

\section*{Error Handling}

There are two types of error scenarios that need to be handled using the \texttt{GET\_STATUS}
procedure:

• Errors resulting from other procedure calls: For example, the \texttt{SET\_PARAMETER}
procedure may produce an \texttt{INCONSISTENT\_ARGS} exception. The client should
immediately call \texttt{GET\_STATUS} with mask=8 (errors) and timeout=0. The returned
\texttt{ku\_Status.error} will contain a \texttt{ku\_LogEntry} that describes the inconsistency in
more detail.

• Errors resulting from events asynchronous to the client(s): An example might be
\texttt{Table already exists} when trying to create a table. The \texttt{ku\_Status.error} will
contain a \texttt{ku\_LogEntry} with all error lines (from all processing layers that added
context about the error) properly ordered.

After a job has begun, a client's main processing loop will typically consist of a call to
\texttt{GET\_STATUS} with an infinite timeout (-1) "listening" for \texttt{KU\_STATUS\_WIP} and
\texttt{KU\_STATUS\_JOB\_ERROR} messages. If status was requested, then \texttt{JOB\_STATUS}
information will also be in the request.

When the \texttt{ku\_Status} is interpreted, the following guidelines should be used:

• \texttt{ku\_Status.ku\_JobStatus.percent\_done} refers only to the amount of data that
has been processed in a job. Metadata is not considered in the calculation. It is
determined using the following formulas:

  – \texttt{EXPORT} or network \texttt{IMPORT}--\( \frac{\text{bytes\_processed}}{\text{estimated\_bytes}} \times 100 \)
  – \texttt{IMPORT}--\( \frac{\text{bytes\_processed}}{\text{total\_expected\_bytes}} \times 100 \)
  – \texttt{SQL\_FILE} or estimate-only \texttt{EXPORT}--0.00 if not done or 100.00 if done

The effects of the \texttt{QUERY} and \texttt{PARTITION\_EXPR} data filters are not considered in
computing \texttt{percent\_done}.

It is expected that the status returned will be transformed by the caller into more
user-friendly status. For example, when percent done is not zero, an estimate of
completion time could be produced using the following formula:

\[
\left( \frac{\text{SYSDATE} - \text{start time}}{\text{ku\_Status.ku\_JobStatus.percent\_done}} \right) \times 100
\]

• The caller should not use \texttt{ku\_Status.ku\_JobStatus.percent\_done} for
determining whether the job has completed. Instead, the caller should only rely on
the state of the job as found in \texttt{job\_state}.  

\hspace*{ill}
45.5.8 LOG_ENTRY Procedure

This procedure inserts a message into the log file.

Syntax

```java
DBMS_DATAPUMP.LOG_ENTRY(
  handle     IN NUMBER,
  message    IN VARCHAR2,
  log_file_only IN NUMBER DEFAULT 0);
```

Parameters

Table 45-13  LOG_ENTRY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.</td>
</tr>
<tr>
<td>message</td>
<td>A text line to be added to the log file</td>
</tr>
<tr>
<td>log_file_only</td>
<td>Specified text should be written only to the log file. It should not be returned in GET_STATUS work-in-progress (KU$_STATUS_WIP) messages.</td>
</tr>
</tbody>
</table>

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

The message is added to the log file. If log_file_only is zero (the default), the message is also broadcast as a KU$_STATUS_WIP message through the GET_STATUS procedure to all users attached to the job.

The LOG_ENTRY procedure allows applications to tailor the log stream to match the abstractions provided by the application. For example, the command-line interface supports INCLUDE and EXCLUDE parameters defined by the user. Identifying these values as calls to the underlying METADATA_FILTER procedure would be confusing to users. Instead, the command-line interface can enter text into the log describing the settings for the INCLUDE and EXCLUDE parameters.

Lines entered in the log stream from LOG_ENTRY are prefixed by the string ";;; "

45.5.9 METADATA_FILTER Procedure

This procedure provides filters that allow you to restrict the items that are included in a job.

Syntax

```java
DBMS_DATAPUMP.METADATA_FILTER(
  handle     IN NUMBER,
```


Parameters

Table 45-14  METADATA_FILTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from the OPEN function</td>
</tr>
<tr>
<td>name</td>
<td>The name of the filter. See Table 45-15 for descriptions of the available filters.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the filter</td>
</tr>
<tr>
<td>object_path</td>
<td>The object path to which the filter applies. If the default is used, the filter applies to all applicable objects. Lists of the object paths supported for each mode are contained in the catalog views for DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, and TABLE_EXPORT_OBJECTS. (Note that the TABLE_EXPORT_OBJECTS view is applicable to both Table and Tablespace mode because their object paths are the same.) For an import operation, object paths reference the mode used to create the dump file rather than the mode being used for the import.</td>
</tr>
</tbody>
</table>

Table 45-15 describes the name, the object type, and the meaning of the filters available with the METADATA_FILTER procedure. The datatype for all the filters is a text expression. All operations support all filters.

Table 45-15  Filters Provided by METADATA_FILTER Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Object Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME_EXPR</td>
<td>Named objects</td>
<td>Defines which object names are included in the job. You use the object type parameter to limit the filter to a particular object type. For Table mode, identifies which tables are to be processed.</td>
</tr>
<tr>
<td>NAME_LIST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCHEMA_EXPR</td>
<td>Schema objects</td>
<td>Restricts the job to objects whose owning schema name is satisfied by the expression. For Table mode, only a single SCHEMA_EXPR filter is supported. If specified, it must only specify a single schema (for example, 'IN (''SCOTT'')'). For Schema mode, identifies which users are to be processed.</td>
</tr>
<tr>
<td>SCHEMA_LIST</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Table 45-15** (Cont.) Filters Provided by METADATA_FILTER Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Object Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLESPACE_E</td>
<td>TABLE,</td>
<td>Restricts the job to objects stored in a tablespace whose name is satisfied by the expression.</td>
</tr>
<tr>
<td>XPR</td>
<td>CLUSTER,INDEX, ROLLBACK_SEGMENT</td>
<td>For Tablespace mode, identifies which tablespaces are to be processed. If a partition of an object is stored in the tablespace, the entire object is added to the job. For Transportable mode, identifies which tablespaces are to be processed. If a table has a single partition in the tablespace set, all partitions must be in the tablespace set. An index is not included within the tablespace set unless all of its partitions are in the tablespace set. A domain index is not included in the tablespace set unless all of its secondary objects are included in the tablespace set.</td>
</tr>
<tr>
<td>TABLESPACE_LIST</td>
<td>TABLE,</td>
<td>Defines which object paths are included in, or excluded from, the job. You use these filters to select only certain object types from the database or dump file set. Objects of paths satisfying the condition are included (INCLUDE_PATH_<em>) or excluded (EXCLUDE_PATH_</em>) from the operation. The object_path parameter is not supported for these filters.</td>
</tr>
<tr>
<td>INCLUDE_PATH_EXPR</td>
<td>All</td>
<td>Specifies that no tables are to be exported.</td>
</tr>
<tr>
<td>INCLUDE_PATH_LIST</td>
<td>All</td>
<td>A comma-separated list of views to be exported as tables: [schema_name.]view_name[table_name] The filter can be called multiple times with multiple values and all values get added to a list. All views on the list are exported as tables.</td>
</tr>
<tr>
<td>EXCLUDE_PATH_EXPR</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>EXCLUDE_PATH_LIST</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>EXCLUDE_TABLES</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>VIEWS_AS_TABLES</td>
<td>All</td>
<td></td>
</tr>
</tbody>
</table>

**Exceptions**

- **INVALID_HANDLE.** The specified handle is not attached to a Data Pump job.
- **INVALID_ARGVAL.** This exception can indicate any of the following conditions:
  - An object_path was specified for an INCLUDE_PATH_EXPR or EXCLUDE_PATH_EXPR filter.
  - The specified object_path is not supported for the current mode.
  - The SCHEMA_EXPR filter specified multiple schemas for a Table mode job.
- **INVALID_STATE.** The user called the METADATA_FILTER procedure after the job left the defining state.
- **INCONSISTENT_ARGS.** The filter value is of the wrong datatype or is missing.
- **SUCCESS_WITH_INFO.** The procedure succeeded but further information is available through the GET_STATUS procedure.
- **NO_SUCH_JOB.** The specified job does not exist.
Usage Notes

- Metadata filters identify a set of objects to be included or excluded from a Data Pump operation. Except for `EXCLUDE_PATH_EXPR` and `INCLUDE_PATH_EXPR`, dependent objects of an identified object will be processed along with the identified object. For example, if an index is identified for inclusion by a filter, grants upon that index will also be included by the filter. Likewise, if a table is excluded by a filter, then indexes, constraints, grants and triggers upon the table will also be excluded by the filter.

- Two versions of each filter are supported: SQL expression and List. The SQL expression version of the filters offer maximum flexibility for identifying objects (for example the use of LIKE to support use of wild cards). The names of the expression filters are as follows:
  - `NAME_EXPR`
  - `SCHEMA_EXPR`
  - `TABLESPACE_EXPR`
  - `INCLUDE_PATH_EXPR`
  - `EXCLUDE_PATH_EXPR`

The list version of the filters allow maximum validation of the filter. An error will be reported if one of the elements in the filter is not found within the source database (for Export and network-based jobs) or is not found within the dump file (for file-based Import and SQLFILE jobs). The names of the list filters are as follows:
  - `NAME_LIST`
  - `SCHEMA_LIST`
  - `TABLESPACE_LIST`
  - `INCLUDE_PATH_LIST`
  - `EXCLUDE_PATH_LIST`

- Filters allow a user to restrict the items that are included in a job. For example, a user could request a full export, but without Package Specifications or Package Bodies.

- If multiple filters are specified for a object type, they are implicitly 'ANDed' together (that is, objects participating in the job must pass all of the filters applied to their object types).

- The same filter name can be specified multiple times within a job. For example, specifying `NAME_EXPR` as `!="EMP"` and `NAME_EXPR` as `!="DEPT"` on a Table mode export would produce a file set containing all of the tables except for EMP and DEPT.
45.5.10 METADATA_REMAP Procedure

This procedure specifies a remapping to be applied to objects as they are processed in the specified job.

Syntax

```sql
DBMS_DATAPUMP.METADATA_REMAP (
    handle      IN NUMBER,
    name        IN VARCHAR2,
    old_value   IN VARCHAR2,
    value       IN VARCHAR2,
    object_type IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 45-16  METADATA_REMAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle for the current job. The current session must have previously attached to the handle through a call to the OPEN function.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the remap. See Table 45-17 for descriptions of the available remaps.</td>
</tr>
<tr>
<td>old_value</td>
<td>Specifies which value in the dump file set should be reset to value</td>
</tr>
<tr>
<td>value</td>
<td>The value of the parameter for the remap. This signifies the new value that old_value should be translated into.</td>
</tr>
<tr>
<td>object_type</td>
<td>Designates the object type to which the remap applies. The list of object types supported for each mode are contained in the DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, TABLE_EXPORT_OBJECTS, and TABLESPACE_EXPORT_OBJECTS catalog views. By default, the remap applies to all applicable objects within the job. The object_type parameter allows a caller to specify different parameters for different object types within a job. Remaps that explicitly specify an object type override remaps that apply to all object types.</td>
</tr>
</tbody>
</table>

Table 45-17 describes the remaps provided by the METADATA_REMAP procedure.
### Table 45-17 Remaps Provided by the METADATA_REMAP Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Object Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMAP_SCHEMA</td>
<td>TEXT</td>
<td>Schema objects</td>
<td>Any schema object in the job that matches the object_type parameter and was located in the old_value schema will be moved to the value schema. Privileged users can perform unrestricted schema remaps. Nonprivileged users can perform schema remaps only if their schema is the target schema of the remap. For example, SCOTT can remap his BLAKE's objects to SCOTT, but SCOTT cannot remap SCOTT's objects to BLAKE.</td>
</tr>
<tr>
<td>REMAP_TABLESPACE</td>
<td>TEXT</td>
<td>TABLE, INDEX, ROLLBACK_SEGMENT, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG, TABLESPACE</td>
<td>Any storage segment in the job that matches the object_type parameter and was located in the old_value tablespace will be relocated to the value tablespace.</td>
</tr>
<tr>
<td>REMAP_DATAFILE</td>
<td>TEXT</td>
<td>LIBRARY, TABLESPACE, DIRECTORY</td>
<td>If old_value and value are both full file specifications, then any data file reference in the job that matches the object_type parameter and that referenced the old_value data file will be redefined to use the value data file. If old_value and value are both directory paths, then any data file reference whose object path matches old_value will have its path substituted with value.</td>
</tr>
<tr>
<td>REMAP_TABLE</td>
<td>TEXT</td>
<td>TABLE</td>
<td>Any reference to a table in the job that matches the old_value table name will be replaced with the value table name. The old_value parameter may refer to a partition such as employees.low. This allows names for tables constructed the by PARTITION_OPTIONS=DEPARTITION parameter to be specified by the user.</td>
</tr>
</tbody>
</table>
Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_ARGVAL. This message can indicate any of the following:
  - The job's mode does not include the specified object_type.
  - The remap has already been specified for the specified old_value and object_type.
- INVALID_OPERATION. Remaps are only supported for SQL_FILE and Import operations. The job's operation was Export, which does not support the use of metadata remaps.
- INVALID_STATE. The user called METADATA_REMAP after the job had started (that is, the job was not in the defining state).
- INCONSISTENT_ARGS. There was no value supplied or it was of the wrong datatype for the remap.
- PRIVILEGE_ERROR. A nonprivileged user attempted to do a REMAP_SCHEMA to a different user's schema or a REMAP_DATAFILE.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- The METADATA_REMAP procedure is only supported for Import and SQL_FILE operations. It enables you to apply commonly desired, predefined remappings to the definition of objects as part of the transfer. If you need remaps that are not supported within this procedure, you should do a preliminary SQL_FILE operation to produce a SQL script corresponding to the dump file set. By editing the DDL directly and then executing it, you can produce any remappings that you need.
- Transforms for the DataPump API are a subset of the remaps implemented by the DBMS_METADATA_SET_TRANSFORM_PARAMETER API. Multiple remaps can be defined for a single job. However, each remap defined must be unique according its parameters. That is, two remaps cannot specify conflicting or redundant remaps.

45.5.11 METADATA_TRANSFORM Procedure

This procedure specifies transformations to be applied to objects as they are processed in the specified job.

Syntax

```sql
DBMS_DATAPUMP.METADATA_TRANSFORM (  
    handle IN NUMBER,  
    name IN VARCHAR2,  
    value IN VARCHAR2,  
    object_type IN VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_DATAPUMP.METADATA_TRANSFORM (  
    handle IN NUMBER,  
    name IN VARCHAR2,  
    value IN NUMBER,  
    object_type IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 45-18 METADATA_TRANSFORM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle for the current job. The current session must have previously attached to the handle through a call to the OPEN function.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the transformation. See Table 45-19 for descriptions of the available transforms.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the parameter for the transform</td>
</tr>
<tr>
<td>object_type</td>
<td>Designates the object type to which the transform applies. The list of object types supported for each mode are contained in the DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, TABLE_EXPORT_OBJECTS, and TABLESPACE_EXPORT_OBJECTS catalog views. By default, the transform applies to all applicable objects within the job. The object_type parameter allows a caller to specify different transform parameters for different object types within a job. Transforms that explicitly specify an object type override transforms that apply to all object types.</td>
</tr>
</tbody>
</table>

Table 45-19 describes the transforms provided by the METADATA_TRANSFORM procedure.

Table 45-19 Transforms Provided by the METADATA_TRANSFORM Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Object Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE_ARCHIVE_LOGGING</td>
<td>NUMBER</td>
<td>TABLE</td>
<td>Specifies whether to disable archive logging for specified object types during import. A value of zero (FALSE) is the default. It specifies that archive logging will take place. This is the default behavior if this transform parameter is not specified. A non-zero (TRUE) value disables archive logging for the specified object types before data is loaded during import. If no object type is specified, then archive logging is disabled for both TABLE and INDEX object types. All other object types processed during data pump import are logged. <strong>Note:</strong> If the database is in FORCE LOGGING mode, then the DISABLE_ARCHIVE_LOGGING transform does not disable logging when indexes and tables are created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INDEX</td>
<td></td>
</tr>
</tbody>
</table>
### Table 45-19 (Cont.) Transforms Provided by the METADATA_TRANSFORM Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Object Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INMEMORY</td>
<td>NUMBER</td>
<td>TABLE</td>
<td>The <strong>INMEMORY</strong> transform is related to the In-Memory Column Store (IM column store), an optional area in the SGA that stores whole tables, table partitions, and individual columns in a compressed columnar format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABLESPACE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MATERIALIZED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VIEWS</td>
<td></td>
</tr>
<tr>
<td>INMEMORY_CLA</td>
<td>TEXT</td>
<td>TABLE</td>
<td>The <strong>INMEMORY_CLA</strong> USE transform is related to the In-Memory Column Store (IM column store), an optional area in the SGA that stores whole tables, table partitions, and individual columns in a compressed columnar format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABLESPACE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MATERIALIZED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VIEWS</td>
<td></td>
</tr>
<tr>
<td>LOB_STORAGE</td>
<td>TEXT</td>
<td>TABLE</td>
<td>Specifies the storage type to use for LOB segments. The options are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• SECUREFILE - LOB storage is returned as SECUREFILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• BASICFILE - LOB storage is returned as BASICFILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DEFAULT - The keyword (SECUREFILE or BASICFILE) is omitted in the <strong>LOB STORE AS</strong> clause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• NO_CHANGE - LOB segments are created with the same storage they had in the source database. This is the default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specifying this transform changes the LOB storage for all tables in the job, including tables that provide storage for materialized views.</td>
</tr>
</tbody>
</table>

Note: The **INMEMORY** transform is available only in Oracle Database 12c Release 1 (12.1.0.2) or higher.

Note: The **INMEMORY_CLAUSE** transform is available only in Oracle Database 12c Release (12.1.0.2) or higher.
### Table 45-19 (Cont.) Transforms Provided by the METADATA_TRANFORM Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Object Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>NUMBER</td>
<td>TYPE TABLE</td>
<td>If zero, inhibits the assignment of the exported OID during type or table creation. Instead, a new OID will be assigned. Use of this transform on Object Tables will cause breakage in REF columns that point to the table. Defaults to 1.</td>
</tr>
<tr>
<td>PCTSPACE</td>
<td>NUMBER</td>
<td>TABLE INDEX TABLESPACE</td>
<td>Specifies a percentage multiplier used to alter extent allocations and data file sizes. Used to shrink large tablespaces for testing purposes. Defaults to 100.</td>
</tr>
<tr>
<td>SEGMENT_ATTRIBUTES</td>
<td>NUMBER</td>
<td>TABLE INDEX</td>
<td>If nonzero (TRUE), emit storage segment parameters. Defaults to 1.</td>
</tr>
<tr>
<td>SEGMENT_CREATION</td>
<td>NUMBER</td>
<td>TABLE</td>
<td>If nonzero (TRUE), the SQL SEGMENT CREATION clause is added to the CREATE TABLE statement. That is, the CREATE TABLE statement will explicitly say either SEGMENT CREATION DEFERRED or SEGMENT CREATION IMMEDIATE. If the value is FALSE, then the SEGMENT CREATION clause is omitted from the CREATE TABLE statement. Set this parameter to FALSE to use the default segment creation attributes for the table(s) being loaded. Defaults to nonzero (TRUE).</td>
</tr>
<tr>
<td>STORAGE</td>
<td>NUMBER</td>
<td>TABLE</td>
<td>If nonzero (TRUE), emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is zero.) Defaults to nonzero (TRUE).</td>
</tr>
<tr>
<td>TABLE_COMPRESSION_CLAUSE</td>
<td>TEXT</td>
<td>TABLE</td>
<td>Specifies a table compression clause (for example, COMPRESS BASIC) to use when the table is created. Specify NONE to omit the table compression clause. The table will have the default compression for the tablespace. Specifying this transform changes the compression type for all tables in the job, including tables that provide storage for materialized views.</td>
</tr>
</tbody>
</table>

### Exceptions
- **INVALID_HANDLE.** The specified handle is not attached to a Data Pump job.
- **INVALID_ARGVAL.** This message can indicate any of the following:
– The mode is transportable, which doesn't support transforms.
– The job's mode does not include the specified object_type.
– The transform has already been specified for the specified value and object_type.

• INVALID_OPERATION. Transforms are only supported for SQL_FILE and Import operations. The job's operation was Export which does not support the use of metadata transforms.
• INVALID_STATE. The user called METADATA_TRANSFORM after the job had started (that is, the job was not in the defining state).
• INCONSISTENT_ARGS. There was no value supplied or it was of the wrong datatype for the transform.
• PRIVILEGE_ERROR. A nonprivileged user attempted to do a REMAP_SCHEMA to a different user's schema or a REMAP_DATAFILE.
• SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
• NO_SUCH_JOB. The specified job does not exist.

Usage Notes
• The METADATA_TRANSFORM procedure is only supported for Import and SQL_FILE operations. It enables you to apply commonly desired, predefined transformations to the definition of objects as part of the transfer. If you need transforms that are not supported within this procedure, you should do a preliminary SQL_FILE operation to produce a SQL script corresponding to the dump file set. By editing the DDL directly and then executing it, you can produce any transformations that you need.
• Transforms for the DataPump API are a subset of the transforms implemented by the DBMS_METADATA.SET_TRANSFORM_PARAMETER API. Multiple transforms can be defined for a single job. However, each transform defined must be unique according its parameters. That is, two transforms cannot specify conflicting or redundant transformations.

45.5.12 OPEN Function

This function is used to declare a new job using the Data Pump API.

The handle that is returned is used as a parameter for calls to all other procedures (but not to the ATTACH function).

Syntax

DBMS_DATAPUMP.OPEN ( 
  operation    IN VARCHAR2,
  job_mode         IN VARCHAR2,
  remote_link  IN VARCHAR2 DEFAULT NULL,
  job_name     IN VARCHAR2 DEFAULT NULL,
  version      IN VARCHAR2 DEFAULT 'COMPATIBLE')
RETURN NUMBER;
Parameters

Table 45-20 OPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>The type of operation to be performed. Table 45-21 contains descriptions of valid operation types.</td>
</tr>
<tr>
<td>job_mode</td>
<td>The scope of the operation to be performed. Table 45-22 contains descriptions of valid modes. Specifying NULL generates an error.</td>
</tr>
<tr>
<td>remote_link</td>
<td>If the value of this parameter is non-null, it provides the name of a database link to the remote database that will be the source of data and metadata for the current job.</td>
</tr>
<tr>
<td>job_name</td>
<td>The name of the job. The name is limited to 128 characters; it will be truncated if more than 128 characters are used. It may consist of printable characters and spaces. It is implicitly qualified by the schema of the user executing the OPEN function and must be unique to that schema (that is, there cannot be other Data Pump jobs using the same name). The name is used to identify the job both within the API and with other database components such as identifying the job in the DBA_RESUMABLE view if the job becomes suspended through lack of resources. If no name is supplied, a system generated name will be provided for the job in the following format: &quot;SYS_&lt;OPERATION&gt;<em>&lt;MODE&gt;</em>%N&quot;. The default job name is formed where %N expands to a two-digit incrementing integer starting at '01' (for example, &quot;SYS_IMPORT_FULL_03&quot;). The name supplied for the job will also be used to name the master table and other resources associated with the job.</td>
</tr>
</tbody>
</table>
| version        | The version of database objects to be extracted. This option is only valid for Export, network Import, and SQL_FILE operations. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are as follows:  
  - COMPATIBLE - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the $COMPATIBILITY view). Database compatibility must be set to 9.2 or higher.  
  - LATEST - the version of the metadata corresponds to the database version.  
  - A specific database version, for example, ‘11.0.0’. Specify a value of 12 to allow all existing database features, components, and options to be exported from Oracle Database 11g release 2 (11.2.0.3) or later into an Oracle Database 12c Release 1 (12.1) (either a multitenant container database (CDB) or a non-CDB). |

Table 45-21 describes the valid operation types for the OPEN function.

Table 45-21 Valid Operation Types for the OPEN Function

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORT</td>
<td>Saves data and metadata to a dump file set or obtains an estimate of the size of the data for an operation.</td>
</tr>
<tr>
<td>IMPORT</td>
<td>Restores data and metadata from a dump file set or across a database link.</td>
</tr>
</tbody>
</table>
Table 45-21 Valid Operation Types for the OPEN Function

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_FILE</td>
<td>Displays the metadata within a dump file set, or from across a network link, as a SQL script. The location of the SQL script is specified through the ADD_FILE procedure.</td>
</tr>
</tbody>
</table>

Table 45-22 describes the valid modes for the OPEN function.

Table 45-22 Valid Modes for the OPEN Function

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>Operates on the full database or full dump file set except for Oracle Database internal schemas. (Some tables from Oracle Database internal schemas may be registered to be exported and imported in full operations in order to provide consistent metadata during import.) The TRANSPORTABLE parameter can be set to ALWAYS during a full database export in order to move data via transportable tablespaces rather than in the Data Pump dump file.</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>Operates on a set of selected schemas. Defaults to the schema of the current user. All objects in the selected schemas are processed. In SCHEMA mode, you cannot specify Oracle-internal schemas (for example, SYS, XDB, ORDSYS, MDSYS, CTXSYS, ORDPLUGINS, or LBACSYS).</td>
</tr>
<tr>
<td>TABLE</td>
<td>Operates on a set of selected tables. Defaults to all of the tables in the current user's schema. Only tables and their dependent objects are processed.</td>
</tr>
<tr>
<td>TABLESPACE</td>
<td>Operates on a set of selected tablespaces. No defaulting is performed. Tables that have storage in the specified tablespaces are processed in the same manner as in Table mode.</td>
</tr>
<tr>
<td>TRANSPORTABLE</td>
<td>Operates on metadata for tables (and their dependent objects) within a set of selected tablespaces to perform a transportable tablespace export/import.</td>
</tr>
</tbody>
</table>

Return Values

- An opaque handle for the job. This handle is used as input to the following procedures: ADD_FILE, CREATE_JOB_VIEW, DATA_FILTER, DETACH, GET_STATUS, LOG_ENTRY, LOG_ERROR, METADATA_FILTER, METADATA_REMAP, METADATA_TRANSFORM, SET_PARALLEL, SET_PARAMETER, START_JOB, STOP_JOB, and WAIT_FOR_JOB

Exceptions

- INVALID_ARGVAL. An invalid operation or mode was specified. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- JOB_EXISTS. A table already exists with the specified job name.
- PRIVILEGE_ERROR. The user does not have the necessary privileges or roles to use the specified mode.
- INTERNAL_ERROR. The job was created under the wrong schema or the master table was of the wrong format.
• SUCCESS_WITH_INFO. The function succeeded, but further information is available through the `GET_STATUS` procedure.

• NO_SUCH_JOB. The specified job does not exist.

Usage Notes

• When the job is created, a master table is created for the job under the caller's schema within the caller's default tablespace. A handle referencing the job is returned that attaches the current session to the job. Once attached, the handle remains valid until either an explicit or implicit detach occurs. The handle is only valid in the caller's session. Other handles can be attached to the same job from a different session by using the `ATTACH` function.

• If the call to the `OPEN` function fails, call the `GET_STATUS` procedure with a null handle to retrieve additional information about the failure.

45.5.13 SET_PARALLEL Procedure

This procedure adjusts the degree of parallelism within a job.

Syntax

```sql
DBMS_DATAPUMP.SET_PARALLEL(
    handle IN NUMBER,
    degree IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of a job. The current session must have previously attached to the handle through a call to either the <code>OPEN</code> or <code>ATTACH</code> function.</td>
</tr>
<tr>
<td>degree</td>
<td>The maximum number of worker processes that can be used for the job. You use this parameter to adjust the amount of resources used for a job.</td>
</tr>
</tbody>
</table>

Exceptions

• INVALID_HANDLE. The specified handle is not attached to a Data Pump job.

• INVALID_OPERATION. The `SET_PARALLEL` procedure is only valid for export and import operations.

• INVALID_ARGVAL. An invalid value was supplied for an input parameter.

• SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.

• NO_SUCH_JOB. The specified job does not exist.

Usage Notes

• The `SET_PARALLEL` procedure is only available in the Enterprise Edition of the Oracle database.

• The `SET_PARALLEL` procedure can be executed by any session attached to a job. The job must be in one of the following states: Defining, Idling, or Executing.
• The effect of decreasing the degree of parallelism may be delayed because ongoing work needs to find an orderly completion point before SET_PARALLEL can take effect.

• Decreasing the parallelism will not result in fewer worker processes associated with the job. It will only decrease the number of worker processes that will be executing at any given time.

• Increasing the parallelism will take effect immediately if there is work that can be performed in parallel.

• The degree of parallelism requested by a user may be decreased based upon settings in the resource manager or through limitations introduced by the PROCESSES or SESSIONS initialization parameters in the init.ora file.

• To parallelize an Export job to a degree of \( n \), the user should supply \( n \) files in the dump file set or specify a substitution variable in a file specification. Otherwise, some of the worker processes will be idle while waiting for files.

• SQL_FILE operations always operate with a degree of 1. Jobs running in the Transportable mode always operate with a degree of 1.

45.5.14 SET_PARAMETER Procedures

This procedure is used to specify job-processing options.

Syntax

```sql
DBMS_DATAPUMP.SET_PARAMETER(
  handle       IN NUMBER,
  name         IN VARCHAR2,
  value        IN VARCHAR2);

DBMS_DATAPUMP.SET_PARAMETER (      
  handle      IN NUMBER,
  name        IN VARCHAR2,
  value       IN VARCHAR2);
```

Parameters

**Table 45-24** SET_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of a job. The current session must have previously attached to the handle through a call to the OPEN function.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the parameter. <strong>Table 45-25</strong> describes the valid parameter names.</td>
</tr>
<tr>
<td>value</td>
<td>The value for the specified parameter</td>
</tr>
</tbody>
</table>

**Table 45-25** describes the valid options for the name parameter of the SET_PARAMETER procedure.
### Table 45-25  Valid Options for the name Parameter in the SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Datatype</th>
<th>Supported Operations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_COMMAND</td>
<td>TEXT</td>
<td>All</td>
<td>An opaque string used to describe the current operation from the client's perspective. The command-line procedures will use this string to store the original command used to invoke the job.</td>
</tr>
<tr>
<td>COMPRESSION</td>
<td>TEXT</td>
<td>Export</td>
<td>Allows you to trade off the size of the dump file set versus the time it takes to perform export and import operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The DATA_ONLY option compresses only user data in the dump file set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The METADATA ONLY option compresses only metadata in the dump file set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The ALL option compresses both user data and metadata.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The NONE option stores the dump file set in an uncompressed format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The METADATA_ONLY and NONE options require a job version of 10.2 or later. All other options require a job version of 11.1 or later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Default=METADATA_ONLY</td>
</tr>
<tr>
<td>COMPRESSION_ALG</td>
<td>TEXT</td>
<td>Export</td>
<td>Indicates the compression algorithm is to be used when compressing dump file data. The choices are as follows:</td>
</tr>
<tr>
<td>ORITHM</td>
<td></td>
<td></td>
<td>• BASIC--Offers a good combination of compression ratios and speed; the algorithm used is the same as in previous versions of Oracle Data Pump.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• LOW---Least impact on backup throughput and suited for environments where CPU resources are the limiting factor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• MEDIUM---Recommended for most environments. This option, like the BASIC option, provides a good combination of compression ratios and speed, but it uses a different algorithm than BASIC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• HIGH--Best suited for exports over slower networks where the limiting factor is network speed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To use this feature, the COMPATIBLE initialization parameter must be set to at least 12.0.0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This feature requires that the Oracle Advanced Compression option be enabled.</td>
</tr>
<tr>
<td>DATA_ACCESS_METHOD</td>
<td>TEXT</td>
<td>Export and Import</td>
<td>Allows you to specify an alternative method to unload data if the default method does not work for some reason. The choices are AUTOMATIC, DIRECT_PATH, or EXTERNAL_TABLE. Oracle recommends that you use the default option (AUTOMATIC) whenever possible because it allows Data Pump to automatically select the most efficient method.</td>
</tr>
</tbody>
</table>
Table 45-25  (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Datatype</th>
<th>Supported Operations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_OPTIONS</td>
<td>Number</td>
<td>Export and Import</td>
<td>A bitmask to supply special options for processing the job. The possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· KU$DATAOPT_SKIP_CONST_ERR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· KU$DATAOPT_XMLTYPE_CLOB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· KU$DATAOPT_DISABL_APPEND_HINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· KU$DATAOPT_GRP_PART_TAB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· KU$DATAOPT_TRUST_EXIST_TB_PAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· KU$DATAOPT_CONT_LOAD_ON_FMT_ERR</td>
</tr>
</tbody>
</table>

Export supports the value KU$DATAOPT_XMLTYPE_CLOB. This option stores compressed XMLType columns in the dump file in CLOB format rather than as XML documents.

Note: XMLType stored as CLOB is deprecated as of Oracle Database 12c Release 1 (12.1).

Import supports the value KU$DATAOPT_SKIP_CONST_ERR. This option specifies that if constraint violations occur while data is being imported into user tables, the rows that cause the violations will be rejected and the load will continue. If this option is not set, a constraint error will abort the loading of the entire partition (or table for unpartitioned tables). Setting this option may affect performance, especially for pre-existing tables with unique indexes or constraints.

Import also supports the value KU$DATAOPT_DISABL_APPEND_HINT. This option prevents the append hint from being applied to the data load. Disabling the APPEND hint can be useful if there is a small set of data objects to load that already exist in the database and some other application may be concurrently accessing one or more of the data objects.

Data Pump Export supports the value KU$DATAOPT_GRP_PART_TAB. This option tells Data Pump to unload all table data in one operation rather than unload each table partition as a separate operation. As a result, the definition of the table will not matter at import time because Import will see one partition of data that will be loaded into the entire table.

Data Pump Import supports the value KU$DATAOPT_TRUST_EXIST_TB_PAR. This option tells Data Pump to load partition data in parallel into existing tables.

Use of the DATA_OPTIONS parameter requires that the version on the OPEN function be set to 11.1 or later.

Default=0

Data Pump Import supports the value KU$DATAOPT_CONT_LOAD_ON_FMT_ERR. This option tells Data Pump to skip forward to the start of the next granule if a stream format error is encountered while loading table data. Most stream format errors are caused by corrupt dump files. This value can be used if Data Pump encounters a stream format error and the original export database is not available to export the table data again. If Data Pump skips over data, not all data...
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Datatype</th>
<th>Supported Operations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPTION</td>
<td>TEXT</td>
<td>Export</td>
<td>from the source database is imported potentially skipping hundreds or thousands of rows. Specifies what to encrypt in the dump file set, as follows: ALL enables encryption for all data and metadata in the export operation. DATA_ONLY specifies that only data is written to the dump file set in encrypted format. ENCRYPTED_COLUMNS_ONLY specifies that only encrypted columns are written to the dump file set in encrypted format. METADATA_ONLY specifies that only metadata is written to the dump file set in encrypted format. NONE specifies that no data is written to the dump file set in encrypted format. This parameter requires a job version of 11.1 or later. The default value depends upon the combination of encryption-related parameters that are used. To enable encryption, either ENCRYPTION or ENCRYPTION_PASSWORD or both, must be specified. If only ENCRYPTION_PASSWORD is specified, then ENCRYPTION defaults to ALL. If neither ENCRYPTION nor ENCRYPTION_PASSWORD is specified, then ENCRYPTION defaults to NONE. To specify ALL, DATA_ONLY, or METADATA_ONLY, the COMPATIBLE initialization parameter must be set to at least 11.1. <strong>NOTE:</strong> If the data being exported includes SecureFiles that you want to be encrypted, then you must specify ENCRYPTION=ALL to encrypt the entire dump file set. Encryption of the entire dump file set is the only way to achieve encryption security for SecureFiles during a Data Pump export operation.</td>
</tr>
<tr>
<td>ENCRYPTION_ALGO</td>
<td>TEXT</td>
<td>Export</td>
<td>Identifies which cryptographic algorithm should be used to perform encryption. Possible values are AES128, AES192, and AES256. The ENCRYPTION_ALGORITHM parameter requires that you also specify either ENCRYPTION or ENCRYPTION_PASSWORD; otherwise an error is returned. See Oracle Database Advanced Security Guide for information about encryption algorithms. This parameter requires a job version of 11.1 or later. Default=AES128</td>
</tr>
</tbody>
</table>
Table 45-25  (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Datatype</th>
<th>Supported Operations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPTION_MODE</td>
<td>TEXT</td>
<td>Export</td>
<td>Identifies the types of security used for encryption and decryption. The values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>PASSWORD</strong> requires that you provide a password when creating encrypted dump file sets. You will need to provide the same password when you import the dump file set. <strong>PASSWORD</strong> mode requires that you also specify the <strong>ENCRYPTION_PASSWORD</strong> parameter. The <strong>PASSWORD</strong> mode is best suited for cases in which the dump file set will be imported into a different or remote database, but which must remain secure in transit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>TRANSPARENT</strong> allows an encrypted dump file set to be created without any intervention from a database administrator (DBA), provided the required Oracle Encryption Wallet is available. Therefore, the <strong>ENCRYPTION_PASSWORD</strong> parameter is not required, and will in fact, cause an error if it is used in <strong>TRANSPARENT</strong> mode. This encryption mode is best suited for cases in which the dump file set will be imported into the same database from which it was exported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>DUAL</strong> creates a dump file set that can later be imported using either the Oracle Encryption Wallet or the password that was specified with the <strong>ENCRYPTION_PASSWORD</strong> parameter. <strong>DUAL</strong> mode is best suited for cases in which the dump file set will be imported on-site using the Oracle Encryption Wallet, but which may also need to be imported offsite where the Oracle Encryption Wallet is not available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When you use the <strong>ENCRYPTION_MODE</strong> parameter, you must also use either the <strong>ENCRYPTION</strong> or <strong>ENCRYPTION_PASSWORD</strong> parameter. Otherwise, an error is returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To use <strong>DUAL</strong> or <strong>TRANSPARENT</strong> mode, the <strong>COMPATIBLE</strong> initialization parameter must be set to at least 11.1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The default mode depends on which other encryption-related parameters are used. If only <strong>ENCRYPTION</strong> is specified, then the default mode is <strong>TRANSPARENT</strong>. If <strong>ENCRYPTION_PASSWORD</strong> is specified and the Oracle Encryption Wallet is open, then the default is <strong>DUAL</strong>. If <strong>ENCRYPTION_PASSWORD</strong> is specified and the Oracle Encryption Wallet is closed, then the default is <strong>PASSWORD</strong>.</td>
</tr>
</tbody>
</table>

| ENCRYPTION_PASSWORD     | TEXT     | Export and Import    | For export operations, this parameter is required if **ENCRYPTION_MODE** is set to either **PASSWORD** or **DUAL**. It is also required for transportable export/import operations (**job mode**=FULL and **TRANSPORTABLE**=ALWAYS) when the database includes either encrypted tablespaces or tables with encrypted columns. |
Table 45-25   (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Datatype</th>
<th>Supported Operations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESTIMATE</td>
<td>TEXT</td>
<td>Export and Import</td>
<td>Specifies that the estimate method for the size of the tables should be performed before starting the job. If \textit{BLOCKS}, a size estimate for the user tables is calculated using the count of blocks allocated to the user tables. If \textit{STATISTICS}, a size estimate for the user tables is calculated using the statistics associated with each table. If no statistics are available for a table, the size of the table is estimated using \textit{BLOCKS}. The \textit{ESTIMATE} parameter cannot be used in Transportable Tablespace mode. Default=\textit{STATISTICS}</td>
</tr>
<tr>
<td>ESTIMATE_ONLY</td>
<td>NUMBER</td>
<td>Export</td>
<td>Specifies that only the estimation portion of an export job should be performed. This option is useful for estimating the size of dump files when the size of the export is unknown.</td>
</tr>
<tr>
<td>FLASHBACK_SCN</td>
<td>NUMBER</td>
<td>Export and network Import</td>
<td>System change number (SCN) to serve as transactionally consistent point for reading user data. If neither \textit{FLASHBACK_SCN} nor \textit{FLASHBACK_TIME} is specified, there will be no transactional consistency between partitions, except for logical standby databases and Streams targets. \textit{FLASHBACK_SCN} is not supported in Transportable mode. For \textit{FLASHBACK_SCN}, Data Pump supports the new 8-byte big SCNs used in Oracle Database 12c release 2 (12.2).</td>
</tr>
<tr>
<td>FLASHBACK_TIME</td>
<td>TEXT</td>
<td>Export and network Import</td>
<td>Either the date and time used to determine a consistent point for reading user data or a string of the form \textit{TO_TIMESTAMP(...).} If neither \textit{FLASHBACK_SCN} nor \textit{FLASHBACK_TIME} is specified, there will be no transactional consistency between partitions. \textit{FLASHBACK_SCN} and \textit{FLASHBACK_TIME} cannot both be specified for the same job. \textit{FLASHBACK_TIME} is not supported in Transportable mode.</td>
</tr>
<tr>
<td>INCLUDE_METADAT_A</td>
<td>NUMBER</td>
<td>Export and Import</td>
<td>If nonzero, metadata for objects will be moved in addition to user table data. If zero, metadata for objects will not moved. This parameter converts an Export operation into an unload of user data and an Import operation into a load of user data. \textit{INCLUDE_METADATA} is not supported in Transportable mode. Default=1</td>
</tr>
<tr>
<td>KEEP_MASTER</td>
<td>NUMBER</td>
<td>Export and Import</td>
<td>Specifies whether the master table should be deleted or retained at the end of a Data Pump job that completes successfully. The master table is automatically retained for jobs that do not complete successfully. Default=0.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Datatype</td>
<td>Supported Operations</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOGTIME</td>
<td>TEXT</td>
<td>Export and Import</td>
<td>Specifies that messages displayed during export and import operations be timestamped. Valid options are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- NONE—No timestamps on status or log file messages (this is the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- STATUS—Timestamps on status messages only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- LOGFILE—Timestamps on log file messages only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ALL—Timestamps on both status and log file messages</td>
</tr>
<tr>
<td>MASTER_ONLY</td>
<td>NUMBER</td>
<td>Import</td>
<td>Indicates whether to import just the master table and then stop the job so that the contents of the master table can be examined. Default=0.</td>
</tr>
<tr>
<td>METRICS</td>
<td>NUMBER</td>
<td>Export and Import</td>
<td>Indicates whether additional information about the job should be reported to the Data Pump log file. Default=0.</td>
</tr>
<tr>
<td>PARTITION_OPTSION</td>
<td>TEXT</td>
<td>Import</td>
<td>Specifies how partitioned tables should be handled during an import operation. The options are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- NONE means that partitioning is reproduced on the target database as it existed in the source database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DEPARTITION means that each partition or subpartition that contains storage in the job is reproduced as a separate unpartitioned table. Intermediate partitions that are subpartitioned are not re-created (although their subpartitions are converted into tables). The names of the resulting tables are system-generated from the original table names and partition names unless the name is overridden by the REMAP_TABLE metadata transform.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- MERGE means that each partitioned table is re-created in the target database as an unpartitioned table. The data from all of the source partitions is merged into a single storage segment. This option is not supported for transportable jobs or when the TRANSPORTABLE parameter is set to ALWAYS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This parameter requires a job version of 11.1 or later. Default=NONE</td>
</tr>
<tr>
<td>REUSE_DATAFILES</td>
<td>NUMBER</td>
<td>Import</td>
<td>Specifies whether the import job should reuse existing data files for tablespace creation. Default=0.</td>
</tr>
<tr>
<td>SKIP_UNUSABLE_INDEXES</td>
<td>NUMBER</td>
<td>Import</td>
<td>If nonzero, rows will be inserted into tables having unusable indexes. SKIP_UNUSABLE_INDEXES is not supported in Transportable mode. Default=1</td>
</tr>
<tr>
<td>SOURCE_EDITION</td>
<td>TEXT</td>
<td>Export and network Import</td>
<td>The application edition that will be used for determining the objects that will be unloaded for export and for network import.</td>
</tr>
<tr>
<td>STREAMS_CONFIGURATION</td>
<td>NUMBER</td>
<td>Import</td>
<td>Specifies whether to import any Streams metadata that may be present in the export dump file. Default=1.</td>
</tr>
</tbody>
</table>
Table 45-25  (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

| Parameter Name         | Datatype | Supported Operations | Meaning \n|------------------------|----------|----------------------|-----------------------------------------------|
| TABLE_EXISTS_ACTION    | TEXT     | Import               | Specifies the action to be performed when data is loaded into a preexisting table. The possible actions are: TRUNCATE, REPLACE, APPEND, and SKIP.  
  - If INCLUDE_METADATA=0, only TRUNCATE and APPEND are supported.  
  - If TRUNCATE, rows are removed from a preexisting table before inserting rows from the Import.  
  - Note that if TRUNCATE is specified on tables referenced by foreign key constraints, the TRUNCATE will be modified into a REPLACE.  
  - If REPLACE, preexisting tables are replaced with new definitions. Before creating the new table, the old table is dropped.  
  - If APPEND, new rows are added to the existing rows in the table.  
  - If SKIP, the preexisting table is left unchanged.  
  - TABLE_EXISTS_ACTION is not supported in Transportable mode.  
  - The default is SKIP if metadata is included in the import. The default is APPEND if INCLUDE_METADATA is set to 0. |
| TABLESPACE_DATAFILE    | TEXT     | Import               | Specifies the full file specification for a data file in the transportable tablespace set. TABLESPACE_DATAFILE is only valid for transportable mode imports.  
  - TABLESPACE_DATAFILE can be specified multiple times, but the value specified for each occurrence must be different. |
| TARGET_EDITION         | TEXT     | Import               | The application edition that will be used for determining where the objects will be loaded for import and for network import. |
### Table 45-25  (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Datatype</th>
<th>Supported Operations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSPORTABLE</td>
<td>TEXT</td>
<td>Export (and network import</td>
<td>This option is for export operations done in table mode, and also for full-mode imports and network imports. It allows the data to be moved using transportable tablespaces. In table-mode storage segments in the moved tablespaces that are not associated with the parent schemas (tables) will be reclaimed at import time. If individual partitions are selected in a table-mode job, only the tablespaces referenced by those partitions will be moved. During import, the moved partitions can only be reconstituted as tables by using the PARTITION_OPTIONS=DEPARTITION parameter. Use of the TRANSPORTABLE parameter prohibits the subsequent import of the dump file into a database at a lower version or using different character sets. Additionally, the data files may need to be converted if the target database is on a different platform. In table-mode, the TRANSPORTABLE parameter is not allowed if a network link is supplied on the OPEN call. The possible values for this parameter are as follows: ALWAYS - data is always moved by moving data files. This option is valid only for table mode and full mode. NEVER - data files are never used for copying user data This parameter requires a job version of 11.1 or later This parameter requires a job version of 12.1 or later when the job mode is FULL. Default=NEVER</td>
</tr>
<tr>
<td>TTS_FULL_CHECK</td>
<td>NUMBER</td>
<td>Export</td>
<td>If nonzero, verifies that a transportable tablespace set has no dependencies (specifically, IN pointers) on objects outside the set, and vice versa. Only valid for Transportable mode Exports. Default=0</td>
</tr>
<tr>
<td>USER_METADATA</td>
<td>NUMBER</td>
<td>Export and network Import</td>
<td>For schema-mode operations, specifies that the metadata to re-create the users' schemas (for example, privilege grants to the exported schemas) should also be part of the operation if set to nonzero. Users must be privileged to explicitly set this parameter. The USER_METADATA parameter cannot be used in Table, Tablespace, or Transportable Tablespace mode. Default=1 if user has DATAPUMP_EXP_FULL_DATABASE role; 0 otherwise.</td>
</tr>
</tbody>
</table>

### Exceptions

- **INVALID_HANDLE.** The specified handle is not attached to a Data Pump job.
- **INVALID_ARGVAL.** This exception could be due to any of the following causes:
  - An invalid name was supplied for an input parameter
  - The wrong datatype was used for value
  - A value was not supplied
– The supplied value was not allowed for the specified parameter name
– A flashback parameter had been established after a different flashback parameter had already been established
– A parameter was specified that did not support duplicate definitions
  • INVALID_OPERATION. The operation specified is invalid in this context.
  • INVALID_STATE. The specified job is not in the Defining state.
  • INCONSISTENT_ARGS. Either the specified parameter is not supported for the current operation type or it is not supported for the current mode.
  • PRIVILEGE_ERROR. The user does not have the DATAPUMP_EXP_FULL_DATABASE or DATAPUMP_IMP_FULL_DATABASE role required for the specified parameter.
  • SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
  • NO_SUCH_JOB. The specified job does not exist.

Usage Notes
• The SET_PARAMETER procedure is used to specify optional features for the current job. See Table 45-25 for a list of supported options.

45.5.15 START_JOB Procedure

This procedure begins or resumes job execution.

Syntax

```sql
DBMS_DATAPUMP.START_JOB (       handle IN NUMBER,     skip_current IN NUMBER DEFAULT 0,     abort_step IN NUMBER DEFAULT 0,     cluster_ok IN NUMBER DEFAULT 1,     service_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 45-26  START_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.</td>
</tr>
<tr>
<td>skip_current</td>
<td>If nonzero, causes actions that were 'in progress' on a previous execution of the job to be skipped when the job restarts. The skip will only be honored for Import jobs. This mechanism allows the user to skip actions that trigger fatal bugs and cause the premature termination of a job. Multiple actions can be skipped on a restart. The log file will identify which actions are skipped. If a domain index was being processed, all pieces of the domain index are skipped even if the error occurred in only a subcomponent of the domain index. A description of the actions skipped is entered into the log file. skip_current is ignored for the initial START_JOB in a job. If zero, no data or metadata is lost upon a restart.</td>
</tr>
</tbody>
</table>
Table 45-26  (Cont.) START_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort_step</td>
<td>Value must be 0. Inserting values other than 0 into this argument will have unintended consequences.</td>
</tr>
<tr>
<td>cluster_ok</td>
<td>If = 0, all workers are started on the current instance. Otherwise, workers are started on instances usable by the job.</td>
</tr>
<tr>
<td>service_name</td>
<td>If specified, indicates a service name used to constrain the job to specific instances or to a specific resource group.</td>
</tr>
</tbody>
</table>

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_STATE. The causes of this exception can be any of the following:
  - No files have been defined for an Export, non-network Import, or SQL_FILE job
  - An ADD_FILE procedure has not been called to define the output for a SQL_FILE job
  - A TABLESPACE_DATAFILE parameter has not been defined for a Transportable Import job
  - A TABLESPACE_EXPR metadata filter has not been defined for a Transportable or Tablespace mode Export or Network job
  - The dump file set on an Import or SQL_FILE job was either incomplete or missing a master table specification
- INVALID_OPERATION. Unable to restore master table from a dump file set.
- INTERNAL_ERROR. An inconsistency was detected when the job was started. Additional information may be available through the GET_STATUS procedure.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- When this procedure is called to request that the corresponding job be started or restarted, the state of the job is changed from either the Defining or Idling state to the Executing state.
- If the SET_PARALLEL procedure was not called prior to the START_JOB procedure, the initial level of parallelism used in the job will be 1. If SET_PARALLEL was called prior to the job starting, the degree specified by the last SET_PARALLEL call determines the parallelism for the job. On restarts, the parallelism is determined by the previous parallel setting for the job, unless it is overridden by another SET_PARALLEL call.
- To restart a stopped job, an ATTACH function must be performed prior to executing the START_JOB procedure.
45.5.16 STOP_JOB Procedure

This procedure terminates a job, but optionally, preserves the state of the job.

Syntax

```sql
DBMS_DATAPUMP.STOP_JOB (
    handle      IN NUMBER,
    immediate   IN NUMBER DEFAULT 0,
    keep_master IN NUMBER DEFAULT NULL,
    delay       IN NUMBER DEFAULT 60);
```

Parameters

Table 45-27  STOP_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of a job. The current session must have previously attached to the handle through a call to either the <code>OPEN</code> or <code>ATTACH</code> function. At the end of the procedure, the user is detached from the handle.</td>
</tr>
<tr>
<td>immediate</td>
<td>If nonzero, the worker processes are aborted immediately. This halts the job quickly, but parts of the job will have to be rerun if the job is ever restarted. If zero, the worker processes are allowed to complete their current work item (either metadata or table data) before they are terminated. The job is placed in a Stop Pending state while the workers finish their current work.</td>
</tr>
<tr>
<td>keep_master</td>
<td>If nonzero, the master table is retained when the job is stopped. If zero, the master table is dropped when the job is stopped. If the master table is dropped, the job will not be restartable. If the master table is dropped during an export job, the created dump files are deleted.</td>
</tr>
<tr>
<td>delay</td>
<td>The number of seconds to wait until other attached sessions are forcibly detached. The delay allows other sessions attached to the job to be notified that a stop has been performed. The job keeps running until either all clients have detached or the delay has been satisfied. If no delay is specified, then the default delay is 60 seconds. If a shorter delay is used, clients might not be able to retrieve the final messages for the job through the <code>GET_STATUS</code> procedure.</td>
</tr>
</tbody>
</table>

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_STATE`. The job is already in the process of being stopped or completed.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- This procedure is used to request that the corresponding job stop executing.
- The termination of a job that is in an Executing state may take several minutes to complete in an orderly fashion.
• For jobs in the Defining, Idling, or Completing states, this procedure is functionally equivalent to the DETACH procedure.
• Once a job is stopped, it can be restarted using the ATTACH function and START_JOB procedures, provided the master table and the dump file set are left intact.
• If the KEEP_MASTER parameter is not specified, and the job is in the Defining state or has a mode of Transportable, the master table is dropped. Otherwise, the master table is retained.

45.5.17 WAIT_FOR_JOB Procedure

This procedure runs a job until it either completes normally or stops for some other reason.

Syntax

```
DBMS_DATAPUMP.WAIT_FOR_JOB (  
    handle      IN   NUMBER,  
    job_state   OUT  VARCHAR2);
```

Parameters

Table 45-28  WAIT_FOR_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle of the job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function. At the end of the procedure, the user is detached from the handle.</td>
</tr>
<tr>
<td>job_state</td>
<td>The state of the job when it has stopped executing; either STOPPED or COMPLETED.</td>
</tr>
</tbody>
</table>

Exceptions

• SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS API.
• INVALID_HANDLE. The job handle is no longer valid.

Usage Notes

This procedure provides the simplest mechanism for waiting for the completion of a Data Pump job. The job should be started before calling WAIT_FOR_JOB. When WAIT_FOR_JOB returns, the job will no longer be executing. If the job completed normally, the final status will be COMPLETED. If the job stopped executing because of a STOP_JOB request or an internal error, the final status will be STOPPED.
The **DBMS_DB_VERSION** package specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.

### 46.1 DBMS_DB_VERSION Overview

The **DBMS_DB_VERSION** package specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.

The package for the Oracle Database 18c Release 1 version is shown below.

```sql
PACKAGE DBMS_DB_VERSION IS
  VERSION CONSTANT PLS_INTEGER := 18; -- RDBMS version number
  RELEASE CONSTANT PLS_INTEGER := 1;  -- RDBMS release number
  ver_le_9_1    CONSTANT BOOLEAN := FALSE;
  ver_le_9_2    CONSTANT BOOLEAN := FALSE;
  ver_le_9      CONSTANT BOOLEAN := FALSE;
  ver_le_10_1   CONSTANT BOOLEAN := FALSE;
  ver_le_10_2   CONSTANT BOOLEAN := FALSE;
  ver_le_10     CONSTANT BOOLEAN := FALSE;
  ver_le_11_1   CONSTANT BOOLEAN := FALSE;
  ver_le_11_2   CONSTANT BOOLEAN := FALSE;
  ver_le_11     CONSTANT BOOLEAN := FALSE;
  ver_le_12_1   CONSTANT BOOLEAN := FALSE;
  ver_le_12_2   CONSTANT BOOLEAN := TRUE;
  ver_le_12     CONSTANT BOOLEAN := TRUE;
  ver_le_18     CONSTANT BOOLEAN := TRUE;
END DBMS_DB_VERSION;
```

The boolean constants follow a naming convention. Each constant gives a name for a boolean expression. For example:

- **VER_LE_9_1** represents version <= 9 and release <= 1
- **VER_LE_10_2** represents version <= 10 and release <= 2
- **VER_LE_10** represents version <= 10
A typical usage of these boolean constants is:

```$IF DBMS_DB_VERSION.VER_LE_10 $THEN
   version 10 and earlier code
$ELSIF DBMS_DB_VERSION.VER_LE_11 $THEN
   version 11 code
$ELSE
   version 12 and later code
$END```

This code structure will protect any reference to the code for version 12. It also prevents the controlling package constant `DBMS_DB_VERSION.VER_LE_11` from being referenced when the program is compiled under version 10. A similar observation applies to version 11. This scheme works even though the static constant `VER_LE_11` is not defined in version 10 database because conditional compilation protects the `$ELSIF` from evaluation if `DBMS_DB_VERSION.VER_LE_10` is TRUE.

### 46.2 DBMS_DB_VERSION Constants

The `DBMS_DB_VERSION` package contains different constants for different Oracle Database releases.

The Oracle Database 18c Release 1 version of the `DBMS_DB_VERSION` package uses the constants shown in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION</td>
<td>PLS_INTEGER</td>
<td>18</td>
<td>Current version</td>
</tr>
<tr>
<td>RELEASE</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>Current release</td>
</tr>
<tr>
<td>VER_LE_9</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;= 9</td>
</tr>
<tr>
<td>VER_LE_9_1</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;= 9 and release &lt;= 1</td>
</tr>
<tr>
<td>VER_LE_9_2</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;= 9 and release &lt;= 2</td>
</tr>
<tr>
<td>VER_LE_10</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;= 10</td>
</tr>
<tr>
<td>VER_LE_10_1</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;= 10 and release &lt;= 1</td>
</tr>
<tr>
<td>VER_LE_10_2</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;= 10 and release &lt;= 2</td>
</tr>
<tr>
<td>VER_LE_11</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;= 11</td>
</tr>
<tr>
<td>VER_LE_11_1</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;=11 and release &lt;= 1</td>
</tr>
<tr>
<td>VER_LE_11_2</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;=11 and release &lt;= 2</td>
</tr>
<tr>
<td>VER_LE_12</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;=12</td>
</tr>
<tr>
<td>VER_LE_12_1</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;=12 and release &lt;= 1</td>
</tr>
<tr>
<td>VER_LE_12_2</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Version &lt;=12 and release &lt;= 2</td>
</tr>
<tr>
<td>VER_LE_18</td>
<td>BOOLEAN</td>
<td>TRUE</td>
<td>Version &lt;=12</td>
</tr>
</tbody>
</table>
46.3 DBMS_DB_VERSION Examples

This example uses conditional compilation to guard new features.

CREATE OR REPLACE PROCEDURE whetstone IS

-- Notice that conditional compilation constructs
-- can interrupt a regular PL/SQL statement.
-- You can locate a conditional compilation directive anywhere
-- there is whitespace in the regular statement.

SUBTYPE my_real IS
    $IF DBMS_DB_VERSION.VER_LE_9 $THEN NUMBER
    $ELSE BINARY_DOUBLE
    $END;

    t  CONSTANT my_real :=
        $IF DBMS_DB_VERSION.VER_LE_9 $THEN 0.499975
        $ELSE 0.499975d
        $END;

    t2 CONSTANT my_real := 
        $IF DBMS_DB_VERSION.VER_LE_9 $THEN 2.0
        $ELSE 2.0d
        $END;

    x  CONSTANT my_real := 
        $IF DBMS_DB_VERSION.VER_LE_9 $THEN 1.0
        $ELSE 1.0d
        $END;

    y  CONSTANT my_real := 
        $IF DBMS_DB_VERSION.VER_LE_9 $THEN 1.0
        $ELSE 1.0d
        $END;

    z  MY_REAL;

PROCEDURE P(x IN my_real, y IN my_real, z OUT NOCOPY my_real) IS
    x1 my_real;
    y1 my_real;
BEGIN
    x1 := x;
    y1 := y;
    x1 := t * (x1 + y1);
    y1 := t * (x1 + y1);
    z := (x1 + y1)/t2;
END P;
BEGIN
    P(x, y, z);
    DBMS_OUTPUT.PUT_LINE ('z = '|| z);
END whetstone;
/
The DBMS_DBCOMP package performs block comparison to detect lost writes or database inconsistencies between a primary database and one or more physical standby databases. It contains one procedure, DBCOMP, which can be executed at any time. (It does not require that the DB_LOST_WRITE_PROTECT initialization parameter be enabled.)

This chapter contains the following sections:

• Using DBMS_DBCOMP
• DBMS_DBCOMP Security Model
• DBCOMP Procedure

Related Topics
• Oracle Data Guard Concepts and Administration

47.1 Using DBMS_DBCOMP

The DBMS_DBCOMP.DBCOMP procedure assumes that there is one primary database and one or more physical standby databases. The databases should be at least mounted before block comparison.

Note:

Logical standby databases, far sync instances, and cascaded standbys cannot be the target database for the DBMS_DBCOMP.DBCOMP procedure.

While this procedure is running, you can monitor progress of the compare operation by querying the V$SESSION_LONGOPS view. For example:

SQL> SELECT target_desc, sofar, totalwork FROM V$SESSION_LONGOPS WHERE opname = 'BlockCompare';

The output might look as follows:

<table>
<thead>
<tr>
<th>TARGET_DESC</th>
<th>SO_FAR</th>
<th>TOTALWORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compared Blocks</td>
<td>23914</td>
<td>340639</td>
</tr>
<tr>
<td>Lost Writes</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

The results can be interpreted as follows:

• The Compared Blocks value is the number of blocks that were scanned (including those that did not use compare callback functions). The number is the sum of all slaves (that is, each slave is responsible for a connection between primary and standby). For example,
there are two slaves and each slave has compared 1000 blocks, then the value of SOFAR should be 2000.

- The Lost Writes value is the number of lost writes found at both the primary and the standby. Also, it is the sum of all slaves. For Lost Writes, the value of SOFAR should be always equal to the value of TOTALWORK.
- The value of SOFAR should be always less than or equal to the value of TOTALWORK.
- When the block compare slave processes finish, the value of SOFAR should be equal to the value of TOTALWORK.

### 47.2 DBMS_DBCOMP Security Model

The DBMS_DBCOMP package runs with invoker’s rights and requires the SYSDBA or SYSDG or SYSBACKUP privilege.

### 47.3 Summary of the DBMS_DBCOMP Subprogram

The DBMS_DBCOMP package contains one procedure, DBCOMP.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCOMP procedure</td>
<td>The DBMS_DBCOMP.DBCOMP procedure performs block comparison to detect lost writes or database inconsistencies between a primary database and one or more physical standby databases. The DBCOMP procedure can be executed on the primary or on the physical standby while the databases are mounted or open.</td>
</tr>
</tbody>
</table>

### 47.3.1 DBCOMP Procedure

Use DBCOMP procedure to compare blocks on a primary database and one or more physical standby databases.

**Syntax**

```sql
DBMS_DBCOMP.DBCOMP (  
datafile IN varchar2,  
outputfile IN varchar2,  
block_dump IN boolean);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datafile</td>
<td>This can be a data file name or a data file number. Specify ‘ALL’ to compare all data files.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>outputfile</td>
<td>This value is used as a prefix in the name of the output file. By default, output files are placed in the dbs/ directory, but the output file argument can contain a relative directory path or an absolute directory path. See Usage Notes for descriptions of the information contained in the output file.</td>
</tr>
<tr>
<td>block_dump</td>
<td>This is a Boolean parameter to specify whether block content should be dumped into output files if a pair of blocks of primary and standby databases is not identical. This parameter does not guarantee that all different block pairs are dumped. The default value is FALSE.</td>
</tr>
</tbody>
</table>

**Usage Notes**

- The output file contains the following columns:
  - **ID**: The block type. For instance, type 02 is undo block type, and 06 is KTB-managed data block type.
  - **TOTAL**: The total number of blocks in the specific block type.
  - **CORR**: The number of corrupted blocks.
  - **SKIPPED**: The number of block pairs that are skipped (ignored) without comparison, such as direct loaded blocks, RMAN optimized blocks, and flashback optimized blocks.
  - **DIFFV**: The number of blocks that have different version numbers between primary and standby.
  - **SAMEV**: The number of block pairs that have the same version number between primary and standby.
  - **SAMEV&C**: The number of block pairs that have the same version and the same checksum.
  - **ENCERR**: The number of block pairs that have encryption related errors (usually because the Wallet is not open).
  - **LWLOC**: The number of lost writes on the local database.
  - **LWRMT**: The number of lost writes on the remote databases.
  - **DIFFPAIR**: The number of block pairs that have the same version but different checksum, and which the block compare callback function has concluded are different.

**Example 47-1  Example**

The following is an example of using the `DBMS_DBCOMP.DBCOMP` procedure.

```
declare
  Datafile_Name_or_Number varchar2(1000);
  Output_File varchar2(1000);
begin
  dbms_output.enable(1000000);
```
Datafile_Name_or_Number:= 'all' ;
Output_File:='BlockCompareAll_';
sys.dbms_dbc.comp.DBCOMP(Datafile_Name_or_Number, Output_File, true);
end;
The `DBMS_DBFS_CONTENT` package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Exceptions
- Operational Notes
- Data Structures
- Summary of DBMS_DBFS_CONTENT Subprograms

**See Also:**

- Oracle Database SecureFiles and Large Objects Developer's Guide

### 48.1 DBMS_DBFS_CONTENT Overview

The `DBMS_DBFS_CONTENT` package provides an interface that connects the Oracle database to the DBFS client-side.

In the server, the `DBMS_DBFS_CONTENT` package is backed by the `DBMS_DBFS_CONTENT_SPI` package, which includes descriptions but not actual implementations of DBFS stores.

**See Also:**

- Oracle Database SecureFiles and Large Objects Developer's Guide for a description of `DBMS_DBFS_CONTENT` and the DBFS architecture
- Oracle Database SecureFiles and Large Objects Developer's Guide for conceptual information about the `DBMS_DBFS_CONTENT` package

### 48.2 DBMS_DBFS_CONTENT Security Model

The `DBMS_DBFS_CONTENT` package runs under AUTHID CURRENT_USER.
48.3 DBMS_DBFS_CONTENT Constants

The **DBMS_DBFS_CONTENT** package uses various types of constants. These are shown in the following tables:

- Table 48-1
- Table 48-2
- Table 48-3
- Table 48-4
- Table 48-5
- Table 48-6
- Table 48-7
- Table 48-8
- Table 48-9
- Table 48-10

**Path Name Constants and Types**

The following constants are useful for declaring paths and item names. Paths are limited to 1024 characters and item names are limited to 256 characters.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME_MAX</td>
<td>PLS_INTEGER</td>
<td>256</td>
<td>Maximum length of an absolute path name visible to clients</td>
</tr>
<tr>
<td>NAME_T</td>
<td>VARCHAR2(256)</td>
<td>NAME_MAX</td>
<td>Portable alias for string that can represent component names</td>
</tr>
<tr>
<td>PATH_MAX</td>
<td>PLS_INTEGER</td>
<td>1024</td>
<td>Maximum length of any individual component of an absolute path name visible to clients</td>
</tr>
<tr>
<td>PATH_T</td>
<td>VARCHAR2(1024)</td>
<td>PATH_MAX</td>
<td>Portable alias for string that can represent path names</td>
</tr>
</tbody>
</table>

**ContentID Constants**

Stores may expose to the user a unique identifier that represents a particular path item in the store. These identifiers are limited to 128 characters.
### Table 48-2  DBMS_DBFS_CONTENT Constants - ContentID

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENT_ID_MAX</td>
<td>PLS_INTEGER</td>
<td>128</td>
<td>Maximum length of a store-specific provider-generated contentID that identifies a file-type content item</td>
</tr>
<tr>
<td>CONTENT_ID_T</td>
<td>RAW(128)</td>
<td>CONTENT_ID_MAX</td>
<td>Portable alias for raw buffers that can represent contentID values</td>
</tr>
</tbody>
</table>

### Path Properties Constants

Every path name in a store is associated with a set of properties. Each property is identified by a string “name”, has a string “value” (which might be `NULL` if unset or undefined or unsupported by a specific store implementation) and a value “typecode” (a numeric discriminant for the actual type of value held in the “value” string.)

### Table 48-3  DBMS_DBFS_CONTENT Constants - Properties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPNAME_MAX</td>
<td>PLS_INTEGER</td>
<td>32</td>
<td>Maximum length of a property name</td>
</tr>
<tr>
<td>PROPNAME_T</td>
<td>VARCHAR2(32)</td>
<td>PROPNAME_MAX</td>
<td>Portable alias for string that can represent property names</td>
</tr>
<tr>
<td>PROPVAL_MAX</td>
<td>PLS_INTEGER</td>
<td>1024</td>
<td>Maximum length of the string value of a property</td>
</tr>
<tr>
<td>PROPVAL_T</td>
<td>VARCHAR2(1024)</td>
<td>PATH_MAX</td>
<td>Portable alias for string that can represent property values</td>
</tr>
</tbody>
</table>

### Path Name Type Constants

Path items in a store have a item type associated with them. These types represent the kind of entry the item represents in the store.

### Table 48-4  DBMS_DBFS_CONTENT Constants - Path Name Types

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_FILE</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>A regular file storing content (a logically linear sequence of bytes accessed as a BLOB</td>
</tr>
<tr>
<td>TYPE_DIRECTORY</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>A container of other path name types, including file types</td>
</tr>
</tbody>
</table>
Table 48-4  (Cont.) DBMS_DBFS_CONTENT Constants - Path Name Types

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_LINK</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td>A symbolic link (that is, an uninterpreted string value associated with a path name). Since symbolic links may represent path names that fall outside the scope of any given store (or even the entire aggregation of stores managed by the DBMS_DBFS_CONTENT interface), or may not even represent path names, clients must be careful in creating symbolic links, and stores must be careful in trying to resolve these links internally.</td>
</tr>
<tr>
<td>TYPE_REFERENCE</td>
<td>PLS_INTEGER</td>
<td>4</td>
<td>A hard link which is always a valid path name alias to content.</td>
</tr>
</tbody>
</table>

Store Feature Constants

The DBFS content API allows different store providers (and different stores) to describe themselves through a feature set (a bitmask indicating which features they support and which ones they do not).

Table 48-5  DBMS_DBFS_CONTENT Constants - Store Features

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE_FOLDERS</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Set if the store supports folders (or directories) as part of hierarchical path names.</td>
</tr>
<tr>
<td>FEATURE_FOIAT</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>Set if implicit folder operations within the store (performed as part of a client-requested operation) runs inside autonomous transactions. In general, the use of autonomous transactions is a compromise between (a) simplicity in the implementation and client-controlled transaction scope for all operations, at the cost of greatly reduced concurrency (FEATURE_FOIAT not set), versus (b) more complex implementation and smaller client-controlled transaction scope, at the benefit of greatly increased concurrency (FEATURE_FOIAT set).</td>
</tr>
</tbody>
</table>
### Table 48-5  (Cont.) DBMS_DBFS_CONTENT Constants - Store Features

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE_NOWAIT</td>
<td>PLS_INTEGER</td>
<td>4</td>
<td>Set if the store allows nowait gets of path elements. The default behavior is to wait for row locks; if nowait gets are implemented, the get operation raises an ORA-54 exception if the path element is already locked by another transaction.</td>
</tr>
<tr>
<td>FEATURE_ACLS</td>
<td>PLS_INTEGER</td>
<td>8</td>
<td>Set if the store supports Access Control Lists (ACLs) and internal authorization or checking based on these ACLs. ACLs are standard properties but a store may do nothing more than store and retrieve the ACLs without interpreting them in any way.</td>
</tr>
<tr>
<td>FEATURE_LINKS</td>
<td>PLS_INTEGER</td>
<td>16</td>
<td>Set if the store supports symbolic links, and if certain types of symbolic links (specifically non-absolute path names) can be internally resolved by the store itself.</td>
</tr>
<tr>
<td>FEATURE_LINK_DEREF</td>
<td>PLS_INTEGER</td>
<td>32</td>
<td>Set if the store supports symbolic links, and if certain types of symbolic links (specifically non-absolute path names) can be internally resolved by the store itself.</td>
</tr>
<tr>
<td>FEATURE_REFERENCES</td>
<td>PLS_INTEGER</td>
<td>64</td>
<td>Set if the store supports hard links.</td>
</tr>
<tr>
<td>FEATURE_LOCKING</td>
<td>PLS_INTEGER</td>
<td>128</td>
<td>Set if the store supports user-level locks (read-only, write-only, read-write) that can be applied on various items of the store, and if the store uses these lock settings to control various types of access to the locked items. User-level locks are orthogonal to transaction locks and persist beyond the scope of any specific transaction, session, or connection — this implies that the store itself may not be able to clean up after dangling locks, and client-applications need to perform any garbage collection.</td>
</tr>
<tr>
<td>FEATURE_LOCK_HIERARCHY</td>
<td>PLS_INTEGER</td>
<td>256</td>
<td>Set if the store allows a user-lock to control access to the entire sub-tree under the locked path name.</td>
</tr>
<tr>
<td>FEATURE_LOCK_CONVERSION</td>
<td>PLS_INTEGER</td>
<td>512</td>
<td>Set if the store supports upgrade or downgrade of locks from one mode to another.</td>
</tr>
</tbody>
</table>
### Table 48-5  (Cont.) DBMS_DBFS_CONTENT Constants - Store Features

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE_VERSIONING</td>
<td>PLS_INTEGER</td>
<td>1024</td>
<td>Set if the store supports at least a linear versioning and version management. Different versions of the same path name are identified by monotonic version numbers, with a version-nonqualified path name representing the latest version.</td>
</tr>
<tr>
<td>FEATURE_VERSION_PA</td>
<td>PLS_INTEGER</td>
<td>2048</td>
<td>Set if the store supports a hierarchical namespace for different versions of a path name</td>
</tr>
<tr>
<td>FEATURE_SOFT_DELETES</td>
<td>PLS_INTEGER</td>
<td>4096</td>
<td>Set if the store supports a &quot;soft-delete&quot;, that is, the ability to delete a path name and make it invisible to normal operations, but retain the ability to restore the path name later (as long as it has not been overwritten by a new create operation). The store also supports purging soft-deleted path names (making them truly deleted), and navigation modes that show soft-deleted items.</td>
</tr>
<tr>
<td>FEATURE_HASHING</td>
<td>PLS_INTEGER</td>
<td>8192</td>
<td>Set if the store automatically computes and maintains some type of a secure hash of the contents of a path name (typically a TYPE_FILE path).</td>
</tr>
<tr>
<td>FEATURE_HASH_LOOKUP</td>
<td>PLS_INTEGER</td>
<td>16384</td>
<td>Set if the store allows &quot;content-based addressing&quot;, that is, the ability to locate a content item based, not on its path name, but on its content hash.</td>
</tr>
</tbody>
</table>
Table 48-5  (Cont.) DBMS_DBFS_CONTENT Constants - Store Features

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| FEATURE_FILTERING  | PLS_INTEGER| 32768  | Set if the store allows clients to pass a filter function (a PL/SQL function conforming to the signature below) that returns a logical boolean indicating if a given store item satisfies a selection predicate. Stores that support filtering may be able to more efficiently perform item listing, directory navigation, and deletions by embedding the filtering logic inside their implementation. If filtering is not supported, clients can retrieve more items than necessary and perform the filtering checks themselves, albeit less efficiently. A filter predicate is a function with the following signature:

```plsql
function filterFunction(
    path        IN   VARCHAR2,
    store_name  IN   VARCHAR2,
    opcode      IN   INTEGER,
    item_type   IN   INTEGER,
    properties  IN   DBMS_DBFS_CONTENT_PROPERTIES_T,
    content     IN   BLOB)
    RETURN  INTEGER;
```

Any PL/SQL function conforming to this signature can examine the contents and properties of a store item, and determine if the item satisfies the selection criterion for the current operation. Any nonzero return value results in the DBMS_DBFS_CONTENT interface processing the item as part of the current operation; a return value that is zero or NULL results in the item being skipped from processing. |
| FEATURE_SEARCHING  | PLS_INTEGER| 65536  | Set if the store allows clients to pass a text-search filter query to locate type_file path names based on their content. Stores that support searching may use indexes to accelerate such searches; otherwise, clients need to build their own indexes, or else search a potentially larger set of items to locate the ones of interest for the current search. |
| FEATURE_ASOF       | PLS_INTEGER| 131072 | Set if the store allows clients to use a flashback timestamp in query operations (non-mutating GETPATH Procedures and LIST Function). |
### Table 48-5 (Cont.) DBMS_DBFS_CONTENT Constants - Store Features

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE_PROVIDER_PROPERTIES</td>
<td>PLS_INTEGER</td>
<td>262144</td>
<td>Set if the store allows per-operation properties (that control the behavior of the store with regard to the current operation, as opposed to properties associated with individual items).</td>
</tr>
<tr>
<td>FEATURE_SNAPSHOTS</td>
<td>PLS_INTEGER</td>
<td>524288</td>
<td>Set if the store allows the use of named, read-only snapshots of its contents. It is up to the provider to implement snapshots using any suitable means (including creating immediate copies of the content, or using copy-on-write) and managing dependencies between snapshots and its parent content view.</td>
</tr>
<tr>
<td>FEATURE_CLONES</td>
<td>PLS_INTEGER</td>
<td>1048576</td>
<td>Set if the store allows the use of named, writable clones of its contents. It is up to the provider to implement clones using any suitable means (including creating immediate copies of the content, or using copy-on-write) and managing dependencies between clones and its parent content view.</td>
</tr>
<tr>
<td>FEATURE_LOCATOR</td>
<td>PLS_INTEGER</td>
<td>2097152</td>
<td>Set if the store allows direct access to file contents through a LOB locator. Stores that internally manipulate the file contents, perhaps by shredding or reassembling them in separate pieces, performing other transformations, and so on, cannot transparently give out a LOB locator to clients. The file contents of these stores should be accessed using the buffer-based interfaces.</td>
</tr>
<tr>
<td>FEATURE_CONTENT_ID</td>
<td>PLS_INTEGER</td>
<td>4194304</td>
<td>Set if the store allows a &quot;pathless&quot;, contentID-based access to files (there is no notion of a directory, link, or reference in this model)</td>
</tr>
<tr>
<td>FEATURE_LAZY_PATH</td>
<td>PLS_INTEGER</td>
<td>8388608</td>
<td>Set if the store allows a lazy binding of a path name to file content elements that are otherwise identified by a contentID; this feature makes sense only in conjunction with FEATURE_CONTENT_ID</td>
</tr>
</tbody>
</table>

### Lock Type Constants

Stores that support locking should implement 3 types of locks: LOCK_READ_ONLY, LOCK_WRITE_ONLY, and LOCK_READ_WRITE.
### Table 48-6  DBMS_DBFS_CONTENT Constants - Lock Types

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCK_READ_ONLY</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Locks as read-only</td>
</tr>
<tr>
<td>LOCK_WRITE_ONLY</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>Locks as write-only</td>
</tr>
<tr>
<td>LOCK_READ_WRITE</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td>Locks as read-write</td>
</tr>
</tbody>
</table>

### Standard Property Constants

Standard properties are well-defined, mandatory properties associated with all content path names that all stores should support (in the manner described by the content interface), with some exceptions. For example, a read-only store need not implement a `modification_time` or `creation_time`.

All standard properties informally use the `STD` namespace, which clients and stores should avoid using.

### Table 48-7  DBMS_DBFS_CONTENT Constants - Standard Properties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD_ACCESS_TIME</td>
<td>VARCHAR2(32)</td>
<td>'std:access_time'</td>
<td>TYPECODE_TIMESTAMP in UTC: The time of last access of a path name's contents</td>
</tr>
<tr>
<td>STD_ACL</td>
<td>VARCHAR2(32)</td>
<td>'std:acl'</td>
<td>TYPECODE_VARCHAR2: The access control list (in standard ACL syntax) associated with the path name</td>
</tr>
<tr>
<td>STD_CANONICAL_PATH</td>
<td>VARCHAR2(32)</td>
<td>'std:canonical_path'</td>
<td>TYPECODE_VARCHAR2: The canonical store-specific path name of an item.</td>
</tr>
<tr>
<td>STD_CHANGE_TIME</td>
<td>VARCHAR2(32)</td>
<td>'std:change_time'</td>
<td>TYPECODE_TIMESTAMP in UTC: The time of last change to the metadata of a path name</td>
</tr>
<tr>
<td>STD_CHILDREN</td>
<td>VARCHAR2(32)</td>
<td>'std:children'</td>
<td>TYPECODE_NUMBER: The number of child directories/folders a directory/folder path has (this property should be available in providers that support the FEATURE_FOLDERS feature)</td>
</tr>
<tr>
<td>STD_CONTENT_TYPE</td>
<td>VARCHAR2 (32)</td>
<td>'std:content_type'</td>
<td>TYPECODE_NUMBER: The client-supplied mime-type(s) (in standard RFC syntax) describing the (typically type_file) path name. The content type is not necessarily interpreted by the store.</td>
</tr>
<tr>
<td>STD_CREATION_TIME</td>
<td>VARCHAR2(32)</td>
<td>'std:creation_time'</td>
<td>TYPECODE_TIMESTAMP in UTC: The time at which the item was created (once set, this value never changes for the lifetime of the path name)</td>
</tr>
</tbody>
</table>
### Table 48-7  (Cont.) DBMS_DBFS_CONTENT Constants - Standard Properties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD_DELETED</td>
<td>VARCHAR2(32)</td>
<td>'std:deleted'</td>
<td>TYPECODE_NUMBER as a BOOLEAN: Set to a nonzero number if the path name has been soft-deleted but not yet purged.</td>
</tr>
<tr>
<td>STD_GUID</td>
<td>VARCHAR2(32)</td>
<td>'std:guid'</td>
<td>TYPECODE_NUMBER: A store-specific unique identifier for a path name. Clients must not depend on the GUID being unique across different stores, but a given (store-name, store-specific-path name) has a stable and unique GUID for its lifetime.</td>
</tr>
<tr>
<td>STD_LENGTH</td>
<td>VARCHAR2(32)</td>
<td>'std:length'</td>
<td>TYPECODE_NUMBER: The length of the content (BLOB) of a TYPE_FILE/TYPE_REFERENCE path, or the length of the referent of a TYPE_LINK symbolic link. Directories do not have a well-defined length and stores are free to set this property to zero, NULL, or any other value.</td>
</tr>
<tr>
<td>STD_MODIFICATION_TIME</td>
<td>VARCHAR2(32)</td>
<td>'std:modification_time'</td>
<td>TYPECODE_TIMESTAMP in UTC: The time of last change to the data associated with a path name. Change to the content of a TYPE_FILE/TYPE_REFERENCE path, the referent of the TYPE_LINK path, and addition or deletion of immediate children in a TYPE_DIRECTORY path, all constitute data changes.</td>
</tr>
<tr>
<td>STD_OWNER</td>
<td>VARCHAR2(32)</td>
<td>'std:owner'</td>
<td>TYPECODE_VARCHAR2: A client-supplied (or implicit) owner name for the path name. The owner name may be used (along with the current &quot;principal&quot;) for access checks by stores that support ACLs, locking, or both.</td>
</tr>
<tr>
<td>STD_PARENT_GUID</td>
<td>VARCHAR2(32)</td>
<td>'std:parent_guid'</td>
<td>TYPECODE_NUMBER: A store-specific unique identifier for the parent of a path name. Clients must not depend on the GUID being unique across different stores, but a given (store-name, store-specific-path name) has a stable and unique GUID for its lifetime.</td>
</tr>
</tbody>
</table>

\[ \text{STD\_PARENT\_GUID(path name)} == \text{STD\_GUID(parent(path name))} \]
### Table 48-7  DBMS_DBFS_CONTENT Constants - Standard Properties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD_REFERENT</td>
<td>VARCHAR2(32)</td>
<td>'std:referent'</td>
<td>TYPECODE_VARCHAR2: The content of the symbolic link of a TYPE_LINK path; NULL otherwise. As mentioned, the STD_REFERENT can be an arbitrary string and must not necessarily be interpreted as path name by clients (or such interpretation should be done with great care).</td>
</tr>
</tbody>
</table>

#### Optional Property Constants

Optional properties are well-defined properties (not mandatory) associated with all content path names that all stores are free to support (but only in the manner described by the DBFS content API).

All optional properties informally use the opt: namespace, which clients and stores should avoid using.

### Table 48-8  DBMS_DBFS_CONTENT Constants - Optional Properties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_HASH_TYPE</td>
<td>VARCHAR2(32)</td>
<td>'opt:hash_type'</td>
<td>TYPECODE_NUMBER: The type of hash provided in the opt_hash_value property; see DBMS_CRYPTO for possible options.</td>
</tr>
<tr>
<td>OPT_HASH_VALUE</td>
<td>VARCHAR2(32)</td>
<td>'opt:hash_value'</td>
<td>TYPECODE_NUMBER: The hash value of type OPT_HASH_TYPE describing the content of the path name.</td>
</tr>
<tr>
<td>OPT_LOCK_COUNT</td>
<td>VARCHAR2(32)</td>
<td>'opt:lock_count'</td>
<td>TYPECODE_NUMBER: The number of (compatible) locks placed on a path name. If different principals are allowed to place compatible (read) locks on a path, the opt_locker must specify all lockers (with repeats so that lock counts can be correctly maintained).</td>
</tr>
<tr>
<td>OPT_LOCK_DATA</td>
<td>VARCHAR2(32)</td>
<td>'opt:lock_data'</td>
<td>TYPECODE_NUMBER: The client-supplied user-data associated with a user-lock, uninterpreted by the store.</td>
</tr>
<tr>
<td>OPT_LOCKER</td>
<td>VARCHAR2(32)</td>
<td>'opt:locker'</td>
<td>TYPECODE_NUMBER: One or more implicit or client-specified principals that applied a user-lock on a path name.</td>
</tr>
</tbody>
</table>
Table 48-8  (Cont.) DBMS_DBFS_CONTENT Constants - Optional Properties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_LOCK_STATUS</td>
<td>VARCHAR2(32)</td>
<td>'opt:lock_status'</td>
<td>TYPECODE_NUMBER: One of the LOCK_READ_ONLY, LOCK_WRITE_ONLY, LOCK_READ_WRITE values describing the type of lock currently applied on a path name.</td>
</tr>
<tr>
<td>OPT_VERSION</td>
<td>VARCHAR2(32)</td>
<td>'opt:version'</td>
<td>TYPECODE_NUMBER: A sequence number for linear versioning of a path name.</td>
</tr>
<tr>
<td>OPT_VERSION_PATH</td>
<td>VARCHAR2(32)</td>
<td>'opt:version_path'</td>
<td>TYPECODE_NUMBER: A version-path name for hierarchical versioning of a path name.</td>
</tr>
<tr>
<td>OPT_CONTENT_ID</td>
<td>VARCHAR2(32)</td>
<td>'opt:content_id'</td>
<td>TYPECODE_NUMBER: A provider-generated store-specific unique contentID in the form of a string for a file content element (that may optionally not be associated with a path; see FEATURE_CONTENT_ID and FEATURE_LAZY_PATH).</td>
</tr>
</tbody>
</table>

Property Access Flag Constants

Content interface methods to get or set properties can use combinations of property access flags to fetch properties from different name spaces in a single interface call.

Table 48-9  DBMS_DBFS_CONTENT Constants - Property Access Flags

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROP_NONE</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>None: used when the client is not interested in any properties, and is invoking the content access method for other reasons (path name existence or lockability validation, data access, and so on)</td>
</tr>
<tr>
<td>PROP_STD</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Mandatory: used when the client is interested in the standard properties; all standard properties are retrieved if this flag is specified.</td>
</tr>
<tr>
<td>PROP_OPT</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>Optional: used when the client is interested in the optional properties; all optional properties are retrieved if this flag is specified.</td>
</tr>
<tr>
<td>PROP_USR</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td>User-defined: used when the client is interested in the user-defined properties; all user-defined properties are retrieved if this flag is specified.</td>
</tr>
</tbody>
</table>
Table 48-9  (Cont.) DBMS_DBFS_CONTENT Constants - Property Access Flags

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROP_ALL</td>
<td>PLS_INTEGER</td>
<td>PROP_ST D + PROP(OP T + PROP_US R)</td>
<td>All: an alias for the combination of all standard, optional, and user-defined properties</td>
</tr>
<tr>
<td>PROP_DATA</td>
<td>PLS_INTEGER</td>
<td>8</td>
<td>Content: used when the client is interested only in data access, and does not care about properties</td>
</tr>
<tr>
<td>PROP_SPC</td>
<td>PLS_INTEGER</td>
<td>16</td>
<td>Specific: used when the client is interested in a mix-and-match of different subsets of various property name spaces; the names of the specific properties to fetch are passed into the content interface method call as arguments, and only these property values are fetched and returned to the client. This is useful in cases where there are a very large number of properties potentially accessible, but the client is interested in only a small number of them (and knows the names of these “interesting” properties beforehand). PROP_SPC is applicable only to the various GETPATH operations. Other operations that specify properties ignore PROP_SPC specifications.</td>
</tr>
</tbody>
</table>

Operation Code Constants

All of the operations in the DBFS content API are represented as abstract opcodes.

Clients can use these opcodes to directly and explicitly by invoking the CHECKACCESS Function to verify if a particular operation can be invoked by a given principal on a particular path name.

Table 48-10  DBMS_DBFS_CONTENT Constants - Operation Codes

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP_CREATE</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Create a path item</td>
</tr>
<tr>
<td>OP_CREATEFILE</td>
<td>PLS_INTEGER</td>
<td>OP_CREAT E</td>
<td>Create a file</td>
</tr>
<tr>
<td>OP_CREATELINK</td>
<td>PLS_INTEGER</td>
<td>OP_CREAT E</td>
<td>Create a soft link</td>
</tr>
<tr>
<td>OP_CREATEREFERENCE</td>
<td>PLS_INTEGER</td>
<td>OP_CREAT E</td>
<td>Create a reference (hard link)</td>
</tr>
</tbody>
</table>
Table 48-10 (Cont.) DBMS_DBFS_CONTENT Constants - Operation Codes

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP_DELETE</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>Soft-deletion, purge, and restore operations are all represented by OP_DELETE</td>
</tr>
<tr>
<td>OP_DELETEFILE</td>
<td>PLS_INTEGER</td>
<td>OP_DELETE</td>
<td>Delete a file</td>
</tr>
<tr>
<td>OP_DELETEDIRECTORY</td>
<td>PLS_INTEGER</td>
<td>OP_DELETE</td>
<td>Delete a directory</td>
</tr>
<tr>
<td>OP_RESTORE</td>
<td>PLS_INTEGER</td>
<td>OP_DELETE</td>
<td>Restore a soft-deleted path item</td>
</tr>
<tr>
<td>OP_PURGE</td>
<td>PLS_INTEGER</td>
<td>OP_DELETE</td>
<td>Purge a soft-deleted path item</td>
</tr>
<tr>
<td>OP_READ</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td>Read from a path item</td>
</tr>
<tr>
<td>OP_GET</td>
<td>PLS_INTEGER</td>
<td>OP_READ</td>
<td>Get a path item for either read or update operations</td>
</tr>
<tr>
<td>OP_WRITE</td>
<td>PLS_INTEGER</td>
<td>4</td>
<td>Write a path item</td>
</tr>
<tr>
<td>OP_PUT</td>
<td>PLS_INTEGER</td>
<td>OP_WRITE</td>
<td>Put (write) to a path item</td>
</tr>
<tr>
<td>OP_RENAME</td>
<td>PLS_INTEGER</td>
<td>5</td>
<td>Rename a path item</td>
</tr>
<tr>
<td>OP_RENAMEFROM</td>
<td>PLS_INTEGER</td>
<td>OP_RENAME</td>
<td>Operations performed on the source of a rename</td>
</tr>
<tr>
<td>OP_RENAMETO</td>
<td>PLS_INTEGER</td>
<td>OP_RENAME</td>
<td>Operations performed on the destination of a rename</td>
</tr>
<tr>
<td>OP_SETPATH</td>
<td>PLS_INTEGER</td>
<td>OP_RENAME</td>
<td>Set a path item name</td>
</tr>
<tr>
<td>OP_LIST</td>
<td>PLS_INTEGER</td>
<td>6</td>
<td>Perform a path listing</td>
</tr>
<tr>
<td>OP_SEARCH</td>
<td>PLS_INTEGER</td>
<td>7</td>
<td>Perform a search</td>
</tr>
<tr>
<td>OP_LOCK</td>
<td>PLS_INTEGER</td>
<td>8</td>
<td>Lock a path item</td>
</tr>
<tr>
<td>OP_UNLOCK</td>
<td>PLS_INTEGER</td>
<td>9</td>
<td>Unlock a path item</td>
</tr>
<tr>
<td>OP_ACL</td>
<td>PLS_INTEGER</td>
<td>10</td>
<td>An implicit operation invoked during an OP_CREATE or OP_PUT that specifies a STD_ACL property; the operation tests to see if the principal is allowed to set or change the ACL of a store item</td>
</tr>
<tr>
<td>OP_STORE</td>
<td>PLS_INTEGER</td>
<td>11</td>
<td>A catch-all category for miscellaneous store operations that do not fall under any of the other operational interfaces</td>
</tr>
</tbody>
</table>

48.4 Exceptions

DBFS content API operations can raise any one of these top-level exceptions.
Table 48-11  DBMS_DBFS_CONTENT Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH_EXISTS</td>
<td>64000</td>
<td>A specified path name already exists</td>
</tr>
<tr>
<td>INVALID_PARENT</td>
<td>64001</td>
<td>Parent of a specified path name does not exist</td>
</tr>
<tr>
<td>INVALID_PATH</td>
<td>64002</td>
<td>Specified path name does not exist, or is not valid</td>
</tr>
<tr>
<td>UNSUPPORTED_OPERATION</td>
<td>64003</td>
<td>An operation unsupported by a store was invoked</td>
</tr>
<tr>
<td>INVALID_ARGUMENTS</td>
<td>64004</td>
<td>An operation was invoked with invalid arguments</td>
</tr>
<tr>
<td>INVALID_ACCESS</td>
<td>64005</td>
<td>Access control checks failed for the current operation</td>
</tr>
<tr>
<td>LOCK_CONFLICT</td>
<td>64006</td>
<td>Current operation failed lock conflict check</td>
</tr>
<tr>
<td>INVALID_STORE</td>
<td>64007</td>
<td>An invalid store name was specified</td>
</tr>
<tr>
<td>INVALID_MOUNT</td>
<td>64008</td>
<td>An invalid mount point was specified</td>
</tr>
<tr>
<td>INVALID_PROVIDER</td>
<td>64009</td>
<td>An invalid provider-package was specified</td>
</tr>
<tr>
<td>READONLY_PATH</td>
<td>64010</td>
<td>A mutating operation was invoked on a read-only mount or store</td>
</tr>
</tbody>
</table>

48.5 DBMS_DBFS_CONTENT Operational Notes

This topic lists operational notes for DBMS_DBFS_CONTENT implementation, path names, and other operations.

- Implementation
- Path Names
- Other DBMS_DBFSCONTENT Operations

Implementation

Since the interconnection of the DBMS_DBFS_CONTENT interface and the provider SPI is a 1-to-many pluggable architecture, the interface uses dynamic SQL to invoke methods in the provider SPI, this can lead to runtime errors.

There are no explicit INIT or FINI methods to indicate when the DBMS_DBFS_CONTENT interface plugs or unplugs a particular provider SPI. Provider SPIs must be willing to auto-initialize themselves at any SPI entry-point.

All operations performed by a store provider are “stateless” in that they are complete operations unto themselves. If state is necessary to be maintained for some reason, then the state must be maintained in data structures such as auxiliary tables that can be queried as needed.

Path Names

All path names used in the provider SPI are store-qualified in pair form (store_name, pathname) where the path name is rooted within the store namespace.

Stores and their providers that support contentID-based access (see FEATURE_CONTENT_ID in Table 48-5) also support a form of addressing that is not based on path names. Content items are identified by an explicit store name, a NULL pathname, and possibly a contentID specified as a parameter or by way of the OPT_CONTENT_ID (see Table 48-8) property.
Not all operations are supported with contentID-based access, and applications should depend only on the simplest create or delete functionality being available.

This table lists other operations and provides links to related discussions.

**Other DBMS_DBFS_CONTENT Operations**

### Table 48-12  Other DBMS_DBFS_CONTENT Operations

<table>
<thead>
<tr>
<th>Other Operations</th>
<th>See …</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on creation operations</td>
</tr>
<tr>
<td>Deletion</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on deletion operations</td>
</tr>
<tr>
<td>Get (Retrieve) and Put (Insert)</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Get and Put operations</td>
</tr>
<tr>
<td>Rename and Move</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Rename and Move operations</td>
</tr>
<tr>
<td>Directory Navigation and Search</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Navigation and Search operations</td>
</tr>
<tr>
<td>Locking</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Locking operations</td>
</tr>
<tr>
<td>Access Check</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Access Check operations</td>
</tr>
</tbody>
</table>

## 48.6 DBMS_DBFS_CONTENT Data Structures

The `DBMS_DBFS_CONTENT` package defines `RECORD` types and `TABLE` types.

### RECORD Types

- FEATURE_T Record Type
- MOUNT_T Record Type
- PATH_ITEM_T Record Type
- PROP_ITEM_T Record Type
- PROPERTY_T Record Type
- STORE_T Record Type

### TABLE Types

- FEATURES_T Table Type
- MOUNTS_T Table Type
- PATH_ITEM_T Table Type
- PROP_ITEM_T Table Type
- PROPERTIES_T Table Type
- STORES_T Table Type
Usage Notes

There is an approximate correspondence between DBMS_DBFS_CONTENT_PROPERTY_T and PROPERTY_T — the former is a SQL object type that describes the full property tuple, while the latter is a PL/SQL record type that describes only the property value component.

Likewise, there is an approximate correspondence between DBMS_DBFS_CONTENT_PROPERTIES_T and PROPERTIES_T — the former is a SQL nested table type, while the latter is a PL/SQL hash table type.

Dynamic SQL calling conventions force the use of SQL types, but PL/SQL code may be implemented more conveniently in terms of the hash-table types.

The DBMS_DBFS_CONTENT interface provides convenient utility functions to convert between DBMS_DBFS_CONTENT_PROPERTIES_T and PROPERTIES_T (see propertiesT2H and propertiesH2T).

Clients can query the DBMS_DBFS_CONTENT interface for the list of available stores, determine which store is to handle access to a given path name, and determine the feature set for the store.

48.6.1 DBMS_DBFS_CONTENT FEATURE_T Record Type

This type describes a store mount point and its properties.

Syntax

```sql
TYPE feature_t IS RECORD (
    feature_name    VARCHAR2(32),
    feature_mask    INTEGER,
    feature_state   VARCHAR2(3));
```

Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_name</td>
<td>Name of feature</td>
</tr>
<tr>
<td>feature_mask</td>
<td>Value used to mask off all other bits other than this feature in the feature value</td>
</tr>
<tr>
<td>feature_state</td>
<td>'YES' or 'NO' depending on whether the feature is supported on this store</td>
</tr>
</tbody>
</table>

48.6.2 DBMS_DBFS_CONTENT MOUNT_T Record Type

This type describes a store mount point and its properties.

Syntax

```sql
TYPE mount_t IS RECORD (
    store_name          VARCHAR2(32),
    store_id            NUMBER,
    provider_name       VARCHAR2(32),
    provider_pkg        VARCHAR2(32),
```


provider_id NUMBER,
provider_version VARCHAR2(32),
store_features INTEGER,
store_guid NUMBER,
store_mount NAME_T,
mount_properties DBMS_DBFS_CONTENT_PROPERTIES_T);

Fields

Table 48-14  MOUNT_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>store_id</td>
<td>ID of store</td>
</tr>
<tr>
<td>provider_name</td>
<td>Name of the content store</td>
</tr>
<tr>
<td>provider_pkg</td>
<td>PL/SQL package name for the content store</td>
</tr>
<tr>
<td>provider_id</td>
<td>Unique identifier for the content store</td>
</tr>
<tr>
<td>provider_version</td>
<td>Version number for the content store</td>
</tr>
<tr>
<td>respos_features</td>
<td>Features supported by this content store</td>
</tr>
<tr>
<td>store_guid</td>
<td>Unique ID for this instance of the store</td>
</tr>
<tr>
<td>store_mount</td>
<td>Location at which this store instance is mounted</td>
</tr>
<tr>
<td>mount_properties</td>
<td>Properties for this mount point (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
</tbody>
</table>

48.6.3 DBMS_DBFS_CONTENT PATH_ITEM_T Record Type

A PATH_ITEM_T is a tuple describing a (store, mount) qualified path in a store, with all standard and optional properties associated with it.

Syntax

```sql
TYPE path_item_t IS RECORD ( 
    store                    NAME_T,
    mount                    NAME_T,
    pathname                 PATH_T,
    pathtype                 VARCHAR2(32),
    filedata                 BLOB,
    std_access_time          TIMESTAMP,
    std_acl                  VARCHAR2(1024),
    std_change_time          TIMESTAMP,
    std_children             NUMBER,
    std_content_type         VARCHAR2(1024),
    std_creation_time        TIMESTAMP,
    std_deleted              INTEGER,
    std_guid                 INTEGER,
    std_modification_time    TIMESTAMP,
    std_owner                VARCHAR2(32),
    std_parent_guid          INTEGER,
    std_referent             VARCHAR2(1024),
    opt_hash_type            VARCHAR2(32),
    opt_hash_value           VARCHAR2(128),
    opt_lock_count           INTEGER,
    opt_lock_data            VARCHAR2(128),
);```
opt_locker            VARCHAR2(128),
opt_lock_status       INTEGER,
opt_version           INTEGER,
opt_version_path      PATH_T,
opt_content_id        CONTENT_ID_T);

Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>Name of store</td>
</tr>
<tr>
<td>mount</td>
<td>Location at which instance of store is mounted</td>
</tr>
<tr>
<td>pathname</td>
<td>Name of path to item</td>
</tr>
<tr>
<td>pathtype</td>
<td>Type of object path (see Table 48-4)</td>
</tr>
<tr>
<td>filedata</td>
<td>BLOB locator that can be used to access data in the path item</td>
</tr>
<tr>
<td>std_access_time</td>
<td>Time of last access of a pathname's contents</td>
</tr>
<tr>
<td>std_acl</td>
<td>Access Control List (in standard ACL syntax)</td>
</tr>
<tr>
<td>std_change_time</td>
<td>Time of last change to the metadata of a path name</td>
</tr>
<tr>
<td>std_children</td>
<td>Number of child directories or folders a directory or folder path (this</td>
</tr>
<tr>
<td></td>
<td>property should be available in providers that support the feature_folders</td>
</tr>
<tr>
<td></td>
<td>feature).</td>
</tr>
<tr>
<td>std_content_type</td>
<td>One or more client-supplied mime-types (in standard RFC syntax) describing</td>
</tr>
<tr>
<td></td>
<td>the path name which is typically of type_file. The content type is not</td>
</tr>
<tr>
<td></td>
<td>necessarily interpreted by the store.</td>
</tr>
<tr>
<td>std_creation_time</td>
<td>Time at which the item was created. Once set, this value remains the</td>
</tr>
<tr>
<td></td>
<td>same for the lifetime of the path name.</td>
</tr>
<tr>
<td>std_deleted</td>
<td>Set to a nonzero number if the path name has been soft-deleted but not</td>
</tr>
<tr>
<td></td>
<td>yet purged (see Table 48-5)</td>
</tr>
<tr>
<td>std_guid</td>
<td>Store-specific unique identifier for a path name. Clients must not depend</td>
</tr>
<tr>
<td></td>
<td>on the GUID being unique across different stores, but a given</td>
</tr>
<tr>
<td></td>
<td>store-name, store-specific-pathname has a stable and unique GUID for its</td>
</tr>
<tr>
<td></td>
<td>lifetime.</td>
</tr>
<tr>
<td>std_modification_time</td>
<td>Time of last change to the data associated with a path name.</td>
</tr>
<tr>
<td></td>
<td>Changes to the content of a type_file or type_reference path, the referent</td>
</tr>
<tr>
<td></td>
<td>of the type_link path, and addition or deletion of immediate children in a</td>
</tr>
<tr>
<td></td>
<td>type_directory path, all constitute data changes.</td>
</tr>
<tr>
<td>std_owner</td>
<td>Client-supplied (or implicit) owner name for the path name</td>
</tr>
<tr>
<td>std_parent_guid</td>
<td>Store-specific unique identifier for the parent of a path name. Clients</td>
</tr>
<tr>
<td></td>
<td>must not depend on the GUID being unique across different stores, but a</td>
</tr>
<tr>
<td></td>
<td>given store-name, store-specific-pathname has a stable and unique GUID for</td>
</tr>
<tr>
<td></td>
<td>its lifetime.</td>
</tr>
<tr>
<td>std_parent_guid(pathname) == std_guid(parent(pathname))</td>
<td></td>
</tr>
<tr>
<td>std_referent</td>
<td>Content of the symbolic link of a type_link path, otherwise NULL.</td>
</tr>
<tr>
<td></td>
<td>As mentioned before, the std_referent can be an arbitrary string and</td>
</tr>
<tr>
<td></td>
<td>must not necessarily be interpreted as pathname by clients (or such</td>
</tr>
<tr>
<td></td>
<td>interpretation should be done with great care).</td>
</tr>
</tbody>
</table>
Table 48-15  (Cont.) PATH_ITEM_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt_hash_type</td>
<td>Type of hash provided in the opt_hash_value property (see DBMS_CRYPTO for possible options)</td>
</tr>
<tr>
<td>opt_hash_value</td>
<td>Hash value of type opt_hash_type describing the content of the path name</td>
</tr>
<tr>
<td>opt_lock_count</td>
<td>Number of compatible locks placed on a path name. If different principals are allowed to place compatible (read) locks on a path, the opt_locker must specify all lockers with repeats so that lock counts can be correctly maintained.</td>
</tr>
<tr>
<td>opt_lock_data</td>
<td>Client-supplied user-data associated with a user-lock, uninterpreted by the store</td>
</tr>
<tr>
<td>opt_locker</td>
<td>One or more implicit or client-specified principals that applied a user-lock on a path name</td>
</tr>
<tr>
<td>opt_lock_status</td>
<td>One of the lock_read_only, lock_write_only, lock_read_write values describing the type of lock currently applied on a path name</td>
</tr>
<tr>
<td>opt_version</td>
<td>Sequence number for linear versioning of a path name</td>
</tr>
<tr>
<td>opt_version_path</td>
<td>Version path name for hierarchical versioning of a path name</td>
</tr>
<tr>
<td>opt_content_id</td>
<td>Stringified provider-generated store-specific unique contentID for a file element (that may optionally not be associated with a path (see FEATURE_CONTENT_ID and FEATURE_LAZY_PATH in Table 48-5)</td>
</tr>
</tbody>
</table>

48.6.4 DBMS_DBFS_CONTENT PROP_ITEM_T Record Type

A PROP_ITEM_T is a tuple describing a (store, mount) qualified path in a store, with all user-defined properties associated with it, expanded out into individual (name, value, type) tuples.

Syntax

```sql
TYPE prop_item_t IS RECORD (  
  store               NAME_T,  
  mount                    NAME_T,  
  pathname                 PATH_T,  
  property_name            PROPNAME_T,  
  property_value           PROPVAL_T,  
  property_type            INTEGER);
```

Fields

Table 48-16  PROP_ITEM_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>Name of store</td>
</tr>
<tr>
<td>mount</td>
<td>Location at which instance of store is mounted</td>
</tr>
<tr>
<td>pathname</td>
<td>Name of path to item</td>
</tr>
<tr>
<td>property_name</td>
<td>Name of the property</td>
</tr>
</tbody>
</table>
Table 48-16 (Cont.) PROP_ITEM_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property_value</td>
<td>Value of the property</td>
</tr>
<tr>
<td>property_type</td>
<td>PL/SQL typecode for the property value</td>
</tr>
</tbody>
</table>

48.6.5 PROPERTY_T Record Type

This type describes a single (value, typecode) property value tuple; the property name is implied.

See PROPERTIES_T Table Type for more information.

Syntax

```sql
TYPE property_t IS RECORD (
    propvalue PROPVAL_T,
    typecode INTEGER);
```

Fields

Table 48-17 PROPERTY_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propvalue</td>
<td>Value of property</td>
</tr>
<tr>
<td>typecode</td>
<td>Typecode</td>
</tr>
</tbody>
</table>

48.6.6 DBMS_DBFS_CONTENT STORE_T Record Type

This type describes a store registered with and managed by the DBMS_DBFS_CONTENT interface.

Syntax

```sql
TYPE store_t IS RECORD (
    store_name VARCHAR2(32),
    store_id NUMBER,
    provider_name VARCHAR2(32),
    provider_pkg VARCHAR2(32),
    provider_id NUMBER,
    provider_version VARCHAR2(32),
    store_features INTEGER,
    store_guid NUMBER);
```

Fields

Table 48-18 STORET_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>
Table 48-18  (Cont.) STORET_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>ID of store</td>
</tr>
<tr>
<td>provider_name</td>
<td>Name of the content store</td>
</tr>
<tr>
<td>provider_pkg</td>
<td>PL/SQL package name for the content store</td>
</tr>
<tr>
<td>provider_id</td>
<td>Unique identifier for the content store</td>
</tr>
<tr>
<td>provider_version</td>
<td>Version number for the content store</td>
</tr>
<tr>
<td>respos_features</td>
<td>Features supported by this content store</td>
</tr>
<tr>
<td>store_guid</td>
<td>Unique ID for this instance of the store</td>
</tr>
</tbody>
</table>

48.6.7 DBMS_DBFS_CONTENT FEATURES_T Table Type

A table type of FEATURE_T Record Type.

Syntax

TYPE features_t IS TABLE OF feature_t;

Related Topics

• DBMS_DBFS_CONTENT FEATURE_T Record Type
  This type describes a store mount point and its properties.

48.6.8 MOUNTS_T Table Type

A table type of MOUNT_T Record Type.

Syntax

TYPE mounts_t IS TABLE OF mount_t;

Related Topics

• DBMS_DBFS_CONTENT MOUNT_T Record Type
  This type describes a store mount point and its properties.

48.6.9 DBMS_DBFS_CONTENT PATH_ITEMS_T Table Type

A table type of PATH_ITEM_T Record Type

Syntax

TYPE path_items_t IS TABLE OF path_item_t;

Related Topics

• DBMS_DBFS_CONTENT PATH_ITEM_T Record Type
  A PATH_ITEM_T is a tuple describing a (store, mount) qualified path in a store, with all standard and optional properties associated with it.
48.6.10 DBMS_DBFS_CONTENT PROP_ITEMS_T Table Type

A table type of PATH_ITEM_T Record Type.

Syntax

```
TYPE prop_items_t IS TABLE OF prop_item_t;
```

Related Topics

• DBMS_DBFS_CONTENT PATH_ITEM_T Record Type
  A PATH_ITEM_T is a tuple describing a (store, mount) qualified path in a store, with all standard and optional properties associated with it.

48.6.11 DBMS_DBFS_CONTENT PROPERTIES_T Table Type

This is a name-indexed hash table of property tuples. The implicit hash-table association between the index and the value allows the client to build up the full DBMS_DBFS_CONTENT_PROPERTY_T tuples for a PROPERTIES_T.

Syntax

```
TYPE properties_t IS TABLE OF property_t INDEX BY propname_t;
```

48.6.12 STORES_T Table Type

This type describes a store registered with and managed by the DBMS_DBFS_CONTENT interface.

Syntax

```
TYPE stores_t IS TABLE OF store_t;
```

48.7 Summary of DBMS_DBFS_CONTENT Subprograms

This table lists and describes the subprograms used in the DBMS_DBFS_CONTENT Package.

Table 48-19    DBMS_DBFS_CONTENT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKACCESS Function</td>
<td>Reports if the user (principal) can perform the specified operation on the given path</td>
</tr>
<tr>
<td>CHECKSPI Functions and Procedures</td>
<td>Checks if a user-provided package implements all of the DBMS_DBFS_CONTENT_SPI subprograms with the proper signatures, and reports on the conformance.</td>
</tr>
<tr>
<td>CREATEDIRECTORY Procedures</td>
<td>Creates a directory</td>
</tr>
<tr>
<td>CREATEFILE Procedures</td>
<td>Creates a file</td>
</tr>
<tr>
<td>CREATELINK Procedures</td>
<td>Creates a new reference to the source file system element</td>
</tr>
<tr>
<td>CREATEREFFERENCES Procedures</td>
<td>Creates a physical link to an already existing file system element</td>
</tr>
</tbody>
</table>
### Table 48-19  (Cont.) DBMS_DBFS_CONTENT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECODEFEATURES Function</td>
<td>Given a feature bit set integer value, returns a <code>FEATURES_T</code> table of the feature bits as <code>FEATURE_T</code> records</td>
</tr>
<tr>
<td>DELETECONTENT Procedure</td>
<td>Deletes the file specified by the given contentID</td>
</tr>
<tr>
<td>DELETEDIRECTORY Procedure</td>
<td>Deletes a directory</td>
</tr>
<tr>
<td>DELETEFILE Procedure</td>
<td>Deletes a file</td>
</tr>
<tr>
<td>FEATURENAME Function</td>
<td>Given a feature bit, returns a <code>VARCHAR2</code> of that feature’s name</td>
</tr>
<tr>
<td>FLUSHSTATS Function</td>
<td>Flushes <code>DBMS_DBFS_CONTENT</code> statistics to disk</td>
</tr>
<tr>
<td>GETDEFAULTACL Procedure</td>
<td>Returns the ACL parameter of the default context</td>
</tr>
<tr>
<td>GETDEFAULTASOF Procedure</td>
<td>Returns the <code>asof</code> parameter of the default context</td>
</tr>
<tr>
<td>GETDEFAULTCONTEXT Procedure</td>
<td>Returns the default context</td>
</tr>
<tr>
<td>GETDEFAULTOWNER Procedure</td>
<td>Returns the <code>owner</code> parameter of the default context</td>
</tr>
<tr>
<td>GETDEFAULTPRINCIPAL Procedure</td>
<td>Returns the <code>principal</code> parameter of the default context</td>
</tr>
<tr>
<td>GETFEATURESBYMOUNT Function</td>
<td>Returns features of a store by mount point</td>
</tr>
<tr>
<td>GETFEATURESBYNAME Function</td>
<td>Returns features of a store by store name</td>
</tr>
<tr>
<td>GETFEATURESBYPATH Function</td>
<td>Returns features of a store by path</td>
</tr>
<tr>
<td>GETPATHBYMOUNTID Function</td>
<td>Returns the full absolute path name</td>
</tr>
<tr>
<td>GETPATH Procedures</td>
<td>Returns existing path items (such as files and directories)</td>
</tr>
<tr>
<td>GETPATHBystoreID Function</td>
<td>If the underlying GUID is found in the underlying store, returns the store-qualified path name</td>
</tr>
<tr>
<td>GETPATHNOWAIT Procedures</td>
<td>Implies that the operation is for an update, and, if implemented, allows providers to return an exception (ORA-00054) rather than wait for row locks.</td>
</tr>
<tr>
<td>GETSTOREBYMOUNT Function</td>
<td>Returns a store by way of its mount point</td>
</tr>
<tr>
<td>GETSTOREBYNAME Function</td>
<td>Returns a store by way of its name</td>
</tr>
<tr>
<td>GETSTOREBYPATH Function</td>
<td>Returns a store by way of its path</td>
</tr>
<tr>
<td>GETSTATS Procedure</td>
<td>Returns information about <code>DBMS_DBFS_CONTENT</code> statistics collection</td>
</tr>
<tr>
<td>GETTRACE Function</td>
<td>Returns whether or not <code>DBMS_DBFS_CONTENT</code> tracing is turned on</td>
</tr>
<tr>
<td>GETVERSION Function</td>
<td>Returns the version of the <code>DBMS_DBFS_CONTENT</code> interface in a standardized format associated with a store</td>
</tr>
<tr>
<td>LIST Function</td>
<td>Lists the path items in the specified path meeting the specified filter and other criteria</td>
</tr>
<tr>
<td>LISTALLCONTENT Function</td>
<td>Lists all path items in all mounts</td>
</tr>
<tr>
<td>LISTALLPROPERTIES Function</td>
<td>Returns a table of all properties for all path items in all mounts</td>
</tr>
<tr>
<td>LISTMOUNTS Function</td>
<td>Lists all available mount points, their backing stores, and the store features</td>
</tr>
<tr>
<td>LISTSTORES Function</td>
<td>Lists all available stores and their features</td>
</tr>
</tbody>
</table>
### Table 48-19  (Cont.) DBMS_DBFS_CONTENT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKPATH Procedure</td>
<td>Applies user-level locks to the given valid path name</td>
</tr>
<tr>
<td>MOUNTSTORE Procedure</td>
<td>Mounts a previously registered store and binds it to the mount point</td>
</tr>
<tr>
<td>NORMALIZEPATH Functions</td>
<td>Converts a store-specific or full-absolute path name into normalized form</td>
</tr>
<tr>
<td>PROPANY Functions</td>
<td>Provides constructors that take one of a variety of types and return a PROPERTY_T</td>
</tr>
<tr>
<td>PROPERTIESH2T Function</td>
<td>Converts a PROPERTY_T hash to a DBMS_DBFS_CONTENT_PROPERTIES_T table</td>
</tr>
<tr>
<td>PROPERTIEST2H Function</td>
<td>Converts a DBMS_DBFS_CONTENT_PROPERTIES_T table to a PROPERTY_T hash</td>
</tr>
<tr>
<td>PROPNUMBER Function</td>
<td>Is a constructor that takes a NUMBER and returns a PROPERTY_T</td>
</tr>
<tr>
<td>PROPRAW Function</td>
<td>Is a constructor that takes a RAW and returns a PROPERTY_T</td>
</tr>
<tr>
<td>PROPTIMESTAMP Function</td>
<td>Is a constructor that takes a TIMESTAMP and returns a PROPERTY_T</td>
</tr>
<tr>
<td>PROPVARCHAR2 Function</td>
<td>Is a constructor that takes a VARCHAR2 and returns a PROPERTY_T</td>
</tr>
<tr>
<td>PURGEALL Procedure</td>
<td>Purges all soft-deleted entries matching the path and optional filter criteria</td>
</tr>
<tr>
<td>PURGEPATH Procedure</td>
<td>Purges any soft-deleted versions of the given path item</td>
</tr>
<tr>
<td>PUTPATH Procedures</td>
<td>Creates a new path item</td>
</tr>
<tr>
<td>REGISTERSTORE Procedure</td>
<td>Registers a new store</td>
</tr>
<tr>
<td>RENAMEPATH Procedures</td>
<td>Renames or moves a path</td>
</tr>
<tr>
<td>RESTOREALL Procedure</td>
<td>Restores all soft-deleted path items meeting the path and filter criteria</td>
</tr>
<tr>
<td>RESTOREPATH Procedure</td>
<td>Restores all soft-deleted path items that match the given path and filter criteria</td>
</tr>
<tr>
<td>SETDEFAULTACL Procedure</td>
<td>Sets the ACL parameter of the default context</td>
</tr>
<tr>
<td>SETDEFAULTASOF Procedure</td>
<td>Sets the &quot;as of&quot; parameter of the default context</td>
</tr>
<tr>
<td>SETDEFAULTCONTEXT Procedure</td>
<td>Sets the default context</td>
</tr>
<tr>
<td>SETDEFAULTOWNER Procedure</td>
<td>Sets the &quot;owner&quot; parameter of the default context</td>
</tr>
<tr>
<td>SETDEFAULTPRINCIPAL Procedure</td>
<td>Sets the &quot;principal&quot; parameter of the default context</td>
</tr>
<tr>
<td>SETPATH Procedures</td>
<td>Assigns a path name to a path item represented by contentID</td>
</tr>
<tr>
<td>SETSTATS Procedure</td>
<td>Enables and disables statistics collection</td>
</tr>
<tr>
<td>SETTRACE Procedure</td>
<td>Sets DBMS_DBFSCONTENT tracing on or off</td>
</tr>
<tr>
<td>SPACEUSAGE Procedure</td>
<td>Queries file system space usage statistics</td>
</tr>
<tr>
<td>TRACE Procedure</td>
<td>Returns a CLOB that contains the evaluation results</td>
</tr>
<tr>
<td>TRACEENABLED Function</td>
<td>Determines if the current trace &quot;severity&quot; set by the SETTRACE Procedure is at least as high as the given trace level</td>
</tr>
<tr>
<td>UNLOCKPATH Procedure</td>
<td>Unlocks path items that were previously locked with the LOCKPATH Procedure</td>
</tr>
</tbody>
</table>
Table 48-19  (Cont.) DBMS_DBFS_CONTENT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNMOUNTSTORE Procedure</td>
<td>Unmounts a registered store</td>
</tr>
<tr>
<td>UNREGISTERSTORE Procedure</td>
<td>Unregisters a store</td>
</tr>
</tbody>
</table>

48.7.1 CHECKACCESS Function

This function reports if the user (principal) can perform the specified operation on the given path. This enables verifying the validity of an operation without attempting to perform the operation. If CHECKACCESS returns 0, then the subprogram invoked to implement that operation should fail with an error.

Syntax

```sql
DBMS_DBFS_CONTENT.CHECKACCESS (path IN VARCHAR2, pathtype IN INTEGER, operation IN VARCHAR2, principal IN VARCHAR2, store_name IN VARCHAR2 DEFAULT NULL) RETURN BOOLEAN;
```

Parameters

Table 48-20  CHECKACCESS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to check for access</td>
</tr>
<tr>
<td>pathtype</td>
<td>Type of object path represents (see Table 48-4)</td>
</tr>
<tr>
<td>operation</td>
<td>Operation to be checked (see Table 48-8)</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Usage Notes

Whether or not the user invokes this function, a store that supports access control internally performs these checks to guarantee security.

48.7.2 CHECKSPI Functions and Procedures

Given the name of a putative DBMS_DBFS_CONTENT_SPI conforming package, this function or procedure checks whether the package implements all of the provider subprograms with the proper signatures, and reports on the conformance.

Syntax

```sql
DBMS_DBFS_CONTENT.CHECKSPI (package_name IN VARCHAR2) RETURN CLOB;
```
DBMS_DBFS_CONTENT.CHECKSPI (  
schema_name         IN              VARCHAR2,  
package_name        IN              VARCHAR2)  
return  clob;

DBMS_DBFS_CONTENT.CHECKSPI (  
package_name        IN              VARCHAR2,  
chk                 IN OUT NOCOPY   CLOB);

DBMS_DBFS_CONTENT.CHECKSPI (  
schema_name         in              VARCHAR2,  
package_name        in              VARCHAR2,  
chk                 in out nocopy   CLOB);

Parameters

Table 48-21  CHECKSPI Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name</td>
<td>Name of package</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of schema</td>
</tr>
<tr>
<td>chk</td>
<td>CLOB that contains the evaluation results</td>
</tr>
</tbody>
</table>

Usage Notes

- The functional form returns a cached temporary LOB of session duration with the results of the analysis. The caller is expected to manage the lifetime of this LOB, as needed.

- The procedural form generates the results of the analysis into the chk LOB parameter; if the value passed in is NULL, the results are written to the foreground trace file provided that DBMS_DBFS_CONTENT interface tracing is enabled. If neither tracing is enabled nor a valid LOB passed in, the checker does not provide any useful indication of the analysis (other than raise exceptions if it encounters a serious error).

- If schema_name is NULL, standard name resolution rules (current schema, private synonym, public synonym) are used to try and locate a suitable package to analyze.

48.7.3 CREATEDIRECTORY Procedures

This procedure creates a directory.

Syntax

DBMS_DBFS_CONTENT.CREATEDIRECTORY (  
path        IN              VARCHAR2,  
properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
prop_flags  IN              INTEGER     DEFAULT PROP_STD,  
recurse     IN              BOOLEAN     DEFAULT FALSE,  
store_name  IN              VARCHAR2    DEFAULT NULL,  
principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.CREATEDIRECTORY (  
path        IN              VARCHAR2,  
properties  IN OUT NOCOPY   PROPERTIES_T,  
prop_flags  IN              INTEGER     DEFAULT PROP_STD,  
recurse     IN              BOOLEAN     DEFAULT FALSE,  
store_name  IN              VARCHAR2    DEFAULT NULL,  
principal   IN              VARCHAR2    DEFAULT NULL);
Parameters

Table 48-22  CREATEDIRECTORY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to the directory</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned, or both, depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting PROP_SPC (see Table 48-9), and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively; otherwise, recursively create the directories above the given directory</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
</tbody>
</table>

48.7.4 CREATEFILE Procedures

This procedure creates a file.

Syntax

```sql
DBMS_DBFS_CONTENT.CREATEFILE ( |
    path        IN              VARCHAR2, |
    properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T, |
    content     IN OUT NOCOPY   BLOB, |
    prop_flags  IN              INTEGER     DEFAULT (PROP_STD + PROP_DATA), |
    store_name  IN              VARCHAR2    DEFAULT NULL, |
    principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 48-23  CREATEFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to the file</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned, or both, depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
</tbody>
</table>
48.7.5 CREATELINK Procedures

This procedure creates a new link element `srcPath` with the value of `dstPath`. The value of `dstPath` is not validated or interpreted in any way by this procedure. This is analogous to a UNIX file system symbolic link.

Syntax

```sql
DBMS_DBFS_CONTENT.CREATELINK (srcPath IN VARCHAR2,
                               dstPath IN VARCHAR2,
                               properties IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
                               prop_flags IN INTEGER DEFAULT PROP_STD,
                               store_name IN VARCHAR2 DEFAULT NULL,
                               principal IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcPath</td>
<td>File system entry to create.</td>
</tr>
<tr>
<td>dstPath</td>
<td>Value to associate with <code>srcPath</code>.</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned depending, or both, on <code>prop_flags</code> (see <code>DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</code>)</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is <code>PROP_STD</code>. Specify properties to be returned by setting <code>prop_spec</code>, and providing an instance of the <code>DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</code> with properties whose values are of interest.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
</tbody>
</table>

### 48.7.6 CREATEREFERENCE Procedures

This procedure creates a physical link, `srcPath`, to an already existing file system element, `dstPath` (such as file or directory). The resulting entry shares the same metadata structures as the value of the `dstPath` parameter, and so is similar to incrementing a reference count on the file system element. This is analogous to a UNIX file system hard link.

**Syntax**

```sql
DBMS_DBFS_CONTENT.CREATEREFERENCE (  
    srcPath IN              VARCHAR2,  
    dstPath IN              VARCHAR2,  
    properties IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
    prop_flags IN              INTEGER     DEFAULT PROP_STD,  
    store_name IN              VARCHAR2    DEFAULT NULL,  
    principal IN              VARCHAR2    DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcPath</td>
<td>File system entry to create.</td>
</tr>
<tr>
<td>dstPath</td>
<td>Path that is the reference to <code>srcPath</code>.</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned, or both, depending on <code>prop_flags</code> (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned. Default is PROP_STD. Specify properties to be returned by setting <code>prop_spec</code>, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
</tbody>
</table>
48.7.7 DECODEFEATURES Function

Given a feature bit set integer value, this function returns a FEATURES_T table of the feature bits as FEATURE_T records.

Syntax

```sql
DBMS_DBFS_CONTENT.DECODEFEATURES (
    featureSet          IN      INTEGER
) RETURN FEATURES_T DETERMINISTIC PIPELINED;
```

Parameters

Table 48-26 DECODEFEATURES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureSet</td>
<td>Feature set</td>
</tr>
</tbody>
</table>

Return Values

FEATURES_T Table Type

48.7.8 DELETECONTENT Procedure

This procedure deletes the file specified by the given contentID.

Syntax

```sql
DBMS_DBFS_CONTENT.DELETECONTENT (
    store_name     IN     VARCHAR2    DEFAULT NULL,
    contentID      IN     RAW,
    filter         IN     VARCHAR2    DEFAULT NULL,
    soft_delete    IN     BOOLEAN     DEFAULT NULL,
    principal      IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 48-27 DELETECONTENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>contentID</td>
<td>Unique identifier for the file to be deleted</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>soft_delete</td>
<td>If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see Oracle Database SecureFiles and Large Objects Developer's Guide, Deletion Operations).</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
</tbody>
</table>
48.7.9 DELETEDIRECTORY Procedure

This procedure deletes a directory.

If `recurse` is nonzero, it recursively deletes all elements of the directory. A filter, if supplied, determines which elements of the directory are deleted.

**Syntax**

```sql
DBMS_DBFS_CONTENT.DELETEDIRECTORY (  
  path IN VARCHAR2,  
  filter IN VARCHAR2 DEFAULT NULL,  
  soft_delete IN BOOLEAN DEFAULT NULL,  
  recurse IN BOOLEAN DEFAULT FALSE,  
  store_name IN VARCHAR2 DEFAULT NULL,  
  principal IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

Table 48-28  DELETEDIRECTORY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to the directory</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>soft_delete</td>
<td>If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete see Oracle Database SecureFiles and Large Objects Developer's Guide, Deletion Operations.</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively. Otherwise, recursively delete the directories and files below the given directory.</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
</tbody>
</table>

48.7.10 DELETEFILE Procedure

This procedure deletes the specified file.

**Syntax**

```sql
DBMS_DBFS_CONTENT.DELETEFILE (  
  path IN VARCHAR2,  
  filter IN VARCHAR2 DEFAULT NULL,  
  soft_delete IN BOOLEAN DEFAULT NULL,  
  recurse IN BOOLEAN DEFAULT FALSE,  
  store_name IN VARCHAR2 DEFAULT NULL,  
  principal IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

Table 48-29  DELETEFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to the file</td>
</tr>
</tbody>
</table>
Table 48-29  (Cont.) DELETEFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>soft_delete</td>
<td>If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see Oracle Database SecureFiles and Large Objects Developer's Guide, Deletion Operations).</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
</tbody>
</table>

48.7.11 FEATURENAME Function

Given a feature bit, this function returns a VARCHAR2 of that feature's name.

Syntax

DBMS_DBFS_CONTENT.FEATURENAME (featureBit IN INTEGER) RETURN VARCHAR2 DETERMINISTIC;

Parameters

Table 48-30  FEATURENAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureBit</td>
<td>Bit representation of the feature (see Table 48-5)</td>
</tr>
</tbody>
</table>

Return Values

Name of the feature

48.7.12 FLUSHSTATS Function

This procedure flushes DBMS_DBFS_CONTENT statistics to disk.

Syntax

DBMS_DBFS_CONTENT.FLUSHSTATS;

48.7.13 GETDEFAULTACL Procedure

This procedure returns the ACL parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

DBMS_DBFS_CONTENT.GETDEFAULTACL (acl OUT NOCOPY VARCHAR2);
### GETDEFAULTACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>ACL for all new elements created (implicitly or explicitly) by the current operation</td>
</tr>
</tbody>
</table>

### GETDEFAULTASOF Procedure

This procedure returns the "as of" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

**Syntax**

```sql
DBMS_DBFS_CONTENT.GETDEFAULTASOF (asof OUT NOCOPY TIMESTAMP);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
</tbody>
</table>

### GETDEFAULTCONTEXT Procedure

This procedure returns the default context. The information contained in the context can be inserted explicitly by way of arguments to the various method calls, allowing for fine-grained control over individual operations.

**Syntax**

```sql
DBMS_DBFS_CONTENT.GETDEFAULTCONTEXT (principal OUT NOCOPY VARCHAR2,
                                     owner OUT NOCOPY VARCHAR2,
                                     acl OUT NOCOPY VARCHAR2,
                                     asof OUT NOCOPY TIMESTAMP);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
<tr>
<td>owner</td>
<td>Owner for new elements created (implicitly or explicitly) by the current operation</td>
</tr>
</tbody>
</table>
Table 48-33  (Cont.) GETDEFAULTCONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>ACL for all new elements created (implicitly or explicitly) by the current operation</td>
</tr>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
</tbody>
</table>

48.7.16 GETDEFAULTOWNER Procedure

This procedure returns the "owner" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```sql
DBMS_DBFS_CONTENT.GETDEFAULTOWNER (
    principal    IN     VARCHAR2);
```

Parameters

Table 48-34  GETDEFAULTOWNER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner for new elements created (implicitly or explicitly) by the current operation</td>
</tr>
</tbody>
</table>

48.7.17 GETDEFAULTPRINCIPAL Procedure

This procedure returns the "principal" parameter of the default context. This information contained can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```sql
DBMS_DBFS_CONTENT.GETDEFAULTPRINCIPAL (
    principal    OUT NOCOPY     VARCHAR2);
```

Parameters

Table 48-35  GETDEFAULTPRINCIPAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>
48.7.18 GETFEATURESBYMOUNT Function

This function returns features of a store by mount point.

Syntax

```sql
DBMS_DBFS_CONTENT.GETFEATURESBYMOUNT
    (store_mount IN VARCHAR2)
    RETURN INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_mount</td>
<td>Mount point</td>
</tr>
</tbody>
</table>

Return Values

A bit mask of supported features (see FEATURES_T Table Type)

48.7.19 GETFEATURESBYNAME Function

This function returns features of a store by store name.

Syntax

```sql
DBMS_DBFS_CONTENT.GETFEATURESBYNAME
    (store_name IN VARCHAR2)
    RETURN INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Return Values

A bit mask of supported features (see FEATURES_T Table Type)

48.7.20 GETFEATURESBYPATH Function

This function returns features of a store by path.

Syntax

```sql
DBMS_DBFS_CONTENT.GETFEATURESBYPATH
    (path IN PATH_T)
    RETURN INTEGER;
```
Parameters

Table 48-38 GETFEATURESBYPATH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>PATH_T</td>
</tr>
</tbody>
</table>

Return Values

A bit mask of supported features (see FEATURES_T Table Type)

48.7.21 GETPATH Procedures

This procedure returns existing path items (such as files and directories). This includes both data and metadata (properties).

The client can request (using prop_flags) that specific properties be returned. File path names can be read either by specifying a BLOB locator using the prop_data bitmask in prop_flags (see Table 48-9) or by passing one or more RAW buffers.

When forUpdate is 0, this procedure also accepts a valid asof timestamp parameter as part of ctx that can be used by stores to implement "as of" style flashback queries. Mutating versions of the GETPATH Procedures do not support these modes of operation.

Syntax

DBMS_DBFS_CONTENT.GETPATH (
  path        IN              VARCHAR2,
  properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
  content     OUT    NOCOPY   BLOB,
  item_type   OUT             INTEGER,
  prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                   PROPOPT +
                                   PROP_DATA),
  asof        IN              TIMESTAMP   DEFAULT NULL,
  forUpdate   IN              BOOLEAN     DEFAULT FALSE,
  deref       IN              BOOLEAN     DEFAULT FALSE,
  store_name  IN              VARCHAR2    DEFAULT NULL,
  principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
  path        IN              VARCHAR2,
  properties  IN OUT NOCOPY   PROPERTIES_T,
  content     OUT    NOCOPY   BLOB,
  item_type   OUT             INTEGER,
  prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                   PROP_OPT +
                                   PROP_DATA),
  asof        IN              TIMESTAMP   DEFAULT NULL,
  forUpdate   IN              BOOLEAN     DEFAULT FALSE,
  deref       IN              BOOLEAN     DEFAULT FALSE,
  store_name  IN              VARCHAR2    DEFAULT NULL,
  principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
  path        IN              VARCHAR2,
  properties  IN OUT NOCOPY   PROPERTIES_T,
  content     OUT    NOCOPY   BLOB,
  item_type   OUT             INTEGER,
  prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                   PROP_OPT +
                                   PROP_DATA),
  asof        IN              TIMESTAMP   DEFAULT NULL,
  forUpdate   IN              BOOLEAN     DEFAULT FALSE,
  deref       IN              BOOLEAN     DEFAULT FALSE,
  store_name  IN              VARCHAR2    DEFAULT NULL,
  principal   IN              VARCHAR2    DEFAULT NULL);
DBMS_DBFS_CONTENT.GETPATH (path IN VARCHAR2,
properties IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
amount IN OUT NUMBER,
offset IN NUMBER,
buffers OUT NOCOPY RAW,
prop_flags IN INTEGER DEFAULT (PROP_STD +
PROP_OPT +
PROP_DATA),
asof IN TIMESTAMP DEFAULT NULL,
store_name IN VARCHAR2 DEFAULT NULL,
principal IN VARCHAR2 DEFAULT NULL);

Parameters

Table 48-39  GETPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to path items</td>
</tr>
</tbody>
</table>
Table 48-39  (Cont.) GETPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>One or more properties and their values to be returned depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>content</td>
<td>BLOB holding data which populates the file (optional)</td>
</tr>
<tr>
<td>item_type</td>
<td>Type of the path item specified (see Table 48-4)</td>
</tr>
<tr>
<td>amount</td>
<td>On input, number of bytes to be read. On output, number of bytes read</td>
</tr>
<tr>
<td>offset</td>
<td>Byte offset from which to begin reading</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer to which to write</td>
</tr>
<tr>
<td>buffers</td>
<td>Buffers to which to write</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
<tr>
<td>forUpdate</td>
<td>Specifies that a lock should be taken to signify exclusive write access to the path item</td>
</tr>
<tr>
<td>deref</td>
<td>If nonzero, attempts to resolve the given path item to actual data provided it is a reference</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

48.7.22 GETPATHBYMOUNTID Function

If the underlying GUID is found in the underlying store, this function returns the full absolute path name.

Syntax

```sql
DBMS_DBFS_CONTENT.GETPATHBYMOUNTID (  
    store_mount    IN     VARCHAR2,  
    guid           IN     INTEGER)  
RETURN VARCHAR2;
```

Parameters

Table 48-40  GETPATHBYMOUNTID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_mount</td>
<td>Mount point in which the path item with guid resides</td>
</tr>
<tr>
<td>guid</td>
<td>Unique ID for the path item</td>
</tr>
</tbody>
</table>
Usage Notes

If the GUID is unknown, a NULL value is returned. Clients are expected to handle this as appropriate.

Return Values

Path of the path item represented by GUID in store_mount

48.7.23 GETPATHBYSTOREID Function

If the underlying GUID is found in the underlying store, this function returns the store-qualified path name.

Syntax

```sql
DBMS_DBFS_CONTENT.GETPATHBYSTOREID (  
    store_name        IN      VARCHAR2,  
    guid              IN      INTEGER)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>guid</td>
<td>Unique ID representing the desired path item</td>
</tr>
</tbody>
</table>

Usage Notes

If the GUID is unknown, a NULL value is returned. Clients are expected to handle this as appropriate.

Return Values

Store-qualified path name represented by the GUID

48.7.24 GETPATHNOWAIT Procedures

This procedure implies that the operation is for an update, and, if implemented, allows providers to return an exception (ORA-00054) rather than wait for row locks.

See FEATURE_NOWAIT in Table 48-5 for more information.

Syntax

```sql
DBMS_DBFS_CONTENT.GETPATHNOWAIT (  
    path        IN              VARCHAR2,  
    properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
    content     OUT    NOCOPY   BLOB,  
    item_type   OUT             INTEGER,  
    prop_flags  IN              INTEGER     DEFAULT (PROP_STD + PROP_OPT + PROP_DATA),
```
Parameters

Table 48-42  GETPATHNOWAIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to path items</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be returned depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>content</td>
<td>BLOB holding data which populates the file (optional)</td>
</tr>
<tr>
<td>item_type</td>
<td>Type of the path item specified (see Table 48-4)</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are returned. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
<tr>
<td>deref</td>
<td>If nonzero, attempts to resolve the given path item to actual data provided it is a reference</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

48.7.25 GETSTOREB YMOUNT Function

This function returns a store by way of its name.

Syntax

```sql
DBMS_DBFS_CONTENT.GETSTOREB YMOUNT (  
  store_mount  IN  VARCHAR2)  
RETURN STORE_T;
```
48.7.26 GETSTOREBYPATH Function

This function returns a store by way of its path.

Syntax

```sql
DBMS_DBFS_CONTENT.GETSTOREBYPATH (  
    path IN PATH_T)  
RETURN STORE_T;
```

Parameters

Table 48-45  GETSTOREBYPATH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>PATH_T</td>
</tr>
</tbody>
</table>

Return Values

STORE_T Record Type

48.7.27 GETSTOREBYPATH Function

This function returns a store by way of its path.

Syntax

```sql
DBMS_DBFS_CONTENT.GETSTOREBYPATH (  
    path IN PATH_T)  
RETURN STORE_T;
```

Parameters

Table 48-45  GETSTOREBYPATH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>PATH_T</td>
</tr>
</tbody>
</table>
48.7.28 GETSTATS Procedure

This procedure returns information about DBMS_DBFS_CONTENT statistics collection.

Syntax

```sql
DBMS_DBFS_CONTENT.GETSTATS ( enabled OUT BOOLEAN, flush_time OUT INTEGER, flush_count OUT INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether statistics collection is enabled</td>
</tr>
<tr>
<td>flush_time</td>
<td>How often to flush the statistics to disk in centiseconds</td>
</tr>
<tr>
<td>flush_count</td>
<td>Number of operations to allow between statistics flushes</td>
</tr>
</tbody>
</table>

48.7.29 GETTRACE Function

This function returns whether DBMS_DBFS_CONTENT tracing is turned on or not.

Syntax

```sql
DBMS_DBFS_CONTENT.GETTRACE RETURN INTEGER.
```

Return Values

Returns zero if tracing is off, non-zero if tracing is on.

48.7.30 GETVERSION Function

This function marks each version of the DBMS_DBFS_CONTENT interface.

Syntax

```sql
DBMS_DBFS_CONTENT.GETVERSION ( RETURN VARCHAR2;
```

Return Values

A string enumerating the version of the DBMS_DBFS_CONTENT interface in standard naming convention: string: a.b.c corresponding to major, minor, and patch components.
48.7.31 ISPATHLOCKED Procedure

This procedure checks if any user-level locks are applied on a given path.

Syntax

```
DBMS_DBFS_CONTENT.ISPATHLOCKED (  
    path           IN     VARCHAR2,  
    who            IN     VARCHAR2,  
    lock_type      IN OUT INTEGER,  
    store_name     IN     VARCHAR2 DEFAULT NULL,  
    principal      IN     VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>who</td>
<td>Transaction identifier that has locked the path</td>
</tr>
<tr>
<td>lock_type</td>
<td>One of the available lock types (see Table 48-6)</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

48.7.32 LIST Function

This function lists the path items in the specified path meeting the specified filter and other criteria.

Syntax

```
DBMS_DBFS_CONTENT.LIST (  
    path          IN     VARCHAR2,  
    filter        IN     VARCHAR2 DEFAULT NULL,  
    recurse       IN     INTEGER DEFAULT 0,  
    asof          IN     TIMESTAMP DEFAULT NULL,  
    store_name    IN     VARCHAR2 DEFAULT NULL,  
    principal     IN     VARCHAR2 DEFAULT NULL)  
RETURN  DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to directories</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively. Otherwise, recursively list the contents of directories and files below the given directory.</td>
</tr>
</tbody>
</table>

Table 48-47 ISPATHLOCKED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>who</td>
<td>Transaction identifier that has locked the path</td>
</tr>
<tr>
<td>lock_type</td>
<td>One of the available lock types (see Table 48-6)</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

Table 48-48 LIST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to directories</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively. Otherwise, recursively list the contents of directories and files below the given directory.</td>
</tr>
</tbody>
</table>

Chapter 48

Summary of DBMS_DBFS_CONTENT Subprograms
Table 48-48  (Cont.) LIST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of repository</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

Return Values

DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type

48.7.33 LISTALLPROPERTIES Function

This function returns a table of all properties for all path items in all mounts.

Syntax

```
DBMS_DBFS_CONTENT.LISTALLPROPERTIES
   RETURN  PROP_ITEMS_T PIPELINED;
```

Return Values

PROP_ITEMS_T Table Type

48.7.34 LISTALLCONTENT Function

This function lists all path items in all mounts.

Syntax

```
DBMS_DBFS_CONTENT.LISTALLCONTENT
   RETURN  PATH_ITEMS_T PIPELINED;
```

Return Values

PATH_ITEMS_T Table Type

48.7.35 LISTMOUNTS Function

This function lists all available mount points, their backing stores, and the store features.

Syntax

```
DBMS_DBFS_CONTENT.LISTMOUNTS
   RETURN MOUNTS_T PIPELINED;
```

Return Values

MOUNTS_T Table Type
Usage Notes

A single mount results in a single returned row, with its `store_mount` field of the returned records set to `NULL`.

48.7.36 LISTSTORES Function

This function lists all available stores and their features.

Syntax

```
DBMS_DBFS_CONTENT.LISTSTORES
   RETURN STORES_T PIPELINED;
```

Return Values

`STORES_T` Table Type

Usage Notes

The `store_mount` field of the returned records is set to `NULL` (since mount-points are separate from stores themselves).

48.7.37 LOCKPATH Procedure

This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.

Syntax

```
DBMS_DBFS_CONTENT.LOCKPATH (  
   path        IN     VARCHAR2,  
   who         IN     VARCHAR2,  
   lock_type   IN     INTEGER,  
   waitForRowLock IN     INTEGER     DEFAULT 1,  
   store_name  IN     VARCHAR2    DEFAULT NULL,  
   principal   IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 48-49  LOCKPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>who</td>
<td>Transaction identifier that has locked the path</td>
</tr>
<tr>
<td>lock_type</td>
<td>One of the available lock types (see Table 48-6)</td>
</tr>
<tr>
<td>waitForRowLock</td>
<td>Determines if a row is locked by a transaction or not</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>
48.7.38 MOUNTSTORE Procedure

This procedure mounts a previously registered store and binds it to the mount point.

Syntax

```sql
DBMS_DBFS_CONTENT.MOUNTSTORE (
    store_mount    in      VARCHAR2   DEFAULT NULL,
    singleton      in      BOOLEAN    DEFAULT FALSE,
    principal      in      VARCHAR2   DEFAULT NULL,
    owner          in      VARCHAR2   DEFAULT NULL,
    acl            in      VARCHAR2   DEFAULT NULL,
    asof           in      TIMESTAMP  DEFAULT NULL,
    read_only      in      BOOLEAN    DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_mount</td>
<td>Path name to use to mount this store</td>
</tr>
<tr>
<td>singleton</td>
<td>Whether the mount is a single backend store on the system</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
<tr>
<td>owner</td>
<td>Owner for new elements created (implicitly or explicitly) by the current operation</td>
</tr>
<tr>
<td>acl</td>
<td>ACL for all new elements created (implicitly or explicitly) by the current operation</td>
</tr>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
<tr>
<td>read_only</td>
<td>Whether the mount is read-only</td>
</tr>
</tbody>
</table>

Usage Notes

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for information on mounting a registered store

48.7.39 NORMALIZEPATH Functions

This function converts a store-specific or full-absolute path name into normalized form.

It does the following:

- verifies that the path name is absolute, and so starts with "/"
- collapses multiple consecutive "/" into a single "/"
- strips trailing "/"
• breaks up a store-specific normalized path name into 2 components - parent
  pathname, trailing component name
• breaks up a full-absolute normalized path name into 3 components - store name,
  parent pathname, trailing component name

Syntax

```sql
DBMS_DBFS_CONTENT.NORMALIZEPATH (
    path        IN              VARCHAR2,
    parent      OUT NOCOPY      VARCHAR2,
    tpath       OUT NOCOPY      VARCHAR2)
RETURN VARCHAR2;
```

```sql
DBMS_DBFS_CONTENT.NORMALIZEPATH (
    path        IN              VARCHAR2,
    store_name  OUT NOCOPY      VARCHAR2,
    parent      OUT NOCOPY      VARCHAR2,
    tpath       OUT NOCOPY      VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table 48-51 NORMALIZEPATH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>parent</td>
<td>Parent path name</td>
</tr>
<tr>
<td>tpath</td>
<td>Name of trailing path item</td>
</tr>
</tbody>
</table>

Return Values

The completely normalized store-specific or full-absolute path name

48.7.40 PROPANY Functions

This function provides constructors that take one of a variety of types and return a
PROPERTY_T.

Syntax

```sql
DBMS_DBFS_CONTENT.PROPANY (val IN NUMBER)
RETURN PROPERTY_T;
```

```sql
DBMS_DBFS_CONTENT.PROPANY (val IN VARCHAR2)
RETURN PROPERTY_T;
```

```sql
DBMS_DBFS_CONTENT.PROPANY (val IN TIMESTAMP)
RETURN PROPERTY_T;
```

```sql
DBMS_DBFS_CONTENT.PROPANY (val IN NUMBER)
RETURN PROPERTY_T;
```
48.7.41 PROPERTIESH2T Function

This function converts a PROPERTY_T hash to a DBMS_DBFS_CONTENT_PROPERTIES_T table.

Syntax

DBMS_DBFS_CONTENT.PROPERTIESH2T (    pprops      IN      PROPERTIES_T)    RETURN DBMS_DBFS_CONTENT_PROPERTIES_T;

Parameters

Table 48-53  PROPERTIEST2H Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sprops</td>
<td>A PROPERTIES_T hash</td>
</tr>
</tbody>
</table>

Return Values

DBMS_DBFS_CONTENT_PROPERTIES_T Table Type

48.7.42 PROPERTIEST2H Function

This function converts a DBMS_DBFS_CONTENT_PROPERTIES_T table to a PROPERTY_T hash.

Syntax

DBMS_DBFS_CONTENT.PROPERTIEST2H (    sprops      IN      DBMS_DBFS_CONTENT_PROPERTIES_T)    RETURN properties_t;

Parameters

Table 48-54  PROPERTIEST2H Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sprops</td>
<td>A DBMS_DBFS_CONTENT_PROPERTIES_T table</td>
</tr>
</tbody>
</table>
48.7.43 PROPNUMBER Function

This function is a constructor that takes a number and returns a PROPERTY_T.

Syntax

```sql
DBMS_DBFS_CONTENT.PROPNUMBER (
    val IN NUMBER)
RETURN PROPERTY_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Value</td>
</tr>
</tbody>
</table>

48.7.44 PROPRAW Function

This function is a constructor that takes a RAW and returns a PROPERTY_T.

Syntax

```sql
DBMS_DBFS_CONTENT.PROPRAW (
    val IN RAW)
RETURN PROPERTY_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Value</td>
</tr>
</tbody>
</table>

Return Values

PROPERTY_T Record Type
48.7.45 PROPTIMESTAMP Function

This function is a constructor that takes a TIMESTAMP and returns a PROPERTY_T.

Syntax

```sql
DBMS_DBFS_CONTENT.PROPTIMESTAMP (val IN TIMESTAMP) RETURN PROPERTY_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Value</td>
</tr>
</tbody>
</table>

Return Values

PROPERTY_T Record Type

48.7.46 PROPVARCHAR2 Function

This function is a constructor that takes a VARCHAR2 and returns a PROPERTY_T.

Syntax

```sql
DBMS_DBFS_CONTENT.PROPVARCHAR2 (val IN VARCHAR2) RETURN PROPERTY_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Value</td>
</tr>
</tbody>
</table>

Return Values

PROPERTY_T Record Type

48.7.47 PURGEALL Procedure

This procedure purges all soft-deleted entries matching the path and optional filter criteria.

Syntax

```sql
DBMS_DBFS_CONTENT.PURGEALL (path IN VARCHAR2,
filter IN VARCHAR2 DEFAULT NULL,
store_name IN VARCHAR2 DEFAULT NULL,
principal IN VARCHAR2 DEFAULT NULL);
```
### Parameters

Table 48-59  **PURGEALL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied based on specified criteria</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

### 48.7.48 PURGEPATH Procedure

This procedure purges any soft-deleted versions of the given path item.

**Syntax**

```sql
DBMS_DBFS_CONTENT.PURGEPATH (    path IN VARCHAR2,    filter IN VARCHAR2 DEFAULT NULL,    store_name IN VARCHAR2 DEFAULT NULL,    principal IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 48-60  **PURGEPATH Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

### 48.7.49 PUTPATH Procedures

This procedure creates a new path item.

**Syntax**

```sql
DBMS_DBFS_CONTENT.PUTPATH (    path IN VARCHAR2,    properties IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,    content IN OUT NOCOPY BLOB,    item_type OUT INTEGER,    prop_flags IN INTEGER DEFAULT (PROP_STD + PROP_OPT + PROP_DATA),    store_name IN VARCHAR2 DEFAULT NULL,    principal IN VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_DBFS_CONTENT.PUTPATH (    path IN VARCHAR2,    properties IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,    content IN OUT NOCOPY BLOB,    item_type OUT INTEGER,    prop_flags IN INTEGER DEFAULT (PROP_STD + PROP_OPT + PROP_DATA),    store_name IN VARCHAR2 DEFAULT NULL,    principal IN VARCHAR2 DEFAULT NULL);
```
properties IN OUT NOCOPY PROPERTIES_T,
content IN OUT NOCOPY BLOB,
item_type OUT INTEGER,
prop_flags IN INTEGER DEFAULT (PROP_STD + 
PROP_OPT + 
PROP_DATA),
store_name IN VARCHAR2 DEFAULT NULL,
principal IN VARCHAR2 DEFAULT NULL);

DBMS_DBFS_CONTENT.PUTPATH (path IN VARCHAR2,
properties IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
amount IN NUMBER,
offset IN NUMBER,
buffer IN RAW,
prop_flags IN INTEGER DEFAULT (PROP_STD + 
PROP_OPT),
store_name IN VARCHAR2 DEFAULT NULL,
principal IN VARCHAR2 DEFAULT NULL);

Parameters

Table 48-61 PUTPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
</tbody>
</table>
**Table 48-61  (Cont.) PUTPATH Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set depending on prop_flags (see DBMS_DBFSCONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>content</td>
<td>BLOB holding data which populates the file (optional)</td>
</tr>
<tr>
<td>item_type</td>
<td>Type of the path item specified (see Table 48-4)</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes to be read</td>
</tr>
<tr>
<td>offset</td>
<td>Byte offset from which to begin reading</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer to which to write</td>
</tr>
<tr>
<td>buffers</td>
<td>Buffers to which to write</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFSCONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

### 48.7.50 REGISTERSTORE Procedure

This procedure registers a new store backed by a provider that uses a store provider (conforming to the DBMS_DBFSCONTENT_SPI package signature).

This method is to be used primarily by store providers after they have created a new store.

**Syntax**

```sql
DBMS_DBFSCONTENT.REGISTERSTORE (  
  store_name          IN      VARCHAR2,  
  provider_name       IN      VARCHAR2,  
  provider_package    IN      VARCHAR2);  
```

**Parameters**

**Table 48-62  REGISTERSTORE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store, must be unique</td>
</tr>
<tr>
<td>provider_name</td>
<td>Name of provider</td>
</tr>
<tr>
<td>provider_package</td>
<td>Store provider</td>
</tr>
</tbody>
</table>
48.7.51 RENAMEPATH Procedures

This procedure renames or moves a path. This operation can be performed across directory hierarchies and mount-points as long as it is within the same store.

Note:

See Oracle Database SecureFiles and Large Objects Developer’s Guide for Rename and Move operations

Syntax

```sql
DBMS_DBFS_CONTENT.RENAMEPATH (
    oldPath     IN              VARCHAR2,
    newPath     IN              VARCHAR2,
    properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
    store_name  IN              VARCHAR2    DEFAULT NULL,
    principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldPath</td>
<td>Name of path prior to renaming</td>
</tr>
<tr>
<td>newPath</td>
<td>Name of path after renaming</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store, must be unique</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

48.7.52 RESTOREALL Procedure

This procedure restores all soft-deleted path items meeting the path and optional filter criteria.

Syntax

```sql
DBMS_DBFS_CONTENT.RESTOREALL (
    path          IN      VARCHAR2,
    filter        IN      VARCHAR2    DEFAULT NULL,
```
48.7.53 RESTOREPATH Procedure

This procedure restores all soft-deleted path items that match the given path and optional filter criteria.

Syntax

```sql
DBMS_DBFS_CONTENT.RESTOREPATH (
  path           IN      VARCHAR2,
  filter         IN      VARCHAR2    DEFAULT NULL,
  store_name     IN      VARCHAR2    DEFAULT NULL,
  principal      IN      VARCHAR2    DEFAULT NULL);
```

Parameters

Table 48-65  RESTOREPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

48.7.54 SETDEFAULTACL Procedure

This procedure sets the ACL parameter of the default context.

This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```sql
DBMS_DBFS_CONTENT.SETDEFAULTACL (  
  acl    IN     VARCHAR2);
```
48.7.55 SETDEFAULTASOF Procedure

This procedure sets the "as of" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```sql
DBMS_DBFS_CONTENT.SETDEFAULTASOF (  
  asof    IN     TIMESTAMP);
```

Parameters

Table 48-67  SETDEFAULTASOF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
</tbody>
</table>

Usage Notes

- NULL by default, this parameter be can be cleared by setting it to NULL.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

48.7.56 SETDEFAULTCONTEXT Procedure

This procedure sets the default context. The information contained in the context can be inserted explicitly by way of arguments to the various method calls, allowing for fine-grained control over individual operations.

Syntax

```sql
DBMS_DBFS_CONTENT.SETDEFAULTCONTEXT (  
  principal    IN     VARCHAR2,  
  owner        IN     VARCHAR2,  
  acl          IN     VARCHAR2,  
  asof         IN     TIMESTAMP);
```
Parameters

Table 48-68  SETDEFAULTCONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
<tr>
<td>owner</td>
<td>Owner for new elements created (implicitly or explicitly) by the current operation</td>
</tr>
<tr>
<td>acl</td>
<td>ACL for all new elements created (implicitly or explicitly) by the current operation</td>
</tr>
<tr>
<td>asof</td>
<td>The &quot;as of&quot; timestamp at which the underlying read-only operation (or its read-only sub-components) executes</td>
</tr>
</tbody>
</table>

Usage Notes

- All of the context parameters are NULL by default, and be can be cleared by setting them to NULL.
- The context parameters, once set, remain as defaults for the duration of the session, and are inherited by all operations for which the defaults are not explicitly overridden.

48.7.57 SETDEFAULTOWNER Procedure

This procedure sets the "owner" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```sql
DBMS_DBFS_CONTENT.SETDEFAULTOWNER (principal    IN     VARCHAR2);
```

Parameters

Table 48-69  SETDEFAULTOWNER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner for new elements created (implicitly or explicitly) by the current operation</td>
</tr>
</tbody>
</table>

Usage Notes

- NULL by default, this parameter be can be cleared by setting it to NULL.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.
48.7.58 SETDEFAULTPRINCIPAL Procedure

This procedure sets the "principal" parameter of the default context. This information contained can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

DBMS_DBFS_CONTENT.SETDEFAULTPRINCIPAL ( principal IN VARCHAR2);

Parameters

Table 48-70 SETDEFAULTPRINCIPAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

Usage Notes

• NULL by default, this parameter be can be cleared by setting it to NULL.
• The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

48.7.59 SETPATH Procedures

This procedure assigns a path name to a path item represented by contentID.

Stores and their providers that support contentID-based access and lazy path name binding also support the SETPATH Procedure that associates an existing contentID with a new path.

Note:

See Oracle Database SecureFiles and Large Objects Developer’s Guide for Rename and Move operations

Syntax

DBMS_DBFS_CONTENT.SETPATH ( store_name IN VARCHAR2, contentID IN RAW, path IN VARCHAR2, properties IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T, principal IN VARCHAR2 DEFAULT NULL);

DBMS_DBFS_CONTENT.SETPATH ( store_name IN VARCHAR2, contentID IN RAW, path IN VARCHAR2, properties IN OUT NOCOPY PROPERTIES_T, principal IN VARCHAR2 DEFAULT NULL);
48.7.60 SETSTATS Procedure

This procedure enables and disables statistics collection.

The client can optionally control the flush settings by specifying non-NULL values for the time, count or both parameters.

**Syntax**

```sql
DBMS_DBFS_CONTENT.SETSTATS (enable IN BOOLEAN, flush_time IN INTEGER, flush_count IN INTEGER);
```

### Parameters

**Table 48-72  SETSTATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>If TRUE, enable statistics collection. If FALSE, disable statistics collection.</td>
</tr>
<tr>
<td>flush_time</td>
<td>How often to flush the statistics to disk in centiseconds</td>
</tr>
<tr>
<td>flush_count</td>
<td>Number of operations to allow between statistics flushes</td>
</tr>
</tbody>
</table>

**Usage Notes**

The `SETSTATS` Procedure buffers statistics in-memory for a maximum of `flush_time` centiseconds or a maximum of `flush_count` operations (whichever limit is reached first), or both, at which time the buffers are implicitly flushed to disk.
48.7.61 SETTRACE Procedure

This procedure sets the DBMS_DBFS_CONTENT tracing severity to the given level, 0 being "off".

Syntax

```
DBMS_DBFS_CONTENT.SETTRACE
    trclvl IN INTEGER);
```

Parameters

Table 48-73  SETTRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trclvl</td>
<td>Level of the tracing, higher values implying more tracing</td>
</tr>
</tbody>
</table>

48.7.62 SPACEUSAGE Procedure

This procedure queries file system space usage statistics.

Providers are expected to support this subprogram for their stores (and to make a best effort determination of space usage, especially if the store consists of multiple tables, indexes, LOBs, and so on).

Syntax

```
DBMS_DBFS_CONTENT.SPACEUSAGE (    
    path          IN        VARCHAR2,
    blksize       OUT       INTEGER,
    tbytes        OUT       INTEGER,
    fbytes        OUT       INTEGER,
    nfile         OUT       INTEGER,
    ndir          OUT       INTEGER,
    nlink         OUT       INTEGER,
    nref          OUT       INTEGER,
    store_name    IN        VARCHAR2 DEFAULT NULL);
```

Parameters

Table 48-74  SPACEUSAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>blksize</td>
<td>Natural tablespace blocksize that holds the store. If multiple tablespaces with different blocksizes are used, any valid blocksize is acceptable.</td>
</tr>
<tr>
<td>tbytes</td>
<td>Total size of the store in bytes computed over all segments that comprise the store</td>
</tr>
<tr>
<td>fbytes</td>
<td>Free or unused size of the store in bytes computed over all segments that comprise the store</td>
</tr>
<tr>
<td>nfile</td>
<td>Number of currently available files in the store</td>
</tr>
</tbody>
</table>
Table 48-74  (Cont.) SPACEUSAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndir</td>
<td>Number of currently available directories in the store</td>
</tr>
<tr>
<td>nlink</td>
<td>Number of currently available links in the store</td>
</tr>
<tr>
<td>nref</td>
<td>Number of currently available references in the store</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Usage Notes

- A space usage query on the top-level root directory returns a combined summary of the space usage of all available distinct stores under it (if the same store is mounted multiple times, is still counted only once).
- Since database objects are dynamically expandable, it is not easy to estimate the division between “free” space and “used” space.

### 48.7.63 TRACE Procedure

This procedure outputs tracing to the current foreground trace file.

**Syntax**

```sql
DBMS_DBFS_CONTENT.TRACE
sev    IN    INTEGER,
msg0   IN    VARCHAR2,
msg1   IN    VARCHAR  DEFAULT '',
msg2   IN    VARCHAR  DEFAULT '',
msg3   IN    VARCHAR  DEFAULT '',
msg4   IN    VARCHAR  DEFAULT '',
msg5   IN    VARCHAR  DEFAULT '',
msg6   IN    VARCHAR  DEFAULT '',
msg7   IN    VARCHAR  DEFAULT '',
msg8   IN    VARCHAR  DEFAULT '',
msg9   IN    VARCHAR  DEFAULT '',
msg10  IN    VARCHAR  DEFAULT '');
```

**Parameters**

Table 48-75  TRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sev</td>
<td>Severity at which trace message is output</td>
</tr>
<tr>
<td>msg*</td>
<td>One or more message strings to be output. If more than one message is specified, all are output.</td>
</tr>
</tbody>
</table>

Usage Notes

- Trace information is written to the foreground trace file, with varying levels of detail as specified by the trace level arguments.
- The global trace level consists of 2 components: “severity” and “detail”. These can be thought of as additive bitmasks.
The "severity" allows the separation of top level as compared to low-level tracing of different components, and allows the amount of tracing to be increased as needed. There are no semantics associated with different levels, and users are free to set trace at any severity they choose, although a good rule of thumb would use severity "1" for top level API entry and exit traces, "2" for internal operations, and "3" or greater for very low-level traces.

The "detail" controls how much additional information: timestamps, short-stack, etc. is dumped along with each trace record.

48.7.64 TRACEENABLED Function

This function determines if the current trace "severity" set by the SETTRACE Procedure is at least as high as the given trace level.

Syntax

```sql
DBMS_DBFS_CONTENT.TRACEENABLED(
    sev IN INTEGER)
RETURN INTEGER;
```

Parameters

Table 48-76  TRACEENABLED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sev</td>
<td>Severity at which trace message is output</td>
</tr>
</tbody>
</table>

Return Values

Returns 0 if the requested severity level is lower than the currently set trace severity level; 1 otherwise.

Related Topics

- **SETTRACE Procedure**
  This procedure sets the `DBMS_DBFS_CONTENT` tracing severity to the given level, 0 being "off".

48.7.65 UNLOCKPATH Procedure

This procedure unlocks path items that were previously locked with the LOCKPATH Procedure.

Syntax

```sql
DBMS_DBFS_CONTENT.UNLOCKPATH (
    path IN VARCHAR2,
    who IN VARCHAR2,
    waitForRowLock IN INTEGER DEFAULT 1,
    store_name IN VARCHAR2 DEFAULT NULL,
    principal IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 48-77 UNLOCKPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>who</td>
<td>Transaction identifier that has locked the path</td>
</tr>
<tr>
<td>waitForRowLock</td>
<td>Determines if a row is locked by a transaction or not</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>principal</td>
<td>Agent (principal) invoking the current operation</td>
</tr>
</tbody>
</table>

Related Topics

- LOCKPATH Procedure
  This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.

48.7.66 UNMOUNTSTORE Procedure

This procedure unmounts a registered store, either by name or by mount point.

Syntax

```sql
DBMS_DBFS_CONTENT.UNMOUNTSTORE (
    store_name       IN      VARCHAR2   DEFAULT NULL,
    store_mount      IN      VARCHAR2   DEFAULT NULL,
    ignore_unknown   IN      BOOLEAN    DEFAULT FALSE);
```

Parameters

Table 48-78 UNMOUNTSTORE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>store_mount</td>
<td>Location at which the store instance is mounted</td>
</tr>
<tr>
<td>ignore_unknown</td>
<td>If TRUE, attempts to unregister unknown stores will not raise an exception.</td>
</tr>
</tbody>
</table>

Usage Notes

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for further information on unmounting a previously unmounted store
48.7.67 UNREGISTERSTORE Procedure

This procedure unregisters a previously registered store (invalidating all mount points associated with it).

Syntax

```sql
DBMS_DBFS_CONTENT.UNREGISTERSTORE (  
    store_name          IN      VARCHAR2,
    ignore_unknown      IN      BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>ignore_unknown</td>
<td>If TRUE, attempts to unregister unknown stores will not raise an exception.</td>
</tr>
</tbody>
</table>

Usage Notes

- Once unregistered all access to the store (and its mount points) are not guaranteed to work
- If the `ignore_unknown` argument is TRUE, attempts to unregister unknown stores do not raise an exception.
The `DBMS_DBFS_CONTENT_SPI` package is a specification for `DBMS_DBFS_CONTENT` store providers, which must be implemented. Application designers can create PL/SQL packages conforming to this specification to extend `DBMS_DBFS_CONTENT` to use custom store providers.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Operational Notes**
- **Summary of `DBMS_DBFS_CONTENT_SPI` Subprograms**

**Related Topics**

- **`DBMS_DBFS_CONTENT`**
  The `DBMS_DBFS_CONTENT` package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.

**See Also:**

- *Oracle Database SecureFiles and Large Objects Developer's Guide*

### 49.1 ODBMS_DBFS_CONTENT_SPI Overview

The `DBMS_DBFS_CONTENT_SPI` package describes an internal contract between the implementation of the `DBMS_DBFS_CONTENT` interface and individual store providers, and whichever package contains their code.

Since PL/SQL does not allow a compile-time, declarative type-conformation between package signatures, store providers should informally conform to the SPI, which is to say, they should implement the SPI by means of a package that contains all of the methods specified in package `DBMS_DBFS_CONTENT_SPI`, with the same method signatures and semantics.

Obviously, these provider packages can implement other methods and expose other interfaces, however, these interfaces are not to be used by the `DBMS_DBFS_CONTENT` interface itself.

Since the provider SPI is merely a contract specification, there is no package body for `DBMS_DBFS_CONTENT_SPI`, and it is not possible to actually invoke any methods using this package.

The SPI references various elements (constants, types, exceptions) defined by the `DBMS_DBFS_CONTENT` interface.
Additionally, there is an almost one-to-one correspondence between the client API exported by the DBMS_DBFS_CONTENT interface and the provider interface that the DBMS_DBFS_CONTENT interface itself expects to work against.

The main distinction in the method naming conventions is that all path name references are always store-qualified. That is, the notion of mount-points and full-absolute path names have been normalized and converted to store-qualified path names by the DBMS_DBFS_CONTENT interface before it invokes any of the provider SPI methods.

Since the interconnection of the DBMS_DBFS_CONTENT interface and the provider SPI is a 1-to-many pluggable architecture, and the interface uses dynamic SQL to invoke methods in the provider SPI, this can lead to runtime errors.

Related Topics
- DBMS_DBFS_CONTENT

The DBMS_DBFS_CONTENT package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.

49.2 DBMS_DBFS_CONTENT_SPI Security Model

Implementations of the DBMS_DBFS_CONTENT_SPI package should be created as AUTHID CURRENT_USER.

49.3 DBMS_DBFS_CONTENT_SPI Operational Notes

This topic lists operational notes for DBMS_DBFS_CONTENT_SPI implementation, path names, and other operations.

- Implementation
- Path Names
- Other DBMS_DBFS_CONTENT Operations

Implementation

Since the interconnection of the DBMS_DBFS_CONTENT interface and the provider SPI is a 1-to-many pluggable architecture, the interface uses dynamic SQL to invoke methods in the provider SPI, this can lead to runtime errors.

There are no explicit INIT or FINI methods to indicate when the DBMS_DBFS_CONTENT interface plugs or unplugs a particular provider SPI. Provider SPIs must be willing to auto-initialize themselves at any SPI entry-point.

All operations performed by a store provider are "stateless" in that they are complete operations unto themselves. If state is necessary to be maintained for some reason, then the state must be maintained in data structures such as auxiliary tables that can be queried as needed.

Path Names

All path names used in the provider SPI are store-qualified in pair form (store_name, pathname) where the path name is rooted within the store namespace.
Stores and their providers that support contentID-based access (see `FEATURE_CONTENT_ID` in Table 48-5) also support a form of addressing that is not based on path names. Content items are identified by an explicit store name, a NULL path name, and possibly a contentID specified as a parameter or by way of the `OPT_CONTENT_ID` (see Table 48-8) property.

Not all operations are supported with contentID-based access, and applications should depend only on the simplest create or delete functionality being available.

**Other DBMS_DBFS_CONTENT Operations**

This table lists other operations and provides links to related discussions.

<table>
<thead>
<tr>
<th>Other Operations</th>
<th>See ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on creation operations</td>
</tr>
<tr>
<td>Deletion</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on deletion operations</td>
</tr>
<tr>
<td>Get (Retrieve) and Put (Insert)</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Get and Put operations</td>
</tr>
<tr>
<td>Rename and Move</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Rename and Move operations</td>
</tr>
<tr>
<td>Directory Navigation and Search</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Navigation and Search operations</td>
</tr>
<tr>
<td>Locking</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Locking operations</td>
</tr>
<tr>
<td>Access Check</td>
<td>Oracle Database SecureFiles and Large Objects Developer’s Guide for further information on Access Check operations</td>
</tr>
</tbody>
</table>

**49.4 Summary of DBMS_DBFS_CONTENT_SPI Subprograms**

This table lists and describes the subprograms used in the `DBMS_DBFS_CONTENT_SPI` Package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKACCESS Function</td>
<td>Reports if the user (principal) can perform the specified operation on the given path</td>
</tr>
<tr>
<td>CREATEDIRECTORY Procedure</td>
<td>Creates a directory</td>
</tr>
<tr>
<td>CREATEFILE Procedure</td>
<td>Creates a file</td>
</tr>
<tr>
<td>CREATELINK Procedure</td>
<td>Creates a physical link to an already existing file system element</td>
</tr>
<tr>
<td>CREATREFERENCE Procedure</td>
<td>Creates a new reference to the source file system element</td>
</tr>
<tr>
<td>DELETECONTENT Procedure</td>
<td>Deletes the file specified by the given contentID</td>
</tr>
<tr>
<td>DELETEDIRECTORY Procedure</td>
<td>Deletes a directory</td>
</tr>
<tr>
<td>DELETEFILE Procedure</td>
<td>Deletes a file</td>
</tr>
<tr>
<td>GETFEATURES Function</td>
<td>Returns the features of a store</td>
</tr>
<tr>
<td>GETPATH Procedures</td>
<td>Returns existing path items (such as files and directories)</td>
</tr>
</tbody>
</table>
### Table 49-2  (Cont.) DBMS_DBFS_CONTENT_SPI Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPATHBSTOREID Function</td>
<td>If the underlying GUID is found in the underlying store, returns the store-qualified path name</td>
</tr>
<tr>
<td>GETPATHNOWAIT Procedure</td>
<td>Implies that the operation is for an update, and, if implemented, allows providers to return an exception (ORA-00054) rather than wait for row locks.</td>
</tr>
<tr>
<td>GETSTOREID Function</td>
<td>Returns the ID of a store</td>
</tr>
<tr>
<td>GETVERSION Function</td>
<td>Returns the version associated with a store</td>
</tr>
<tr>
<td>LIST Function</td>
<td>Lists the contents of a directory path name</td>
</tr>
<tr>
<td>LOCKPATH Procedure</td>
<td>Applies user-level locks to the given valid path name</td>
</tr>
<tr>
<td>PURGEALL Procedure</td>
<td>Purges all soft-deleted entries matching the path and optional filter criteria</td>
</tr>
<tr>
<td>PURGEPATH Procedure</td>
<td>Purges any soft-deleted versions of the given path item</td>
</tr>
<tr>
<td>PUTPATH Procedures</td>
<td>Creates a new path item</td>
</tr>
<tr>
<td>RENAMEPATH Procedure</td>
<td>Renames or moves a path</td>
</tr>
<tr>
<td>RESTOREALL Procedure</td>
<td>Restores all soft-deleted path items meeting the path and filter criteria</td>
</tr>
<tr>
<td>RESTOREPATH Procedure</td>
<td>Restores all soft-deleted path items that match the given path and filter criteria</td>
</tr>
<tr>
<td>SEARCH Function</td>
<td>Searches for path items matching the given path and filter criteria</td>
</tr>
<tr>
<td>SETPATH Procedure</td>
<td>Assigns a path name to a path item represented by contentID</td>
</tr>
<tr>
<td>SPACEUSAGE Procedure</td>
<td>Queries file system space usage statistics</td>
</tr>
<tr>
<td>UNLOCKPATH Procedure</td>
<td>Unlocks path items that were previously locked with the LOCKPATH Procedure</td>
</tr>
</tbody>
</table>

#### 49.4.1 CHECKACCESS Function

This function reports if the user (principal) can perform the specified operation on the given path. This enables verifying the validity of an operation without attempting to perform the operation. If `CHECKACCESS` returns 0, then the subprogram invoked to implement that operation should fail with an error.

**Syntax**

```sql
DBMS_DBFS_CONTENT_SPI.CHECKACCESS (  
  store_name     IN     VARCHAR2  DEFAULT NULL,  
  path           IN     VARCHAR2,  
  pathtype       IN     INTEGER,  
  operation      IN     VARCHAR2,  
  principal      IN     VARCHAR2)  
RETURN  INTEGER;
```
### Parameters

#### Table 49-3  CHECKACCESS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to check for access</td>
</tr>
<tr>
<td>path_type</td>
<td>Type of object path represents (see Table 48-4)</td>
</tr>
<tr>
<td>operation</td>
<td>Operation to be checked (see Table 48-8)</td>
</tr>
<tr>
<td>principal</td>
<td>File system user for whom the access check is made</td>
</tr>
</tbody>
</table>

#### Usage Notes

Whether or not the user invokes this function, a store that supports access control internally performs these checks to guarantee security.

### 49.4.2 CREATEDIRECTORY Procedure

This procedure creates a directory.

#### Syntax

```sql
DBMS_DBFS_CONTENT_SPI.CREATEDIRECTORY (
    store_name  IN              VARCHAR2,
    path        IN              VARCHAR2,
    properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
    prop_flags  IN              INTEGER,
    recurse     IN              INTEGER,
    ctx         IN              DBMS_DBFS_CONTENTCONTEXT_T);
```

#### Parameters

#### Table 49-4  CREATEDIRECTORY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to the directory</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned, or both, depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting PROP_SPC (see Table 48-9), and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively; otherwise, recursively create the directories above the given directory</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to create the directory (see DBMS_DBFS_CONTENTCONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>
49.4.3 CREATEFILE Procedure

This procedure creates a file.

**Syntax**

```sql
DBMS_DBFS_CONTENT_SPI.CREATEFILE (
    store_name    IN              VARCHAR2,
    path          IN              VARCHAR2,
    properties    IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
    content       IN OUT NOCOPY   BLOB,
    prop_flags    IN              INTEGER,
    ctx           IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to the file</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned or both depending, or both on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>content</td>
<td>BLOB holding data with which to populate the file (optional)</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to create the file (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.4 CREATELINK Procedure

This procedure creates a physical link to an already existing file system element (such as file or directory). The resulting entry shares the same metadata structures as the value of the srcPath parameter, and so is similar to incrementing a reference count on the file system element. This is analogous to a UNIX file system hard link.

**Syntax**

```sql
DBMS_DBFS_CONTENT_SPI.CREATELINK (
    store_name    IN              VARCHAR2,
    srcPath       IN              VARCHAR2,
    dstPath       IN              VARCHAR2,
    properties    IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
    prop_flags    IN              INTEGER,
    ctx           IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```
Parameters

Table 49-6  CREATELINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>srcPath</td>
<td>File system entry with which to link</td>
</tr>
<tr>
<td>dstPath</td>
<td>Path of the new link element to be created</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned, or both,</td>
</tr>
<tr>
<td></td>
<td>depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is PROP_STD.</td>
</tr>
<tr>
<td></td>
<td>Specify properties to be returned by setting prop_spec,</td>
</tr>
<tr>
<td></td>
<td>and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</td>
</tr>
<tr>
<td></td>
<td>with properties whose values are of interest.</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to create the link (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.5 CREATEREFERENCE Procedure

This procedure creates a new reference to the source file system element (such as a file, or directory). The resulting reference points to the source element but does not directly share metadata with the source element. This is analogous to a UNIX file system symbolic link.

Syntax

DBMS_DBFS_CONTENT_SPI.CREATEREFERENCE (  
    srcPath       IN              VARCHAR2,  
    dstPath       IN              VARCHAR2,  
    properties    IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
    prop_flags    IN              INTEGER,  
    store_name    IN              VARCHAR2,  
    ctx           IN              DBMS_DBFS_CONTENT_CONTEXT_T);  

Parameters

Table 49-7  CREATEREFERENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>srcPath</td>
<td>File system entry with which to link</td>
</tr>
<tr>
<td>dstPath</td>
<td>Path of the new link element to be created</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set, returned, or both,</td>
</tr>
<tr>
<td></td>
<td>depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
</tbody>
</table>
Table 49-7  (Cont.) CREATREFERENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to create the reference (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.6 DELETECONTENT Procedure

This procedure deletes the file specified by the given contentID.

Syntax

```sql
DBMS_DBFS_CONTENT_SPI.DELETECONTENT (  
    store_name     IN     VARCHAR2,  
    contentID      IN     RAW,  
    filter         IN     VARCHAR2,  
    soft_delete    IN     INTEGER,  
    ctx            IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 49-8  DELETECONTENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>contentID</td>
<td>Unique identifier for the file to be deleted</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>soft_delete</td>
<td>If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete see Oracle Database SecureFiles and Large Objects Developer's Guide, Deletion Operations).</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to delete the file (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.7 DELETEDIRECTORY Procedure

This procedure deletes a directory.

If recurse is nonzero, it recursively deletes all elements of the directory. A filter, if supplied, determines which elements of the directory are deleted.

Syntax

```sql
DBMS_DBFS_CONTENT_SPI.DELETEDIRECTORY (  
    store_name     IN     VARCHAR2,  
    path           IN     VARCHAR2,  
    filter         IN     VARCHAR2,  
    soft_delete    IN     INTEGER,  
    ctx            IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```
49.4.8 DELETEFILE Procedure

This procedure deletes the specified file.

Syntax

DBMS_DBFS_CONTENT_SPI.DELETEFILE (  
    store_name IN VARCHAR2,  
    path IN VARCHAR2,  
    filter IN VARCHAR2,  
    soft_delete IN BOOLEAN,  
    ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);  

Parameters

Table 49-10  DELETEFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to the file</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>soft_delete</td>
<td>If 0, execute a hard (permanent) delete. For any value other than 0,</td>
</tr>
<tr>
<td></td>
<td>perform a soft delete see Oracle Database SecureFiles and Large Objects</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively. Otherwise, recursively delete the</td>
</tr>
<tr>
<td></td>
<td>directories and files below the given directory.</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to delete the file (see DBMS_DBFS_CONTENT_CONTEXT_T</td>
</tr>
<tr>
<td></td>
<td>Object Type)</td>
</tr>
</tbody>
</table>

49.4.8 DELETEFILE Procedure

This procedure deletes the specified file.

Syntax

DBMS_DBFS_CONTENT_SPI.DELETEFILE (  
    store_name IN VARCHAR2,  
    path IN VARCHAR2,  
    filter IN VARCHAR2,  
    soft_delete IN BOOLEAN,  
    ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);  

Parameters

Table 49-10  DELETEFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to the file</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>soft_delete</td>
<td>If 0, execute a hard (permanent) delete. For any value other than 0,</td>
</tr>
<tr>
<td></td>
<td>perform a soft delete see Oracle Database SecureFiles and Large Objects</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to delete the file (see DBMS_DBFS_CONTENT_CONTEXT_T</td>
</tr>
<tr>
<td></td>
<td>Object Type)</td>
</tr>
</tbody>
</table>
49.4.9 GETFEATURES Function

This function returns the features of a store.

Syntax

\[
\text{DBMS\_DBFS\_CONTENT\_SPI.\text{GETFEATURES} (}
\begin{align*}
\text{store\_name} & \quad \text{IN} \quad \text{VARCHAR2} \\
\text{RETURN} & \quad \text{INTEGER};
\end{align*}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Return Values

\[
\text{DBMS\_DBFS\_CONTENT\_FEATURE\_* features supported by the Store Provider}
\]

49.4.10 GETPATH Procedures

This procedure returns existing path items (such as files and directories). This includes both data and metadata (properties).

The client can request (using prop_flags) that specific properties be returned. File path names can be read either by specifying a BLOB locator using the prop_data bitmask in prop_flags (see Table 48-9) or by passing one or more RAW buffers.

When forUpdate is 0, this procedure also accepts a valid "as of" timestamp parameter as part of ctx that can be used by stores to implement "as of" style flashback queries. Mutating versions of the GETPATH Procedures do not support these modes of operation.

Syntax

\[
\text{DBMS\_DBFS\_CONTENT\_SPI.\text{GETPATH} (}
\begin{align*}
\text{store\_name} & \quad \text{IN} \quad \text{VARCHAR2}, \\
\text{path} & \quad \text{IN} \quad \text{VARCHAR2}, \\
\text{properties} & \quad \text{IN} \quad \text{OUT} \quad \text{NOCOPY} \quad \text{DBMS\_DBFS\_CONTENT\_PROPERTIES\_T}, \\
\text{content} & \quad \text{OUT} \quad \text{NOCOPY} \quad \text{BLOB}, \\
\text{item\_type} & \quad \text{OUT} \quad \text{INTEGER}, \\
\text{prop\_flags} & \quad \text{IN} \quad \text{INTEGER}, \\
\text{forUpdate} & \quad \text{IN} \quad \text{INTEGER}, \\
\text{deref} & \quad \text{IN} \quad \text{INTEGER}, \\
\text{ctx} & \quad \text{IN} \quad \text{DBMS\_DBFS\_CONTENT\_CONTEXT\_T});
\end{align*}
\]

\[
\text{DBMS\_DBFS\_CONTENT\_SPI.\text{GETPATH} (}
\begin{align*}
\text{store\_name} & \quad \text{IN} \quad \text{VARCHAR2}, \\
\text{path} & \quad \text{IN} \quad \text{VARCHAR2}, \\
\text{properties} & \quad \text{IN} \quad \text{OUT} \quad \text{NOCOPY} \quad \text{DBMS\_DBFS\_CONTENT\_PROPERTIES\_T}, \\
\text{amount} & \quad \text{IN} \quad \text{OUT} \quad \text{NUMBER}, \\
\text{offset} & \quad \text{IN} \quad \text{NUMBER}, \\
\text{buffer} & \quad \text{OUT} \quad \text{NOCOPY} \quad \text{RAW},
\end{align*}
\]

49-10
prop_flags IN INTEGER,
ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);

DBMS_DBFS_CONTENT_SPI.GETPATH {
  store_name IN VARCHAR2,
  path IN VARCHAR2,
  properties IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  amount IN OUT NUMBER,
  offset IN NUMBER,
  buffers OUT NOCOPY DBMS_DBFS_CONTENT_RAW_T,
  prop_flags IN INTEGER,
  ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);

Parameters

Table 49-12 GETPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to path items</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be returned depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>content</td>
<td>BLOB holding data which populates the file (optional)</td>
</tr>
<tr>
<td>item_type</td>
<td>Type of the path item specified (see Table 48-4)</td>
</tr>
<tr>
<td>amount</td>
<td>On input, number of bytes to be read. On output, number of bytes read</td>
</tr>
<tr>
<td>offset</td>
<td>Byte offset from which to begin reading</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer to which to write</td>
</tr>
<tr>
<td>buffers</td>
<td>Buffers to which to write</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>forUpdate</td>
<td>Specifies that a lock should be taken to signify exclusive write access to the path item</td>
</tr>
<tr>
<td>deref</td>
<td>If nonzero, attempts to resolve the given path item to actual data provided it is a reference (symbolic link)</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.11 GETPATHBYSTOREID Function

If the underlying GUID is found in the underlying store, this function returns the store-qualified path name.

Syntax

DBMS_DBFS_CONTENT_SPI.GETPATHBYSTOREID {
  store_name IN VARCHAR2,
  guid IN INTEGER
} RETURN VARCHAR2;
Parameters

Table 49-13  GETPATHBystoreID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>guid</td>
<td>Unique ID representing the desired path item</td>
</tr>
</tbody>
</table>

Return Values

Store-qualified path name represented by the GUID

Usage Notes

If the STD_GUID is unknown, a NULL value is returned. Clients are expected to handle this as appropriate.

49.4.12 GETPATHNOWAIT Procedure

This procedure implies that the operation is for an update, and, if implemented, allows providers to return an exception (ORA-00054) rather than wait for row locks.

See FEATURE_NOWAIT in Table 48-5 for more information.

Syntax

```
DBMS_DBFS_CONTENT_SPI.GETPATHNOWAIT (
    store_name  IN              VARCHAR2,
    path        IN              VARCHAR2,
    properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
    content     OUT    NOCOPY   BLOB,
    item_type   OUT             INTEGER,
    prop_flags  IN              INTEGER,
    deref       IN              INTEGER,
    ctx         IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 49-14  GETPATHNOWAIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to path items</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be returned depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>content</td>
<td>BLOB holding data which populates the file (optional)</td>
</tr>
<tr>
<td>item_type</td>
<td>Type of the path item specified (see Table 48-4)</td>
</tr>
</tbody>
</table>
### Table 49-14 (Cont.) GETPATHNOWAIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prop_flags</td>
<td>Determines which properties are returned. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>deref</td>
<td>If nonzero, attempts to resolve the given path item to actual data provided it is a reference (symbolic link)</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

### 49.4.13 GETSTOREID Function

This function returns the ID of a store.

**Syntax**

```sql
DBMS_DBFS_CONTENT_SPI.GETSTOREID (store_name IN VARCHAR2) RETURN NUMBER;
```

**Parameters**

**Table 49-15 GETSTOREID Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

**Return Values**

ID of the Store

**Usage Notes**

A store ID identifies a provider-specific store, across registrations and mounts, but independent of changes to the store contents. For this reason, changes to the store table or tables should be reflected in the store ID, but re-initialization of the same store table or tables should preserve the store ID.

### 49.4.14 GETVERSION Function

This function returns the version associated with a store.

**Syntax**

```sql
DBMS_DBFS_CONTENT_SPI.GETVERSION (store_name IN VARCHAR2) RETURN VARCHAR2;
```
Parameters

Table 49-16    GETVERSION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Return Values

A “version” (either specific to a provider package, or to an individual store) based on a standard a.b.c naming convention (for major, minor, and patch components)

49.4.15 ISPATHLOCKED Procedure

This procedure checks if any user-level locks are applied on a given path.

Syntax

```sql
DBMS_DBFS_CONTENT.ISPATHLOCKED (
    store_name IN VARCHAR2,
    path IN VARCHAR2,
    who IN VARCHAR2,
    lock_type IN OUT INTEGER,
    ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 49-17    ISPATHLOCKED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Path name of items to be locked</td>
</tr>
<tr>
<td>who</td>
<td>Transaction identifier that has locked the path</td>
</tr>
<tr>
<td>lock_type</td>
<td>One of the available lock types (see Table 48-6)</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.16 LIST Function

This function lists the contents of a directory path name.

The invoker of the subprogram has the option to investigate recursively into sub-directories, to make soft-deleted items visible, to use a flashback “as of” a specified timestamp, and to filter items within the store based on list predicates.

Syntax

```sql
DBMS_DBFS_CONTENT_SPI.LIST (
    store_name IN VARCHAR2,
    path IN VARCHAR2,
    filter IN VARCHAR2,
    recurse IN INTEGER,
```


ctx IN DBMS_DBFS_CONTENT_CONTEXT_T
RETURN DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;

Parameters

Table 49-18 LIST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of repository</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to directories</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively. Otherwise, recursively list the contents of directories and files below the given directory.</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

Return Values

Path items found that match the path, filter and criteria for executing recursively (see DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type)

Usage Notes

This function returns only list items; the client is expected to explicitly use one of the GETPATH Procedures to access the properties or content associated with an item.

49.4.17 LOCKPATH Procedure

This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.

Syntax

```sql
DBMS_DBFS_CONTENT_SPI.LOCKPATH (
    store_name IN VARCHAR2,
    path IN VARCHAR2,
    who IN VARCHAR2,
    lock_type IN INTEGER,
    waitForRowLock IN INTEGER,
    ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 49-19 LOCKPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Path name of items to be locked</td>
</tr>
<tr>
<td>who</td>
<td>Transaction identifier that has locked the path</td>
</tr>
<tr>
<td>lock_type</td>
<td>One of the available lock types (see Table 48-6)</td>
</tr>
<tr>
<td>waitForRowLock</td>
<td>Determines if a row is locked by a transaction or not</td>
</tr>
</tbody>
</table>
Table 49-19  (Cont.) LOCKPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

Usage Notes

- It is the responsibility of the store and its providers (assuming it supports user-defined lock checking) to ensure that lock and unlock operations are performed in a consistent manner.
- The status of locked items is available by means of various optional properties (see OPT_LOCK* in Table 48-8).

49.4.18 PURGEALL Procedure

This procedure purges all soft-deleted entries matching the path and optional filter criteria.

Syntax

\[
\text{DBMS_DBFS_CONTENT_SPI.PURGEALL (}
\begin{align*}
\text{store_name} & \quad \text{IN} & \quad \text{VARCHAR2}, \\
\text{path} & \quad \text{IN} & \quad \text{VARCHAR2}, \\
\text{filter} & \quad \text{IN} & \quad \text{VARCHAR2}, \\
\text{ctx} & \quad \text{IN} & \quad \text{DBMS_DBFS_CONTENT_CONTEXT_T});
\end{align*}
\]

Parameters

Table 49-20  PURGEALL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied based on specified criteria</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.19 PURGEPATH Procedure

This procedure purges any soft-deleted versions of the given path item.

Syntax

\[
\text{DBMS_DBFS_CONTENT_SPI.PURGEPATH (}
\begin{align*}
\text{path} & \quad \text{IN} & \quad \text{VARCHAR2}, \\
\text{filter} & \quad \text{IN} & \quad \text{VARCHAR2}, \\
\text{store_name} & \quad \text{IN} & \quad \text{VARCHAR2}, \\
\text{ctx} & \quad \text{IN} & \quad \text{DBMS_DBFS_CONTENT_CONTEXT_T});
\end{align*}
\]
### Parameters

#### Table 49-21  PURGEPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to file items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a>)</td>
</tr>
</tbody>
</table>

#### 49.4.20 PUTPATH Procedures

This procedure creates a new path item.

#### Syntax

```sql
DBMS_DBFS_CONTENT_SPI.PUTPATH (  
    store_name    IN              VARCHAR2,  
    path          IN              VARCHAR2,  
    properties    IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
    content       IN OUT NOCOPY   BLOB,  
    item_type     OUT             INTEGER,  
    prop_flags    IN              INTEGER,  
    ctx           IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

```sql
DBMS_DBFS_CONTENT_SPI.PUTPATH (  
    store_name    IN              VARCHAR2,  
    path          IN              VARCHAR2,  
    properties    IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
    amount        IN              NUMBER,  
    offset        IN              NUMBER,  
    buffer        IN              RAW,  
    prop_flags    IN              INTEGER,  
    ctx           IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

```sql
DBMS_DBFS_CONTENT_SPI.PUTPATH (  
    store_name    IN              VARCHAR2,  
    path          IN              VARCHAR2,  
    properties    IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
    written       OUT             NUMBER,  
    offset        IN              NUMBER,  
    buffers       IN              DBMS_DBFS_CONTENT_RAW_T,  
    prop_flags    IN              INTEGER,  
    ctx           IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

#### Parameters

#### Table 49-22  PUTPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Path name of item to be put</td>
</tr>
</tbody>
</table>
### Table 49-22 (Cont.) PUTPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set depending on <code>prop_flags</code> (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>content</td>
<td>BLOB holding data which populates the file (optional)</td>
</tr>
<tr>
<td>item_type</td>
<td>Type of the path item specified (see Table 48-4)</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes to be read</td>
</tr>
<tr>
<td>written</td>
<td>Number of bytes written</td>
</tr>
<tr>
<td>offset</td>
<td>Byte offset from which to begin reading</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer to which to write</td>
</tr>
<tr>
<td>buffers</td>
<td>Buffers to which to write</td>
</tr>
<tr>
<td>prop_flags</td>
<td>Determines which properties are set. Default is PROP_STD. Specify properties to be returned by setting <code>prop_spec</code>, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

**Usage Notes**

- All path names allow their metadata (properties) to be read and modified. On completion of the call, the client can access specific properties using `prop_flags` (see Table 48-9).
- On completion of the call, the client can request a new BLOB locator that can be used to continue data access using the `prop_data` bitmask in `prop_flags` (see Table 48-9).
- Files can also be written without using BLOB locators, by explicitly specifying logical offsets or buffer-amounts, and a suitably sized buffer.

### 49.4.21 RENAMEPATH Procedure

This procedure renames or moves a path. This operation can be performed across directory hierarchies and mount-points as long as it is within the same store.

**Note:**

See Oracle Database SecureFiles and Large Objects Developer's Guide for further information on Rename and Move operations.

**Syntax**

```sql
DBMS_DBFS_CONTENT_SPI.RENAMEPATH (  
    store_name    IN              VARCHAR2,  
    oldPath       IN              VARCHAR2,  
    newPath       IN              VARCHAR2,  
)
```
49.4.22 RESTOREALL Procedure

This procedure restores all soft-deleted path items meeting the path and optional filter criteria.

Syntax

DBMS_DBFS_CONTENT_SPI.RESTOREALL (  
   store_name     IN      VARCHAR2,  
   path           IN      VARCHAR2,  
   filter         IN      VARCHAR2,  
   ctx            IN      DBMS_DBFS_CONTENT_CONTEXT_T);  

Parameters

Table 49-24  RESTOREALL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to path items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.23 RESTOREPATH Procedure

This procedure restores all soft-deleted path items that match the given path and optional filter criteria.

Syntax

DBMS_DBFS_CONTENT_SPI.RESTOREPATH (  
   store_name     IN      VARCHAR2,  
   path           IN      VARCHAR2,  
   filter         IN      VARCHAR2,  
   ctx            IN      DBMS_DBFS_CONTENT_CONTEXT_T);  

Parameters

Table 49-24  RESTOREPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to path items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>
filter IN VARCHAR2,
ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);

Parameters

Table 49-25  RESTOREPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to path items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.24 SEARCH Function

This function searches for path items matching the given path and filter criteria.

Syntax

DBMS_DBFS_CONTENT_SPI.SEARCH (  
    store_name     IN      VARCHAR2,
    path           IN      VARCHAR2,
    filter         IN      VARCHAR2,
    recurse        IN      INTEGER,
    ctx            IN      DBMS_DBFS_CONTENT_CONTEXT_T)  
RETURN  DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;

Parameters

Table 49-26  LIST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to the path items</td>
</tr>
<tr>
<td>filter</td>
<td>A filter, if any, to be applied</td>
</tr>
<tr>
<td>recurse</td>
<td>If 0, do not execute recursively. Otherwise, recursively search the contents of directories and files below the given directory.</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

Return Values

Path items matching the given path and filter criteria (see DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type)
49.4.25 SETPATH Procedure

This procedure assigns a path name to a path item represented by contentID.
Stores and their providers that support contentID-based access and lazy path name binding also support the SETPATH Procedure that associates an existing contentID with a new path.

**Note:**
See Oracle Database SecureFiles and Large Objects Developer's Guide for further information on Rename and Move operations.

**Syntax**

```
DBMS_DBFS_CONTENT_SPI.SETPATH (  
    store_name   IN              VARCHAR2,  
    contentID    IN              RAW,  
    path         IN              VARCHAR2,  
    properties   IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,  
    ctx          IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of the store</td>
</tr>
<tr>
<td>contentID</td>
<td>Unique identifier for the item to be associated</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to path item</td>
</tr>
<tr>
<td>properties</td>
<td>One or more properties and their values to be set depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

49.4.26 SPACEUSAGE Procedure

This procedure queries file system space usage statistics.
Providers are expected to support this subprogram for their stores and to make a best effort determination of space usage, especially if the store consists of multiple tables, indexes, LOBs, and so on.

**Syntax**

```
DBMS_DBFS_CONTENT_SPI.SPACEUSAGE (  
    store_name   IN        VARCHAR2,  
    blksize      OUT       INTEGER,  
    tbytes       OUT       INTEGER,  
    fbytes       OUT       INTEGER,  
    ctx          IN        DBMS_DBFS_CONTENT_CONTEXT_T);
```
Parameters

Table 49-28 SPACEUSAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>blksize</td>
<td>Natural tablespace blocksize that holds the store. If multiple tablespaces with different blocksizes are used, any valid blocksize is acceptable.</td>
</tr>
<tr>
<td>tbytes</td>
<td>Total size of the store in bytes computed over all segments that comprise the store</td>
</tr>
<tr>
<td>fbytes</td>
<td>Free or unused size of the store in bytes computed over all segments that comprise the store</td>
</tr>
<tr>
<td>nfile</td>
<td>Number of currently available files in the store</td>
</tr>
<tr>
<td>ndir</td>
<td>Number of currently available directories in the store</td>
</tr>
<tr>
<td>nlink</td>
<td>Number of currently available links in the store</td>
</tr>
<tr>
<td>nref</td>
<td>Number of currently available references in the store</td>
</tr>
</tbody>
</table>

Usage Notes

- A space usage query on the top-level root directory returns a combined summary of the space usage of all available distinct stores under it (if the same store is mounted multiple times, it is still counted only once).
- Since database objects are dynamically expandable, it is not easy to estimate the division between “free” space and “used” space.

49.4.27 UNLOCKPATH Procedure

This procedure unlocks path items that were previously locked with the LOCKPATH Procedure.

Syntax

```sql
DBMS_DBFS_CONTENT_SPI.UNLOCKPATH (
    store_name IN VARCHAR2,
    path IN VARCHAR2,
    who IN VARCHAR2,
    waitForRowLock IN INTEGER,
    ctx IN DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 49-29 UNLOCKPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>path</td>
<td>Name of path to the path items</td>
</tr>
<tr>
<td>who</td>
<td>Transaction identifier that has locked the path</td>
</tr>
<tr>
<td>waitForRowLock</td>
<td>Determines if a row is locked by a transaction or not</td>
</tr>
<tr>
<td>ctx</td>
<td>Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)</td>
</tr>
</tbody>
</table>

**Related Topics**

- **LOCKPATH Procedure**
  
  This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.
The Oracle Database File System Hierarchical Store is implemented in the DBMS_DBFS_HS package. This package provides users the ability to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management for their database tables.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Constants**
- **Operational Notes**
- **Summary of DBMS_DBFS_HS Subprograms**

**See Also:**
- Oracle Database SecureFiles and Large Objects Developer's Guide

### 50.1 DBMS_DBFS_HS Overview

The DBMS_DBFS_HS package is a service provider underneath the DBMS_DBFS_CONTENT package that enables use of tape or Amazon S3 Web service as storage for data.

The data on tape or Amazon S3 Web service is part of the Oracle Database and can be accessed through all standard interfaces, but only through the database. The package allows users to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management of their content.

The package initially stores all content files in level-1 cache. As the level-1 cache fills up, content files are moved to level-2 cache and then to an external storage device using bulk writes.

### 50.2 DBMS_DBFS_HS Security Model

The DBMS_DBFS_HS package runs with invoker's rights.

### 50.3 DBMS_DBFS_HS Constants

The DBMS_DBFS_HS package uses the constants shown in the tables in this topic.
**Table 50-1  DBMS_DBFS_HS Constants - Used by the CREATESTORE Procedure**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORETYPE_TAPE</td>
<td>VARCHAR2(50)</td>
<td>'HS_TAPE'</td>
<td>Use tape as a storage tier</td>
</tr>
<tr>
<td>STORETYPE_AMAZON</td>
<td>VARCHAR2(50)</td>
<td>'HS_S3'</td>
<td>Use Amazon S3 Web service as a storage tier</td>
</tr>
</tbody>
</table>

**Table 50-2  DBMS_DBFS_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPNAME_BUCKET</td>
<td>VARCHAR2(50)</td>
<td>'BUCKET'</td>
<td>Specifies the AWS bucket to be used as a storage tier by the Hierarchical Store. Restrictions on bucket name are: 1) Bucket names can only contain lowercase letters, numbers, periods (.) and dashes(-). Note that underscores (_) are invalid. 2) Bucket names must start with a number or letter. 3) Bucket names cannot be in an IP address style (192.168.5.4). 4) Bucket names must be between 3 and 63 characters long. 5) Bucket names should not end with a dash. 6) Dashes cannot appear next to periods. For example, my-.bucket.com is invalid.</td>
</tr>
<tr>
<td>PROPNAME_CACHES</td>
<td>VARCHAR2(50)</td>
<td>'CACHE_SIZE'</td>
<td>Specifies the cumulative cache size used for the Hierarchical Store. This property is set by the CREATESTORE Procedure and can be modified by the RECONFIGCACHE Procedure. It cannot be modified by the SETSTOREPROPERTY Procedure, though its value can be queried by the GETSTOREPROPERTY Function.</td>
</tr>
<tr>
<td>PROPNAME_COMPRESSION</td>
<td>VARCHAR2(50)</td>
<td>'COMPRESSION_LEVEL'</td>
<td>Use to enable compression of files stored in the DBFS hierarchical store. It specifies the compression level to be used for compressing the files</td>
</tr>
<tr>
<td>Constant</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------</td>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROPNAME_ENABLE CLEANUPONDELETE</td>
<td>VARCHAR2(50)</td>
<td>'ENABLE_CLEAN UP_ON_DELETE'</td>
<td>If this property is set to 'TRUE', whenever the user invokes the DELETEFILE Procedure in the DBMS_DBFS_CONTENT interface on a file residing in the DBMS_DBFS_HS store, the DBMS_DBFS_HS removes the file on the external storage that contains this user file, provided that the file has no other useful data. By default, the property is set to 'TRUE' for STORETYPE_AMAZONS3 and 'FALSE' for STORETYPE_TAPE.</td>
</tr>
<tr>
<td>PROPNAME_HTTPPROXY</td>
<td>VARCHAR2(50)</td>
<td>'HTTP_PROXY'</td>
<td>Specifies the DNS name of the HTTP proxy, if any, that is needed to access the Amazon S3 storage service</td>
</tr>
<tr>
<td>PROPNAME_LICENSED</td>
<td>VARCHAR2(50)</td>
<td>'LICENSE_ID'</td>
<td>Specifies the license ID associated with the library libosbws11.so.</td>
</tr>
<tr>
<td>PROPNAME_LOBCACHE_QUOTA</td>
<td>VARCHAR2(50)</td>
<td>'LOBCACHE_QUOTA'</td>
<td>Specifies fraction of the cache_size which is allocated for level 1 cache. The default value of this parameter is NULL which means that 0.8 (= 80%) of the cache_size is used for level 1 cache. This property cannot be modified by the SETSTOREPROPERTY Procedure though its value can be queried by the GETSTOREPROPERTY Function. Its value is set by CREATESTORE Procedure and can be modified by the RECONFIGCACHE Procedure.</td>
</tr>
<tr>
<td>PROPNAME_MEDIAPOOL</td>
<td>VARCHAR2(50)</td>
<td>'MEDIA_POOL'</td>
<td>Specifies the media pool number to use for storing the content</td>
</tr>
<tr>
<td>PROPVAL_COMPLVL _NONE</td>
<td>VARCHAR2(50)</td>
<td>'NONE'</td>
<td>Indicates no compression</td>
</tr>
<tr>
<td>PROPVAL_COMPLVL _LOW</td>
<td>VARCHAR2(50)</td>
<td>'LOW'</td>
<td>Use to set the compression level to LOW. This is expected to have the best performance while still providing a good compression ratio.</td>
</tr>
<tr>
<td>PROPVAL_COMPLVL _MEDIUM</td>
<td>VARCHAR2(50)</td>
<td>'MEDIUM'</td>
<td>Use to set the compression level to MEDIUM. This compression level is expected to provide better compression ratio than LOW but the time required for compression will be higher than compression level LOW.</td>
</tr>
<tr>
<td>Constant</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROPVAL.COMPLEVEL</td>
<td>VARCHAR2(50)</td>
<td>'HIGH'</td>
<td>Use to set the compression level to HIGH. This compression level is expected to provide the best compression ratio but compression time will in general be highest among the 3 compression levels.</td>
</tr>
<tr>
<td>PROPNAME_OPTTAR</td>
<td>VARCHAR2(50)</td>
<td>'OPTIMAL_TARBALLSIZE'</td>
<td>Specifies optimal_tarball_size as the maximum possible size of an archive file. Multiple content files are bundled together into one archive file and then the archive file is transferred to tape or Amazon S3. This is because creating one file on tape or Amazon S3 for every content file in the store is a prohibitively expensive operation. This property cannot be modified by the SETSTOREPROPERTY Procedure though its value can be queried by the GETSTOREPROPERTY Function. Its value is set by CREATESTORE Procedure and can be modified by the RECONFIGCACHE Procedure.</td>
</tr>
<tr>
<td>PROPNAME_READCH</td>
<td>VARCHAR2(50)</td>
<td>'READ CHUNK SIZE'</td>
<td>Specifies the size used by the SBT protocol to transfer data from tape or S3. This chunk is allocated in memory per transaction for retrieval of content files from an archive store, so the value of this property should be conservative. The default size of 1MB is typically good for most users.</td>
</tr>
<tr>
<td>PROPNAME_S3HOST</td>
<td>VARCHAR2(50)</td>
<td>'S3 HOST'</td>
<td>Specifies the HOST name of the Amazon S3 storage service. It must be s3.amazonaws.com.</td>
</tr>
<tr>
<td>PROPNAME_SBT_LIBRARY</td>
<td>VARCHAR2(50)</td>
<td>'SBT_LIBRARY'</td>
<td>Specifies the path of the shared library used by RMAN to communicate with Amazon S3. It is named libosbsw11.so and is available in rdbms/lib directory.</td>
</tr>
<tr>
<td>PROPNAME_STREAMABLE</td>
<td>VARCHAR2(50)</td>
<td>'STREAMABLE'</td>
<td>Indicates whether buffer-based PUT or GET should be done on this store. Valid values for are TRUE and FALSE. The default value of this property is TRUE.</td>
</tr>
</tbody>
</table>
Table 50-2  (Cont.) DBMS_DBFS_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPNAME_WALLET</td>
<td>VARCHAR2(50)</td>
<td>'WALLET'</td>
<td>The value of this property should be of the form:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LOCATION=file:filename</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CREDENTIAL_ALIAS=access/secret_alias</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PROXY_AUTH_ALIAS=proxyusername/password alias</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Defines the Oracle Wallet which contains the credentials of the Amazon S3 account associated with the store under consideration.</td>
</tr>
<tr>
<td>LOCATION:</td>
<td></td>
<td></td>
<td>The directory path that contains the Oracle wallet. The format is file:directory-path</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The format of wallet_path in Windows is, for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>file:c:\WINNT\Profiles\username\WALLETS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In UNIX or Linux it is, for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>file:/home/username/wallets</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When the package is executed in the Oracle database server, the wallet is accessed from the database server.</td>
</tr>
<tr>
<td>PASSWORD:</td>
<td></td>
<td></td>
<td>Defines the wallet password. If auto-login is enabled in wallet (this can be changed using the OWM utility), this parameter does not have to be specified. By default, the mkstore utility enables auto-login.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CREDENTIAL_ALIAS: Defines the credential alias for ACCESS_KEY and SECRET_KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specifies the size used by the SBT protocol to transfer data to tape or S3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This chunk is allocated in memory per transaction for PUT of Content Files to an archive store so the value should be conservative.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The default size of 1MB is typically good for most users.</td>
</tr>
</tbody>
</table>
Table 50-3  DBMS_DBFS_HS Constants - Used by the REGISTERSTORECOMMAND Function

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEFORE_PUT</td>
<td>NUMBER</td>
<td>'1'</td>
<td>Specified operation must be performed before writing a SECUREFILE to the remote store</td>
</tr>
<tr>
<td>BEFORE_GET</td>
<td>NUMBER</td>
<td>'2'</td>
<td>Specified operation must be performed before a retrieval operation such as reading a SECUREFILE from the remote device</td>
</tr>
</tbody>
</table>

Table 50-4  DBMS_DBFS_HS Constants - Failure/Success/Error

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAIL</td>
<td>NUMBER</td>
<td>'0'</td>
<td>Procedure or function did not execute successfully</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>NUMBER</td>
<td>'1'</td>
<td>Procedure or function completed successfully</td>
</tr>
<tr>
<td>ERROR</td>
<td>NUMBER</td>
<td>'2'</td>
<td>Procedure or function returned an error</td>
</tr>
</tbody>
</table>

50.4 DBMS_DBFS_HS Operational Notes

When the DBMS_DBFS_HS package is executed in the Oracle database server, the wallet is accessed from the database server.

50.5 Summary of DBMS_DBFS_HS Subprograms

This table lists and describes the subprograms in the DBMS_DBFS_HS package.

Table 50-5  DBMS_DBFS_HS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEANUPUNUSEDBACKUPFILES Procedure</td>
<td>Removes files created on the external storage device that hold no currently used data</td>
</tr>
<tr>
<td>CREATEBUCKET Procedure</td>
<td>Creates an AWS bucket, associated with a store of type STORETYPE_AMAZONS3 into which the Hierarchical Store can then move data</td>
</tr>
<tr>
<td>CREATESTORE Procedure</td>
<td>Creates a new hierarchical store</td>
</tr>
<tr>
<td>DEREGSTORECOMMAND Function</td>
<td>Removes a command that had been previously associated with a store through the RECONFIGCACHE Procedure</td>
</tr>
<tr>
<td>DROPSTORE Procedure</td>
<td>Deletes a previously created hierarchical store</td>
</tr>
<tr>
<td>FLUSHCACHE Procedure</td>
<td>Flushes (writes out) dirty contents from the level-1 cache.</td>
</tr>
<tr>
<td>GETSTOREPROPERTY Function</td>
<td>Retrieves the values of a property of a store</td>
</tr>
</tbody>
</table>
Table 50-5  (Cont.) DBMS_DBFS_HS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECONFIGCACHE Procedure</td>
<td>Reconfigures the parameters of the database cache used by the store</td>
</tr>
<tr>
<td>REGISTERSTORECOMMAND Procedure</td>
<td>Registers commands for a store with the Hierarchical Store to be sent to the Media Manager for the external storage device associated with the store</td>
</tr>
<tr>
<td>SENDCOMMAND Procedures</td>
<td>Sends a command to be executed on the external storage device's Media Manager</td>
</tr>
<tr>
<td>SETSTOREPROPERTY Procedure</td>
<td>Stores properties of a store in the database</td>
</tr>
<tr>
<td>STOREPUSH Procedure</td>
<td>Pushes locally staged data to the remote storage</td>
</tr>
</tbody>
</table>

50.5.1 CLEANUPUNUSEDBACKUPFILES Procedure

This procedure removes files created on the external storage device that hold no currently used data in them.

Syntax

```sql
DBMS_DBFS_HS.CLEANUPUNUSEDBACKUPFILES (store_name IN VARCHAR2);
```

Parameters

Table 50-6  CLEANUPUNUSEDBACKUPFILES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Usage Notes

- The action of removing files from external storage device can not be rolled back.
- This method can be executed periodically to clear space on the external storage device. Asynchronously deleting content from the external storage device is useful because it has minimal impact on the OLTP performance. The periodic scheduling can be accomplished using the DBMS_SCHEDULER package.

50.5.2 CREATEBUCKET Procedure

This procedure creates an AWS bucket, associated with a store of type STORETYPE_AMAZONS3 into which the Hierarchical Store can then move data.

Syntax

```sql
DBMS_DBFS_HS.CREATEBUCKET (store_name IN VARCHAR2);
```
Parameters

Table 50-7  CREATEBUCKET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Usage Notes

- The PROPNAME_BUCKET property of the store should be set before this subprogram is called.
- Once this procedure has successfully created a bucket in Amazon S3, the bucket can only be deleted using out-of-band methods, such as logging-in to S3 and deleting data (directories, files, and other items) for the bucket.

50.5.3 CREATESTORE Procedure

This procedure creates a new hierarchical store store_name of type STORE_TYPE (STORETYPE_TAPE or STORETYPE_AMAZONS3) in schema schema_name (defaulting to the current schema) under the ownership of the invoking session user.

Syntax

```sql
DBMS_DBFS_HS.CREATESTORE  (
    store_name              IN     VARCHAR2,
    store_type              IN     VARCHAR2,
    tbl_name                IN     VARCHAR2,
    tbs_name                IN     VARCHAR2,
    cache_size              IN     NUMBER,
    lob_cache_quota         IN     NUMBER DEFAULT NULL,
    optimal_tarball_size    IN     NUMBER DEFAULT NULL,
    schema_name             IN     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 50-8  CREATESTORE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>store_type</td>
<td>STORETYPE_TAPE or STORETYPE_AMAZONS3</td>
</tr>
<tr>
<td>tbl_name</td>
<td>Table for store entries</td>
</tr>
<tr>
<td>tbs_name</td>
<td>Tablespace for the store</td>
</tr>
<tr>
<td>cache_size</td>
<td>Amount of space used by the store to cache content in given tablespace</td>
</tr>
<tr>
<td>lob_cache_quota</td>
<td>Fraction of the cache_size which is allocated for level 1 cache. The default value of this parameter is NULL which means that 0.8 (= 80%) of the cache_size is used for level 1 cache.</td>
</tr>
</tbody>
</table>
Table 50-8 (Cont.) CREATESTORE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimal_tarball_size</td>
<td>Maximum possible size of the archive file. Multiple content files are bundled together into one archive file, and then the archive file is transferred to tape or Amazon S3. This is because creating one file on tape or Amazon S3 for every content file in the store is a prohibitively expensive operation. The value of is set by default to 10GB for tape and to 100MB for Amazon S3.</td>
</tr>
<tr>
<td>schema_name</td>
<td>Schema for the store</td>
</tr>
</tbody>
</table>

Usage Notes

CREATESTORE() sets certain properties of the store to default values. The user can use the methods SETSTOREPROPERTY() and RECONFIGCACHE() to appropriately change the property values and to set other properties of the store.

- Store names must be unique for an owner. The same store names can be used for different stores owned by different owners.
- Once a table space has been specified to store the store's content in a database, it cannot be changed later.
- This subprogram will execute like a DDL statement, performing an automatic COMMIT before and after execution.
- Stores using DBMS_DBFS_HS must not use singleton mount. This means that the singleton parameter should be FALSE and the store_mount parameter should have a non-NULL value in a call to the DBMS_DBFS_CONTENT.MOUNTSTORE procedure.

50.5.4 DEREGSTORECOMMAND Function

This procedure removes a command that had been previously associated with a store through the REGISTERSTORECOMMAND Procedure.

Syntax

```sql
DBMS_DBFS_HS.DEREGSTORECOMMAND ( 
    store_name IN VARCHAR2, 
    message IN VARCHAR2);
```

Parameters

Table 50-9 DEREGSTORECOMMAND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>message</td>
<td>Message to be deregistered</td>
</tr>
</tbody>
</table>
Usage Notes

If this subprogram successfully executes, its actions cannot be rolled back by the user. If the user wants to restore the previous state, the user must call the REGISTERSTORECOMMAND Procedure.

Related Topics

- REGISTERSTORECOMMAND Procedure
  This procedure registers commands for a store with the Hierarchical Store. These commands are sent to the Media Manager for the external storage device associated with the store.

50.5.5 DROPSTORE Procedure

This procedure deletes a previously created hierarchical store specified by name and owned by the invoking session user.

Syntax

```sql
DBMS_DBFS_HS.DROPSTORE  (
    store_name   IN     VARCHAR2,
    opt_flags    IN     INTEGER DEFAULT 0);
```

Parameters

Table 50-10    DROPSTORE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store owned by the invoking session user</td>
</tr>
<tr>
<td>opt_flags</td>
<td>User can specify optional flags. If DISABLE_CleanupBackupFiles is specified as one of the optional flags, the call to the CLEANUPUNUSEDBACKUPFILES Procedure is not issued. By default, when this flag is not set, the procedure implicitly cleans-up all unused backup files.</td>
</tr>
</tbody>
</table>

Usage Notes

- The procedure executes like a DDL in that it auto-commits before and after its execution.
- If CLEANUPBACKUPFILES is disabled during the procedure, the user must resort to out-of-band techniques to cleanup unused backup files. No further invocations of CLEANUPBACKUPFILES for a dropped store are possible through hierarchical store.
- This subprogram will un-register the store from DBMS_DBFS_CONTENT package. All files in the given store are deleted from the store (Tape or Amazon S3 Web service). The database table holding the store’s entries in the database, is also dropped by this subprogram.
50.5.6 FLUSHCACHE Procedure

This procedure flushes out dirty contents from level-1 cache, which can be locked, to level-2 cache, thereby freeing-up space in level 1 cache.

Syntax

```sql
DBMS_DBFS_HS.FLUSHCACHE  (  
    store_name    IN     VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

50.5.7 GETSTOREPROPERTY Function

This function retrieves the values of a property.

Syntax

```sql
DBMS_DBFS_HS.GETSTOREPROPERTY  (  
    store_name      IN     VARCHAR2,  
    property_name   IN     VARCHAR2,  
    noexcp          IN     BOOLEAN DEFAULT FALSE) RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>property_name</td>
<td>Name of property</td>
</tr>
<tr>
<td>noexcp</td>
<td>If set to FALSE, raises an exception if the property does not exist in the database. If noexcp is set to TRUE, returns NULL if the property does not exist.</td>
</tr>
</tbody>
</table>

Return Values

The values of a property.

Usage Notes

The specified store must already have been created.
50.5.8 RECONFIGCACHE Procedure

This procedure reconfigures the parameters of the database cache being used by the store.

Syntax

```sql
DBMS_DBFS_HS.RECONFIGCACHE (  
    store_name              IN    VARCHAR2,  
    cache_size              IN    NUMBER DEFAULT NULL,  
    lobcache_quota          IN    NUMBER DEFAULT NULL,  
    optimal_tarball_size    IN    NUMBER DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>cache_size</td>
<td>Cumulative cache size used for the Hierarchical Store</td>
</tr>
<tr>
<td>lobcache_quota</td>
<td>Fraction of the cache size that are assigned to level 1 cache</td>
</tr>
<tr>
<td>optimal_tarball_size</td>
<td>Maximum possible size of an archive file. Since creating one file for every content file in the store is a prohibitively expensive operation, multiple content files are bundled together into one archive file for transfer to tape or Amazon S3.</td>
</tr>
</tbody>
</table>

Usage Notes

- The specified store must already have been created before reconfiguration.
- The Hierarchical Store uses a level 1 cache and a level 2 cache. The level 1 cache subsumes most of the working set and the level 2 cache is used to perform bulk writes to the backend device.
- If any of the last 3 parameters is NULL, its value specified during store creation is used. If the parameter was NULL when the call to the CREATESTORE Procedure was issued, the DBMS_DBFS_HS package assigns a default value.

The DBMS_DBFS_HS package optimistically tries to allocate more than 1 tarball’s worth of size for level 2 cache to facilitate concurrency, though a minimum of 1 tarball size is necessary for level 2 cache.

The values for cumulative cache size and LOB cache quota decide allocation of space for the two caches. If values are not provided, a user might see an INSUFFICIENT_CACHE exception. In that case, it is better to revise the cache parameters in order to have a working store.

- If this subprogram successfully executes, its actions cannot be rolled back by the user. In that case, the user should call RECONFIGCACHE again with new or modified parameters.
50.5.9 REGISTERSTORECOMMAND Procedure

This procedure registers commands for a store with the Hierarchical Store. These commands are sent to the Media Manager for the external storage device associated with the store.

Syntax

```
DBMS_DBFS_HS.REGISTERSTORECOMMAND (  
  store_name      IN     VARCHAR2,  
  message         IN     VARCHAR2,  
  flags           IN     NUMBER);
```

Parameters

Table 50-14  REGISTERSTORECOMMAND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>message</td>
<td>Message to be sent to the Media Manager of the external store</td>
</tr>
<tr>
<td>flags</td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>· BEFORE_PUT CONSTANT NUMBER := 1;</td>
</tr>
<tr>
<td></td>
<td>· BEFORE_GET CONSTANT NUMBER := 2;</td>
</tr>
</tbody>
</table>

Usage Notes

- These commands are sent before the next read or write of content. When the Hierarchical Store wants to push (or get) data to (or from) the storage device, it begins a session (to communicate with the device). After beginning the session, it sends all registered commands for the to the relevant device before writing (or getting) any data.
- If this method successfully executes, its actions cannot be rolled back by the user. To restore the previous state the user must call the DEREGSTORECOMMAND Function.

50.5.10 SENDCOMMAND Procedure

This procedure sends a command to be executed on the external storage device's Media Manager.

Syntax

```
DBMS_DBFS_HS.SENDCOMMAND (  
  store_name    IN       VARCHAR2,  
  message       IN       VARCHAR2);
```

Parameters

Table 50-15  SENDCOMMAND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>message</td>
<td>Message string to be executed</td>
</tr>
</tbody>
</table>
50.5.11 SETSTOREPROPERTY Procedure

This procedure stores properties of a store in the database as name-value pairs.

Syntax

```sql
DBMS_DBFS_HS.SETSTOREPROPERTY  (
    store_name      IN     VARCHAR2,
    property_name   IN     VARCHAR2,
    property_value  IN     VARCHAR2);
```

Parameters

Table 50-16  SETSTOREPROPERTY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>property_name</td>
<td>For a store using Tape device, there are three properties whose values must be set by the user, and four properties that have default values. Stores of type STORETYPE_AMAZONS3 have properties with default values. The various options for both types of stores are detailed under property_value.</td>
</tr>
<tr>
<td>property_value</td>
<td>Stores using a Tape Device</td>
</tr>
</tbody>
</table>

The values for the following properties must be set by the user:

- **PROPNAME_SBTLIBRARY** - This should point to the shared library used by RMAN to communicate with the external tape device. It is usually named libobk.so.
- **PROPNAME_MEDIAPool** - Media pool number for storing content
- **PROPNAME_CACHE_SIZE** - Amount of space, in bytes, used for the cache of this store

The following properties, which have default values assigned to them when a store is created, benefit from tuning:

- **PROPNAME_READCHUNKSIZE** and **PROPNAME_WRITECHUNKSIZE** - These are the sizes used by the SBT protocol to transfer data to and from the tapes. These chunks are allocated in memory per transaction, so the values should be conservative. The default size is 1MB.
- **PROPNAME_STREAMABLE** - Indicates whether DBFS_LINKs can perform read operations (for example SELECT or DBMS_LOB.READ) directly from the store, or if the data must be copied back into the database before it can be read
- **PROPNAME_ENABLECLEANUPONDELETE** - Indicates if DBMS_DBFS_HS should delete unused files on the external storage. Valid values for this property are 'FALSE' for STORETYPE_TAPE.
- **PROPNAME_COMPRESSLEVEL** - Describes how files written to Tape should be compressed. It can be set to PROPVAL_COMPLVL_NONE, PROPVAL_COMPLVL_LOW, PROPVAL_COMPLVL_MEDIUM or PROPVAL_COMPLVL_HIGH. By default it is set to PROPVAL_COMPLVL_NONE.
**Table 50-16  (Cont.) SETSTOREPROPERTY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(cont) property_value</td>
<td>Stores of type STORETYPE_AMAZONS3</td>
</tr>
<tr>
<td></td>
<td>It is mandatory that the following properties have assigned values, and default values are provided:</td>
</tr>
<tr>
<td></td>
<td>• PROPNAME_SBTLIBRARY - Specifies the path of the shared library used by RMAN to communicate with Amazon S3. It is named libosbws11.so and is available in rdbms/lib directory.</td>
</tr>
<tr>
<td></td>
<td>• PROPNAME_S3HOST - Defines the HOST name of the Amazon S3 storage service. It must be s3.amazonaws.com.</td>
</tr>
<tr>
<td></td>
<td>• PROPNAME_BUCKET - Defines the AWS bucket used as a storage tier by the Hierarchical Store. Restrictions on bucket names are:</td>
</tr>
<tr>
<td></td>
<td>-- Bucket names can only contain lowercase letters, numbers, periods (.) and dashes (-). Use of an underscore (_) is invalid.</td>
</tr>
<tr>
<td></td>
<td>-- Bucket names must start with a number or letter</td>
</tr>
<tr>
<td></td>
<td>-- Bucket names cannot be in an IP address style (&quot;192.168.5.4&quot;)</td>
</tr>
<tr>
<td></td>
<td>-- Bucket names must be between 3 and 63 characters in length</td>
</tr>
<tr>
<td></td>
<td>-- Bucket names should not end with a dash</td>
</tr>
<tr>
<td></td>
<td>-- Dashes cannot appear next to periods. For example, &quot;my-.bucket.com&quot; is invalid.</td>
</tr>
<tr>
<td></td>
<td>• PROPNAME_LICENSEID - Specifies the license ID associated with the library libosbws11.so.</td>
</tr>
<tr>
<td></td>
<td>• PROPNAME_WALLET - Has the form:</td>
</tr>
<tr>
<td></td>
<td>LOCATION=file:&lt;filename&gt;</td>
</tr>
<tr>
<td></td>
<td>CREDENTIAL_ALIAS=&lt;access/secret_alias&gt;</td>
</tr>
<tr>
<td></td>
<td>PROXY_AUTH_ALIAS=&lt;proxyusername/password alias&gt;</td>
</tr>
<tr>
<td></td>
<td>-- LOCATION - Directory path that contains the Oracle wallet. The format is file:directory-path. See Examples for variations in format.</td>
</tr>
<tr>
<td></td>
<td>-- PASSWORD - Defines the wallet password. If auto-login is enabled in the wallet (this can be changed using the user's own utility), and does not have to be specified. By default, the mkstore utility enables auto-login.</td>
</tr>
<tr>
<td></td>
<td>-- CREDENTIAL_ALIAS - Defines the credential alias for ACCESS_KEY and SECRET_KEY</td>
</tr>
<tr>
<td></td>
<td>-- PROXY_AUTH_ALIASES - Defines authentication credentials for the proxy server, if applicable.</td>
</tr>
</tbody>
</table>
{property_value (contd.)}  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(property_value (contd.)</td>
<td>The following properties are optional:</td>
</tr>
<tr>
<td>· PROPNAME_HTTPPROXY</td>
<td>Defines the DNS name of the HTTP proxy, if any, that is needed to access the Amazon S3 storage service.</td>
</tr>
<tr>
<td>· PROPNAME_STREAMABLE</td>
<td>Indicates whether buffer-based PUT or GET operation should be done on this store. Valid values for this property are TRUE (default) and FALSE.</td>
</tr>
<tr>
<td>· PROPNAME_ENABLECLEANUPONDELETE</td>
<td>Indicates if DBMS_DBFS_HS should delete unused files on the external storage device. Default values for this property are FALSE for STORETYPE_TAPE and TRUE for STORETYPE_AMAZONS3.</td>
</tr>
<tr>
<td>· PROPNAME_COMPRESSLEVEL</td>
<td>Describes how files written to tape should be compressed. It can be set to PROPVAL_COMPLVL_NONE, PROPVAL_COMPLVL_LOW, PROPVAL_COMPLVL_MEDIUM or PROPVAL_COMPLVL_HIGH. By default it is set to PROPVAL_COMPLVL_NONE.</td>
</tr>
</tbody>
</table>

Usage Notes

- The specified store must already have been created.
- If this subprogram successfully executes, its actions cannot be rolled back by the user.
- The same property can be set multiple times to the same or different values using this subprogram.
- Regarding PROPNAME_ENABLECLEANUPONDELETE behavior, a job is created for each store by the DBMS_DBFS_HS to remove the unused files from the external storage. By default, the job is enabled for STORETYPE_AMAZONS3 and is disabled for STORETYPE_TAPE. If the ENABLECLEANUPONDELETE property is set to TRUE, the job is enabled; if the property is set to FALSE, the job is disabled. If enabled, the job runs at an interval of one hour by default. The DBMS_SCHEDULER package can be used to modify the schedule. The name of the job can be obtained by querying USER_DBFS_HS_FIXED_PROPERTIES for prop_name = 'DELJOB_NAME'.

Examples

Format

The format of wallet_path in Windows is, for example:

file:\c:\WINNT\Profiles\<username>\WALLETS

The format of wallet_path in UNIX or Linux is, for example:

file:/home/username/wallets
50.5.12 STOREPUSH Procedure

This procedure pushes locally staged data to the remote storage.

Syntax

DBMS_DBFS_HS.STOREPUSH (  
    store_name   IN        VARCHAR2,  
    path         IN        VARCHAR2 DEFAULT NULL);  

Parameters

Table 50-17  STOREPUSH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store whose content the client writes from local cache to the external store</td>
</tr>
<tr>
<td>path</td>
<td>A non-mount qualified (without mount point) path within the store. By default, its value is NULL which corresponds to the root path of the store.</td>
</tr>
</tbody>
</table>

Usage Notes

- The Hierarchical Store caches the content files locally in database tables. When enough content is amassed in the cache to make it efficient to write to the external storage device (or the cache is completely filled), the Hierarchical Store creates a tarball out of the local content and writes these tarballs as files on the external device. The size of the tarball created by the Hierarchical Store is controlled by the store property PROPNAME_OPTTARBALLSIZE.

- When the amount of free space in the cache is such that the caching of a content file will push the space used above cache_size, the Hierarchical Store will internally call STOREPUSH. The STOREPUSH Procedure creates tarball(s) out of the existing dirty or modified content files in the cache and writes them out to the external device. A STOREPUSH call is not guaranteed to write all the dirty content from local cache to the external storage, since some files may be locked by other sessions.

- STOREPUSH has a built-in ability feature allowing it to automatically resume operation. If a STOREPUSH call is interrupted (say by a network outage) after it has transferred some tarballs to the external device, it can be restarted after the outage and will then resume transferring data from the point it was interrupted. In other words, work done before the outage is not lost. STOREPUSH can safely be restarted and the effect is such as if the outage never occurred.

- If this method successfully executes, its actions cannot be rolled back by the user.

- By default, when path is NULL, all files in the store are candidates for STOREPUSH. If path has a valid input value, all files which are under the namespace of given path are written from the local cache to the external store. If a given path is an existing file, it is pushed out again to the remote store.
The `DBMS_DBFS_SFS` package provides an interface to operate a SecureFile-based store (SFS) for the content interface described in the `DBMS_DBFS_CONTENT` package.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Constants**
- **Summary of DBMS_DBFS_SFS Subprograms**

**See Also:**

*Oracle Database SecureFiles and Large Objects Developer's Guide*

### 51.1 DBMS_DBFS_SFS Overview

The `DBMS_DBFS_SFS` package is a sample implementation of a package that implements and extends the `DBMS_DBFS_CONTENT_SPI` interface. It provides a POSIX-compliant file system stored in the RDBMS.

### 51.2 DBMS_DBFS_SFS Security Model

The `DBMS_DBFS_SFS` package runs with `AUTHID CURRENT_USER`.

### 51.3 DBMS_DBFS_SFS Constants

The `DBMS_DBFS_SFS` package uses the constants shown in the following tables.

**Note:**

Oracle has deprecated the older encryptions and hashing algorithms. The deprecated algorithms for `DBMS_CRYPTO` and native network encryption include MD4, MD5, DES, 3DES, and RC4-related algorithms as well as 3DES for Transparent Data Encryption (TDE). Removing older, less secure cryptography algorithms prevents accidental use of these algorithms. To meet your security requirements, Oracle recommends that you use more modern cryptography algorithms, such as the Advanced Encryption Standard (AES).

- **Table 51-1**
### Table 51-1  DBMS_DBFS_SFS Constants - Compression Levels

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESSION_DEFAULT</td>
<td>VARCHAR2(32)</td>
<td>''</td>
<td>Use the default SecureFile compression level</td>
</tr>
<tr>
<td>COMPRESSION_LOW</td>
<td>VARCHAR2(32)</td>
<td>'LOW'</td>
<td>Use compression level 'LOW'</td>
</tr>
<tr>
<td>COMPRESSION_MEDIUM</td>
<td>VARCHAR2(32)</td>
<td>'MEDIUM'</td>
<td>Use compression level 'MEDIUM'</td>
</tr>
<tr>
<td>COMPRESSION_HIGH</td>
<td>VARCHAR2(32)</td>
<td>'HIGH'</td>
<td>Use compression level 'HIGH'</td>
</tr>
</tbody>
</table>

### Table 51-2  DBMS_DBFS_SFS Constants - Used by the encryption Parameter

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPTION_DEFAULT</td>
<td>VARCHAR2(32)</td>
<td>''</td>
<td>Use the default SecureFile encryption algorithm</td>
</tr>
<tr>
<td>ENCRYPTION_AES128</td>
<td>VARCHAR2(32)</td>
<td>'AES128'</td>
<td>Use encryption AES 128 bit</td>
</tr>
<tr>
<td>ENCRYPTION_AES192</td>
<td>VARCHAR2(32)</td>
<td>'AES192'</td>
<td>Use encryption AES 192 bit</td>
</tr>
<tr>
<td>ENCRYPTION_AES256</td>
<td>VARCHAR2(32)</td>
<td>'AES256'</td>
<td>Use encryption AES 256 bit</td>
</tr>
</tbody>
</table>

### Table 51-3  DBMS_DBFS_SFS Constants - Used by the npartitions Parameter

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_PARTITIONS</td>
<td>INTEGER</td>
<td>16</td>
<td>Default to 16 partitions</td>
</tr>
</tbody>
</table>

### Table 51-4  DBMS_DBFS_SFS Constants - Used by the partition_key Parameter

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION_BY_ITEM</td>
<td>INTEGER</td>
<td>1</td>
<td>Use a hash of the item name for the partition key</td>
</tr>
<tr>
<td>PARTITION_BY_PATH</td>
<td>INTEGER</td>
<td>2</td>
<td>Use a hash of the path name for the partition key</td>
</tr>
<tr>
<td>PARTITION_BY_GUID</td>
<td>INTEGER</td>
<td>3</td>
<td>Use a hash of the GUID as the partition key</td>
</tr>
</tbody>
</table>

### 51.4 Summary of DBMS_DBFS_SFS Subprograms

This table lists and describes the DBMS_DBFS_SFS Package subprograms.
Table 51-5  DBMS_DBFS_SFS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEFILESYSTEM Procedure</td>
<td>Creates a file system store</td>
</tr>
<tr>
<td>CREATESTORE Procedure</td>
<td>Creates a new DBFS SFS store</td>
</tr>
<tr>
<td>DROPFILESYSTEM Procedures</td>
<td>Drops the DBFS SFS store</td>
</tr>
<tr>
<td>INITIFS Procedure</td>
<td>Initializes a POSIX file system store</td>
</tr>
</tbody>
</table>

51.4.1 CREATEFILESYSTEM Procedure

This procedure creates a file system store.

Syntax

```sql
DBMS_DBFS_SFS.CREATEFILESYSTEM (  
  store_name         IN     VARCHAR2,  
  schema_name        IN     VARCHAR2    DEFAULT NULL,  
  tbl_name           IN     VARCHAR2    DEFAULT NULL,  
  tbl_tbs            IN     VARCHAR2    DEFAULT NULL,  
  lob_tbs            IN     VARCHAR2    DEFAULT NULL,  
  use_bf             IN     BOOLEAN     DEFAULT FALSE,  
  properties         IN     DBMS_DBFS_CONTENT_PROPERTIES_T DEFAULT NULL,  
  create_only        IN     BOOLEAN     FALSE,  
  use_objects        IN     BOOLEAN     DEFAULT FALSE,  
  with_grants        IN     BOOLEAN     DEFAULT FALSE,  
  do_dedup           IN     BOOLEAN     DEFAULT FALSE,  
  do_compress        IN     BOOLEAN     DEFAULT FALSE,  
  compression        IN     VARCHAR2    DEFAULT COMPRESSION_DEFAULT,  
  do_encrypt         IN     BOOLEAN     DEFAULT FALSE,  
  encryption         IN     VARCHAR2    DEFAULT ENCRYPTION_DEFAULT,  
  do_partition       IN     BOOLEAN     DEFAULT FALSE,  
  npartitions        IN     NUMBER      DEFAULT DEFAULT_PARTITIONS,  
  partition_key      IN     NUMBER      DEFAULT PARTITION_BY_ITEM,  
  partition_guidi    IN     BOOLEAN     DEFAULT FALSE,  
  partition_pathi    IN     BOOLEAN     DEFAULT FALSE,  
  partition_prop     IN     BOOLEAN     DEFAULT TRUE);
```

Parameters

Table 51-6  CREATEFILESYSTEM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>schema_name</td>
<td>Schema for the store, defaulting to the current schema</td>
</tr>
<tr>
<td>tbl_name</td>
<td>Table for store entries. If not specified, an internally generated name is used.</td>
</tr>
<tr>
<td>tbl_tbs</td>
<td>Tablespace for the store, defaulting to the schema's default tablespace</td>
</tr>
<tr>
<td>lob_tbs</td>
<td>Tablespace in which to create the LOB segment. It defaults to the user's default tablespace.</td>
</tr>
<tr>
<td>use_bf</td>
<td>If TRUE, a BasicFile LOB is used; otherwise a SecureFile LOB is used.</td>
</tr>
</tbody>
</table>
Table 51-6  (Cont.) CREATEFILESYSTEM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>Table of (name, value, typecode) tuples used to configure the store properties. Currently no such properties are defined or used.</td>
</tr>
<tr>
<td>create_only</td>
<td>If TRUE, the file system is created, but not registered with the current user</td>
</tr>
<tr>
<td>use_objects</td>
<td>If TRUE, a single base-table with an object-type column (using a nested table) is created to backup the new file system. Otherwise, a pair of (parent, child) tables is used to backup the file system. In both cases, the object type nested table or the child table is used only for user-defined properties.</td>
</tr>
<tr>
<td>with_grants</td>
<td>If TRUE, DML and query access permissions are granted to the DBFS_ROLE as part of creating the file system. Otherwise, explicit grants (or existing permissions) are required to access the file system.</td>
</tr>
<tr>
<td>do_dedup</td>
<td>If TRUE, do deduplication the underlying SecureFile column</td>
</tr>
<tr>
<td>do_compress</td>
<td>If TRUE, do compression the underlying SecureFile column</td>
</tr>
<tr>
<td>compression</td>
<td>Compression algorithm to use (see Table 51-1)</td>
</tr>
<tr>
<td>do_encrypt</td>
<td>If TRUE, encrypt the underlying SecureFile column</td>
</tr>
<tr>
<td>encryption</td>
<td>encryption algorithm to use (see Table 51-2)</td>
</tr>
<tr>
<td>do_partition</td>
<td>If TRUE, partition the table used for storage</td>
</tr>
<tr>
<td>npartitions</td>
<td>Number of partitions to create for the table (see Table 51-3).</td>
</tr>
<tr>
<td>partition_key</td>
<td>How to partition the table: by item name, by path name, or by GUID (see Table 51-4).</td>
</tr>
<tr>
<td>partition_guidi</td>
<td>If TRUE, build an index on GUID</td>
</tr>
<tr>
<td>partition_pathi</td>
<td>If TRUE, build an index on path name</td>
</tr>
<tr>
<td>partition_prop</td>
<td>If TRUE, partition the properties table</td>
</tr>
</tbody>
</table>

Usage Notes

The procedure executes like a DDL in that it auto-commits before and after its execution.

51.4.2 CREATESTORE Procedure

This procedure creates a new DBFS SFS store owned by the invoking session user.

Syntax

```sql
DBMS_DBFS_SFS.CREATESTORE  (  
  store_name     IN     VARCHAR2,  
  tbl_name       IN     VARCHAR2 DEFAULT NULL,  
  tbs_name       in     VARCHAR2 DEFAULT NULL,  
  use_bf         in     BOOLEAN DEFAULT FALSE,  
  stgopts        in     VARCHAR2 DEFAULT '');
```
Parameters

Table 51-7  CREATESTORE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
<tr>
<td>store_type</td>
<td>STORETYPE_TAPE or STORETYPE_AMAZONS3</td>
</tr>
<tr>
<td>tbl_name</td>
<td>Placeholder for the store content cached in database</td>
</tr>
<tr>
<td>tbs_name</td>
<td>Named tablespace</td>
</tr>
<tr>
<td>use_bf</td>
<td>If TRUE, a BasicFile LOB is used; otherwise a SecureFile LOB is used.</td>
</tr>
<tr>
<td>stgopts</td>
<td>Currently non-operational, reserved for future use</td>
</tr>
</tbody>
</table>

51.4.3 DROPFILESYSTEM Procedures

This procedure drops the DBFS SFS store, purging all dictionary information associated with the store, and dropping the underlying file system table.

Syntax

```sql
DBMS_DBFS_SFS.DROPFILESYSTEM  (
  schema_name    IN      VARCHAR2 DEFAULT NULL,
  tbl_name       IN      INTEGER);
```

```sql
DBMS_DBFS_SFS.DROPFILESYSTEM  (
  store_name     IN      VARCHAR2);
```

Parameters

Table 51-8  DROPFILESYSTEM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of schema</td>
</tr>
<tr>
<td>tbl_name</td>
<td>Name of tablespace</td>
</tr>
<tr>
<td>store_name</td>
<td>Name of store path</td>
</tr>
</tbody>
</table>

Usage Notes

- If the specified store table is registered by the current user, it will be unregistered from the content interface described in the DBMS_DBFS_CONTENT package and the POSIX metadata tables.
- Subsequent to unregistration, an attempt will be made to store table(s). This operation may fail if other users are currently using this store table.
- The user attempting a drop of the tables underlying the store must actually have the privileges to complete the drop operation (either as the owner of the tables, or as a sufficiently privileged user for cross-schema operations).
- The procedure executes like a DDL in that it auto-commits before and after its execution.
51.4.4 INITFS Procedure

This procedure initialize a POSIX file system store. The table associated with the POSIX file system store store_name is truncated and reinitialized with a single "root" directory entry.

Syntax

DBMS_DBFS_SFS.INITFS (store_name IN VARCHAR2);

Parameters

Table 51-9  INITFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_name</td>
<td>Name of store</td>
</tr>
</tbody>
</table>

Usage Notes

The procedure executes like a DDL in that it auto-commits before and after its execution.
This package provides access to some SQL data definition language (DDL) statements from stored procedures. It also provides special administration operations that are not available as Data Definition Language statements (DDLs).

This chapter contains the following topics:

- Deprecated Subprograms
- Security Model
- Operational Notes
- Summary of DBMS_DDL Subprograms

52.1 DBMS_DDL Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with release Oracle Database 10g:

- ALTER_COMPILE Procedure

52.2 DBMS_DDL Security Model

This package runs with the privileges of the calling user, rather than the package owner SYS.

52.3 DBMS_DDL Operational Notes

The ALTER_COMPILE procedure commits the current transaction, performs the operation, and then commits again.

52.4 Summary of DBMS_DDL Subprograms

This table lists the DBMS_DDL subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_COMPILE Procedure</td>
<td>Compiles the PL/SQL object</td>
</tr>
<tr>
<td>ALTER_TABLE_NOT_REFERENCEABLEoodle Procedure</td>
<td>Reorganizes object tables</td>
</tr>
<tr>
<td>ALTER_TABLE_REFERENCEABLE Procedure</td>
<td>Reorganizes object tables</td>
</tr>
</tbody>
</table>
Table 52-1  (Cont.) DBMS_DDL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_WRAPPED Procedures</td>
<td>Takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body, generates a CREATE OR REPLACE statement with the PL/SQL source text obfuscated and executes the generated statement</td>
</tr>
<tr>
<td>IS_TRIGGER_FIRE_ONCE Function</td>
<td>Returns TRUE if the specified DML or DDL trigger is set to fire once. Otherwise, returns FALSE</td>
</tr>
<tr>
<td>SET_TRIGGER_FIRING_PROPERTY Procedures</td>
<td>Sets the specified DML or DDL trigger's firing property</td>
</tr>
<tr>
<td>WRAP Functions</td>
<td>Takes as input a CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body and returns a CREATE OR REPLACE statement where the text of the PL/SQL unit has been obfuscated</td>
</tr>
</tbody>
</table>

52.4.1 ALTER_COMPILE Procedure

This procedure is equivalent to the SQL statement: ALTER PROCEDURE|FUNCTION| PACKAGE [<schema>.] <name> Compile [BODY]

Note:

This procedure is deprecated in Oracle Database 10g Release 2 (10.2). While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the DDL equivalent in a dynamic SQL statement.

Syntax

```
DBMS_DDL.ALTER_COMPILE (type VARCHAR2, schema VARCHAR2, name VARCHAR2, reuse_settings BOOLEAN := FALSE);
```

Parameters

Table 52-2  ALTER_COMPILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Must be either PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY or TRIGGER</td>
</tr>
<tr>
<td>schema</td>
<td>Schema name</td>
</tr>
<tr>
<td></td>
<td>If NULL, then use current schema (case-sensitive)</td>
</tr>
</tbody>
</table>
Table 52-2  (Cont.) ALTER_COMPILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the object (case-sensitive)</td>
</tr>
<tr>
<td>reuse_settings</td>
<td>Indicates whether the session settings in the objects should be reused, or</td>
</tr>
<tr>
<td></td>
<td>whether the current session settings should be adopted instead</td>
</tr>
</tbody>
</table>

Exceptions

Table 52-3  ALTER_COMPILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000:</td>
<td>Insufficient privileges or object does not exist</td>
</tr>
<tr>
<td>ORA-20001:</td>
<td>Remote object, cannot compile</td>
</tr>
<tr>
<td>ORA-20002:</td>
<td>Bad value for object type: should be either PACKAGE, PACKAGE BODY,</td>
</tr>
<tr>
<td></td>
<td>PROCEDURE, FUNCTION, or TRIGGER</td>
</tr>
</tbody>
</table>

52.4.2 ALTER_TABLE_NOT_REFERENCEABLE Procedure

This procedure alters the given object table table_schema.table_name so it becomes not the default referenceable table for the schema affected_schema.

This is equivalent to SQL:

ALTER TABLE [table_schema].<table_name> NOT REFERENCEABLE FOR <affected_schema>

which is currently not supported or available as a DDL statement.

Syntax

DBMS_DDL.ALTER_TABLE_NOT_REFERENCEABLE {
  table_name        IN VARCHAR2,
  table_schema      IN DEFAULT NULL,
  affected_schema   IN DEFAULT NULL};

Parameters

Table 52-4  ALTER_TABLE_NOT_REFERENCEABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table to be altered. Cannot be a synonym. Must not be NULL.</td>
</tr>
<tr>
<td></td>
<td>Case sensitive.</td>
</tr>
<tr>
<td>table_schema</td>
<td>Name of the schema owning the table to be altered. If NULL then the current</td>
</tr>
<tr>
<td></td>
<td>schema is used. Case sensitive.</td>
</tr>
<tr>
<td>affected_schema</td>
<td>Name of the schema affected by this alteration. If NULL then the current</td>
</tr>
<tr>
<td></td>
<td>schema is used. Case sensitive.</td>
</tr>
</tbody>
</table>
Usage Notes

This procedure simply reverts for the affected schema to the default table referenceable for PUBLIC; that is, it simply undoes the previous ALTER_TABLE_REFERENCEABLE call for this specific schema. The affected schema must a particular schema (cannot be PUBLIC).

The user that executes this procedure must own the table (that is, the schema is the same as the user), and the affected schema must be the same as the user.

If the user executing this procedure has ALTER ANY TABLE and SELECT ANY TABLE and DROP ANY TABLE privileges, the user doesn't have to own the table and the affected schema can be any valid schema.

52.4.3 ALTER_TABLE_REFERENCEABLE Procedure

This procedure alters the given object table table_schema.table_name so it becomes the referenceable table for the given schema affected_schema.

This is equivalent to SQL:

ALTER TABLE [<table_schema>.]<table_name>  REFERENCEABLE FOR <affected_schema>

which is currently not supported or available as a DDL statement.

Syntax

DBMS_DDL.ALTER_TABLE_REFERENCEABLE
  table_name       IN  VARCHAR2,
  table_schema     IN  DEFAULT  NULL,
  affected_schema  IN  DEFAULT  NULL);

Parameters

Table 52-5  ALTER_TABLE_REFERENCEABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table to be altered. Cannot be a synonym. Must not be NULL. Case sensitive.</td>
</tr>
<tr>
<td>table_schema</td>
<td>Name of the schema owning the table to be altered. If NULL then the current schema is used. Case sensitive.</td>
</tr>
<tr>
<td>affected_schema</td>
<td>Name of the schema affected by this alteration. If NULL then the current schema is used. Case sensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

When you create an object table, it automatically becomes referenceable, unless you use the OID AS clause when creating the table. The OID AS clause makes it possible for you to create an object table and to assign to the new table the same EOID as another object table of the same type. After you create a new table using the OID AS clause, you end up with two object table with the same EOID; the new table is not referenceable, the original one is. All references that used to point to the objects in the original table still reference the same objects in the same original table.
If you execute this procedure on the new table, it makes the new table the referenceable table replacing the original one; thus, those references now point to the objects in the new table instead of the original table.

52.4.4 CREATE_WRAPPED Procedures

The procedure takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body. It then generates a CREATE OR REPLACE statement with the PL/SQL source text obfuscated and executes the generated statement. In effect, this procedure bundles together the operations of wrapping the text and creating the PL/SQL unit.

See Also:
WRAP Functions

This procedure has 3 overloads. Each of the three functions provides better performance than using a combination of individual WRAP Functions and DBMS_SQL.PARSE (or EXECUTE IMMEDIATE) calls. The different functionality of each form of syntax is presented with the definition.

Syntax

Is a shortcut for EXECUTE IMMEDIATE SYS.DBMS_DDL.WRAP(ddl):

DBMS_DDL.CREATE_WRAPPED (  
  ddl     VARCHAR2);

Is a shortcut for DBMS_SQL.PARSE(cursor, SYS.DBMS_DDL.WRAP (input, lb, ub)):

DBMS_DDL.CREATE_WRAPPED(  
  ddl     DBMS_SQL.VARCHAR2A,  
  lb      PLS_INTEGER,  
  ub      PLS_INTEGER);

Is a shortcut for DBMS_SQL.PARSE(cursor, SYS.DBMS_DDL.WRAP (input, lb, ub)):

DBMS_DDL.CREATE_WRAPPED(  
  ddl     DBMS_SQL.VARCHAR2S,  
  lb      PLS_INTEGER,  
  ub      PLS_INTEGER);

Parameters

Table 52-6  CREATE_WRAPPED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddl</td>
<td>A CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body</td>
</tr>
<tr>
<td>lb</td>
<td>Lower bound for indices in the string table that specify the CREATE OR REPLACE statement</td>
</tr>
</tbody>
</table>
Table 52-6  (Cont.) CREATE_WRAPPED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ub</td>
<td>Upper bound for indices in the string table that specify the CREATE OR REPLACE statement.</td>
</tr>
</tbody>
</table>

Usage Notes

- The CREATE OR REPLACE statement is executed with the privileges of the user invoking DBMS_DDL.CREATE_WRAPPED.
- Any PL/SQL code that attempts to call these interfaces should use the fully qualified package name SYS.DBMS_DDL to avoid the possibility that the name DBMS_DDL is captured by a locally-defined unit or by redefining the DBMS_DDL public synonym.
- Each invocation of any accepts only a single PL/SQL unit. By contrast, the PL/SQL wrap utility accepts a entire SQL*Plus file and obfuscates the PL/SQL units within the file leaving all other text as-is. These interfaces are intended to be used in conjunction with or as a replacement for PL/SQL's dynamic SQL interfaces (EXECUTE_IMMEDIATE and DBMS_SQL.PARSE). Since these dynamic SQL interfaces only accept a single unit at a time (and do not understand the SQL*Plus "/") termination character), both the CREATE_WRAPPED Procedures and the WRAP Functions require input to be a single unit.

Exceptions

ORA-24230: If the input is not a CREATE OR REPLACE statement specifying a PL/SQL unit, exception DBMS_DDL.MALFORMED_WRAP_INPUT is raised.

Examples

DECLARE
    ddl VARCHAR2(32767);
BEGIN
    ddl := GENERATE_PACKAGE(...);
    SYS.DBMS_DDL.CREATE_WRAPPED(ddl); -- Instead of EXECUTE IMMEDIATE ddl
END;

52.4.5 IS_TRIGGER_FIRE_ONCE Function

This function returns TRUE if the specified DML or DDL trigger is set to fire once. Otherwise, it returns FALSE.

A fire once trigger fires in a user session but does not fire in the following cases:

- For changes made by a Streams apply process
- For changes made by executing one or more Streams apply errors using the EXECUTE_ERROR or EXECUTE_ALL_ERRORS procedure in the DBMS_APPLY_ADM package
- For changes made by a Logical Standby apply process
**Note:**

Only DML and DDL triggers can be fire once. All other types of triggers always fire.

**See Also:**

"SET_TRIGGER_FIRING_PROPERTY Procedures"

## Syntax

```sql
DBMS_DDL.IS_TRIGGER_FIRE_ONCE
    trig_owner         IN  VARCHAR2,
    trig_name          IN  VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trig_owner</td>
<td>Schema of trigger</td>
</tr>
<tr>
<td>trig_name</td>
<td>Name of trigger</td>
</tr>
</tbody>
</table>

## 52.4.6 SET_TRIGGER_FIRING_PROPERTY Procedures

This procedure sets the specified DML or DDL trigger’s firing property whether or not the property is set for the trigger.

Use this procedure to control a DML or DDL trigger’s firing property for changes:

- Applied by a Streams apply process
- Made by executing one or more Streams apply errors using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package.
- Applied by a Logical Standby apply process

### Syntax

```sql
DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY (  
    trig_owner         IN  VARCHAR2,
    trig_name          IN  VARCHAR2,
    fire_once          IN  BOOLEAN);
```

```sql
DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY (  
    trig_owner         IN  VARCHAR2,
    trig_name          IN  VARCHAR2,
    property           IN  INTEGER,
    setting            IN  BOOLEAN);
```
Parameters

Table 52-8  SET_TRIGGER_FIRINGPROPERTY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trig_owner</td>
<td>Schema of the trigger to set</td>
</tr>
<tr>
<td>trig_name</td>
<td>Name of the trigger to set</td>
</tr>
<tr>
<td>fire_once</td>
<td>If TRUE, the trigger is set to fire once. By default, the fire_once parameter is set to TRUE for DML and DDL triggers. If FALSE, the trigger is set to always fire unless apply_server_only property is set to TRUE, which overrides fire_once property setting.</td>
</tr>
<tr>
<td>property</td>
<td>DBMS_DDL.fire_once to set the fire_once property of the trigger</td>
</tr>
<tr>
<td>setting</td>
<td>Value of property being set</td>
</tr>
</tbody>
</table>

Usage Notes

DML triggers created on a table have their fire-once property set to TRUE. In this case, the triggers only fire when the table is modified by an user process, and they are automatically disabled inside Oracle processes maintaining either a logical standby database (SQL Apply) or Oracle processes doing replication (Streams Apply) processes, and thus do not fire when a SQL Apply or a Streams Apply process modifies the table. There are two ways for a user to fire a trigger as a result of SQL Apply or a Streams Apply process making a change to a maintained table: (a) setting the fire-once property of a trigger to FALSE, which allows it fire both in the context of a user process or a SQL or Streams Apply process, or (b) by setting the apply-server-only property to TRUE and thus making the trigger fire only in the context of a SQL Apply or a Streams Apply process and not in the context of a user process.

- **FIRE_ONCE=TRUE, APPLY_SERVER_ONLY=FALSE**
  This is the default property setting for a DML trigger. The trigger only fires when user process modifies the base table.

- **FIRE_ONCE=TRUE or FALSE, APPLY_SERVER_ONLY=TRUE**
  The trigger only fires when SQL Apply or Streams Apply process modifies the base table. The trigger does not fire when a user process modifies the base table. Thus the apply-server-only property overrides the fire-once property of a trigger.
Note:

- If you dequeue an error transaction from the error queue and execute it without using the DBMS_APPLY_ADM package, then relevant changes resulting from this execution cause a trigger to fire, regardless of the trigger firing property.
- Only DML and DDL triggers can be fire once. All other types of triggers always fire.

See Also:

Oracle Streams Concepts and Administration for more information about the apply process and controlling a trigger's firing property

52.4.7 WRAP Functions

This function takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body and returns a CREATE OR REPLACE statement where the text of the PL/SQL unit has been obfuscated.

The function has 3 overloads to allow for the different ways in which DDL statements can be generated dynamically and presented to DBMS_SQL or EXECUTE IMMEDIATE. The different functionality of each form of syntax is presented with the definition.

See Also:

CREATE_WRAPPED Procedures

Syntax

Provides basic functionality:

```sql
DBMS_DDL.WRAP(
    ddl      VARCHAR2)
RETURN VARCHAR2;
```

Provides the same functionality as the first form, but allows for larger inputs. This function is intended to be used with the PARSE Procedures in the DBMS_SQL package and its argument list follows the convention of DBMS_SQL.PARSE:

```sql
DBMS_DDL.WRAP(
    ddl      DBMS_SQL.VARCHAR2S,
    lb       PLS_INTEGER,
    ub       PLS_INTEGER)
RETURN DBMS_SQL.VARCHAR2S;
```

Provides the same functionality as the second form and is provided for compatibility with multiple forms of the PARSE Procedures in the DBMS_SQL package:
DBMS_DDL.WRAP(
    ddl  DBMS_SQL.VARCHAR2A,
    lb   PLS_INTEGER,
    ub   PLS_INTEGER)
RETURN DBMS_SQL.VARCHAR2A;

Parameters

Table 52-9  WRAP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddl</td>
<td>A CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body</td>
</tr>
<tr>
<td>lb</td>
<td>Lower bound for indices in the string table that specify the CREATE OR REPLACE statement</td>
</tr>
<tr>
<td>ub</td>
<td>Upper bound for indices in the string table that specify the CREATE OR REPLACE statement</td>
</tr>
</tbody>
</table>

Return Values

A CREATE OR REPLACE statement with the text obfuscated. In the case of the second and third form, the return value is a table of strings that need to be concatenated in order to construct the CREATE OR REPLACE string containing obfuscated source text.

Usage Notes

- Any PL/SQL code that attempts to call these interfaces should use the fully qualified package name SYS.DBMS_DDL to avoid the possibility that the name DBMS_DDL is captured by a locally-defined unit or by redefining the DBMS_DDL public synonym.
- Each invocation of any accepts only a single PL/SQL unit. By contrast, the PL/SQL wrap utility accepts a full SQL file and obfuscates the PL/SQL units within the file leaving all other text as-is. These interfaces are intended to be used in conjunction with or as a replacement for PL/SQL’s dynamic SQL interfaces (EXECUTE IMMEDIATE and DBMS_SQL.PARSE). Since these dynamic SQL interfaces only accept a single unit at a time (and do not understand the SQL*Plus "/*" termination character), both the CREATE_WRAPPED Procedures and the WRAP Functions require input to be a single unit.

Exceptions

ORA-24230: If the input is not a CREATE OR REPLACE statement specifying a PL/SQL unit, exception DBMS_DDL.MALFORMED_WRAP_INPUT is raised.

Examples

DECLARE
    ddl VARCHAR2(32767);
BEGIN
    ddl := GENERATE_PACKAGE(...);
    EXECUTE IMMEDIATE SYS.DBMS_DDL.WRAP(ddl); -- Instead of EXECUTE IMMEDIATE ddl
END;
**DBMS_DEBUG**

**DBMS_DEBUG** is deprecated. Use **DBMS_DEBUG_JDWP** instead.

See **DBMS_DEBUG_JDWP** for more information.

**DBMS_DEBUG** is a PL/SQL interface to the PL/SQL debugger layer, Probe, in the Oracle server.

This package is primarily intended to implement server-side debuggers and it provides a way to debug server-side PL/SQL program units.

---

**Note:**

The term *program unit* refers to a PL/SQL program of any type (procedure, function, package, package body, trigger, anonymous block, object type, or object type body).

---

This chapter contains the following topics:

- Overview
- Constants
- Variables
- Exceptions
- Operational Notes
- Data Structures
- Summary of DBMS_DEBUG Subprograms

### 53.1 DBMS_DEBUG Overview

To debug server-side code, you must have two database sessions: one session to run the code in debug mode (the target session), and a second session to supervise the target session (the debug session).

The target session becomes available for debugging by making initializing calls with **DBMS_DEBUG**. This marks the session so that the PL/SQL interpreter runs in debug mode and generates debug events. As debug events are generated, they are posted from the session. In most cases, debug events require return notification: the interpreter pauses awaiting a reply.

Meanwhile, the debug session must also initialize itself using **DBMS_DEBUG**: This tells it which target session to supervise. The debug session may then call entry points in **DBMS_DEBUG** to read events that were posted from the target session and to communicate with the target session.

The following subprograms are run in the target session (the session that is to be debugged):
53.2 DBMS_DEBUG Constants

A breakpoint status may have the following value: `breakpoint_status_unused`—breakpoint is not in use.

Otherwise, the status is a mask of the following values:

- `breakpoint_status_active`—a line breakpoint
- `breakpoint_status_disabled`—breakpoint is currently disabled
- `breakpoint_status_remote`—a shadow breakpoint (a local representation of a remote breakpoint)

53.3 DBMS_DEBUG Variables

The DBMS_DEBUG uses the variables shown in the following table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_timeout</td>
<td>The timeout value (used by both sessions). The smallest possible timeout is 1 second. If this value is set to 0, then a large value (3600) is used.</td>
</tr>
</tbody>
</table>

53.4 DBMS_DEBUG Exceptions

These values are returned by the various functions called in the debug session (`SYNCHRONIZE`, `CONTINUE`, `SET_BREAKPOINT`, and so on). If PL/SQL exceptions worked across client/server and server/server boundaries, then these would all be exceptions rather than error codes.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>Normal termination</td>
</tr>
</tbody>
</table>

Statuses returned by `GET_VALUE` and `SET_VALUE`:
### Table 53-3  DBMS_DEBUG Exceptions Returned by GET_VALUE and SET_VALUE

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_bogus_frame</td>
<td>No such entrypoint on the stack</td>
</tr>
<tr>
<td>error_no_debug_info</td>
<td>Program was compiled without debug symbols</td>
</tr>
<tr>
<td>error_no_such_object</td>
<td>No such variable or parameter</td>
</tr>
<tr>
<td>error_unknown_type</td>
<td>Debug information is unreadable</td>
</tr>
<tr>
<td>error_indexed_table</td>
<td>Returned by GET_VALUE if the object is a table, but no index was provided</td>
</tr>
<tr>
<td>error_illegal_index</td>
<td>No such element exists in the collection</td>
</tr>
<tr>
<td>error_nullcollection</td>
<td>Table is atomically NULL</td>
</tr>
<tr>
<td>error_nullvalue</td>
<td>Value is NULL</td>
</tr>
</tbody>
</table>

**Statuses returned by SET_VALUE:**

### Table 53-4  DBMS_DEBUG Exceptions Returned by SET_VALUE

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_illegal_value</td>
<td>Constraint violation</td>
</tr>
<tr>
<td>error_illegal_null</td>
<td>Constraint violation</td>
</tr>
<tr>
<td>error_value_malformed</td>
<td>Unable to decipher the given value</td>
</tr>
<tr>
<td>error_other</td>
<td>Some other error</td>
</tr>
<tr>
<td>error_name_incomplete</td>
<td>Name did not resolve to a scalar</td>
</tr>
</tbody>
</table>

**Statuses returned by the breakpoint functions:**

### Table 53-5  Statuses Returned by the Breakpoint Functions

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_no_such_breakpt</td>
<td>No such breakpoint</td>
</tr>
<tr>
<td>error_idle_breakpt</td>
<td>Cannot enable or disable an unused breakpoint</td>
</tr>
<tr>
<td>error_bad_handle</td>
<td>Unable to set breakpoint in given program (nonexistent or security violation)</td>
</tr>
</tbody>
</table>

**General error codes (returned by many of the DBMS_DEBUG subprograms):**

### Table 53-6  DBMS_DEBUG Subprograms Error Codes

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_unimplemented</td>
<td>Functionality is not yet implemented</td>
</tr>
<tr>
<td>error_deferred</td>
<td>No program running; operation deferred</td>
</tr>
<tr>
<td>error_exception</td>
<td>An exception was raised in the DBMS_DEBUG or Probe packages on the server</td>
</tr>
</tbody>
</table>
Table 53-6  (Cont.) DBMS_DEBUG Subprograms Error Codes

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_communication</td>
<td>Some error other than a timeout occurred</td>
</tr>
<tr>
<td>error_timeout</td>
<td>Timeout occurred</td>
</tr>
</tbody>
</table>

Table 53-7  illegal_init Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>illegal_init</td>
<td>DEBUG_ON was called prior to INITIALIZE</td>
</tr>
</tbody>
</table>

The following exceptions are raised by procedure SELF_CHECK:

Table 53-8  SELF_CHECK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe_creation_failure</td>
<td>Could not create a pipe</td>
</tr>
<tr>
<td>pipe_send_failure</td>
<td>Could not write data to the pipe</td>
</tr>
<tr>
<td>pipe_receive_failure</td>
<td>Could not read data from the pipe</td>
</tr>
<tr>
<td>pipe_datatype_mismatch</td>
<td>Datatype in the pipe was wrong</td>
</tr>
<tr>
<td>pipe_data_error</td>
<td>Data got garbled in the pipe</td>
</tr>
</tbody>
</table>

53.5 DBMS_DEBUG Operational Notes

There are two ways to ensure that debug information is generated: through a session switch, or through individual recompilation.

To set the session switch, enter the following statement:

```
ALTER SESSION SET PLSQL_DEBUG = true;
```

This instructs the compiler to generate debug information for the remainder of the session. It does not recompile any existing PL/SQL.

To generate debug information for existing PL/SQL code, use one of the following statements (the second recompiles a package or type body):

```
ALTER [PROCEDURE | FUNCTION | PACKAGE | TRIGGER | TYPE] <name> COMPILE DEBUG;
ALTER [PACKAGE | TYPE] <name> COMPILE DEBUG BODY;
```

Figure 53-1 and Figure 53-2 illustrate the flow of operations in the session to be debugged and in the debugging session.
Initialize session for debugging, and generate/specify unique debugID.  
DBMS_DEBUG.initialize()

Start debugging  
DBMS_DEBUG.debug_on()

Stop debugging  
DBMS_DEBUG.debug_off()

Execute PL/SQL programs

Figure 53-1  Target Session
Figure 53-2  Debug Session

Input: debugID from target session
Initialize
DBMS_DEBUG.attach_session()

Manipulate breakpoints
DBMS_DEBUG.set_breakpoint()
DBMS_DEBUG.delete_breakpoint()
DBMS_DEBUG.disable_breakpoint()
DBMS_DEBUG.enable_breakpoint()
DBMS_DEBUG.show_breakpoints()

Read first event from target session
DBMS_DEBUG.synchronize()

Show stack
DBMS_DEBUG.print_backtrace()

Get/set values
DBMS_DEBUG.get_value()
DBMS_DEBUG.set_value()

Show source
DBMS_DEBUG.show_source()
Control of the Interpreter

The interpreter pauses execution at the following times:

1. At startup of the interpreter so any deferred breakpoints may be installed prior to execution.
2. At any line containing an enabled breakpoint.
3. At any line where an interesting event occurs. The set of interesting events is specified by the flags passed to DBMS_DEBUG.CONTINUE in the breakflags parameter.
Session Termination

There is no event for session termination. Therefore, it is the responsibility of the debug session to check and make sure that the target session has not ended. A call to DBMS_DEBUG.SYNCHRONIZE after the target session has ended causes the debug session to hang until it times out.

Deferred Operations

The diagram suggests that it is possible to set breakpoints prior to having a target session. This is true. In this case, Probe caches the breakpoint request and transmits it to the target session at first synchronization. However, if a breakpoint request is deferred in this fashion, then:

- SET_BREAKPOINT does not set the breakpoint number (it can be obtained later from SHOW_BREAKPOINTS if necessary).
- SET_BREAKPOINT does not validate the breakpoint request. If the requested source line does not exist, then an error silently occurs at synchronization, and no breakpoint is set.

Diagnostic Output

To debug Probe, there are diagnostics parameters to some of the calls in DBMS_DEBUG. These parameters specify whether to place diagnostic output in the RDBMS tracefile. If output to the RDBMS tracefile is disabled, these parameters have no effect.

Common Debug Session Sections

- Common Section
- Target Session
- Debug Session Section

Common Section

The following subprograms may be called in either the target or the debug session:

- PROBE_VERSION Procedure
- SELF_CHECK Procedure
- SET_TIMEOUT Function

Target Session

The following subprograms may be called only in the target session:

- INITIALIZE Function
- DEBUG_ON Procedure
- SET_TIMEOUT_BEHAVIOU Procedure
- GET_TIMEOUT_BEHAVIOU Function

Debug Session Section

The following subprograms should be run in the debug session only:

- ATTACH_SESSION Procedure
• SYNCHRONIZE Function
• SHOW_FRAME_SOURCE Procedure
• SHOW_SOURCE Procedures
• GET_MORE_SOURCE Procedure
• PRINT_BACKTRACE Procedure
• CONTINUE Function
• SET_BREAKPOINT Function
• DELETE_BREAKPOINT Function
• SET_OER_BREAKPOINT Function
• DELETE_OER_BREAKPOINT Function
• ENABLE_BREAKPOINT Function
• DISABLE_BREAKPOINT Function
• SHOW_BREAKPOINTS Procedures
• SET_VALUE Function
• GET_VALUE Function
• TARGET_PROGRAM_RUNNING Procedure
• DETACH_SESSION Procedure
• GET_RUNTIME_INFO Function
• PRINT_INSTANTIATIONS Procedure
• PING Procedure
• GET_LINE_MAP Function
• GET_RUNTIME_INFO Function
• GET_INDEXES Function
• EXECUTE Procedure

OER Breakpoints

Exceptions that are declared in PL/SQL programs are known as user-defined exceptions. In addition, there are Oracle Errors (OERs) that are returned from the Oracle kernel. To tie the two mechanisms together, PL/SQL provides the exception_init pragma that turns a user-defined exception into an OER, so that a PL/SQL handler may be used for it, and so that the PL/SQL engine can return OERs to the Oracle kernel. As of the current release, the only information available about an OER is its number. If two user-defined exceptions are exception_init’d to the same OER, they are indistinguishable.

Namespaces

Program units on the server reside in different namespaces. When setting a breakpoint, specify the desired namespace.

1. Namespace_cursor contains cursors (anonymous blocks).
2. Namespace_pgkspec_or_toplevel contains:
   • Package specifications.
• Procedures and functions that are not nested inside other packages, procedures, or functions.
• Object types.

3. Namespace_pkg_body contains package bodies and type bodies.


Libunit Types

These values are used to disambiguate among objects in a given namespace. These constants are used in PROGRAM_INFO when Probe is giving a stack backtrace.

• LibunitType_cursor
• LibunitType_procedure
• LibunitType_function
• LibunitType_package
• LibunitType_package_body
• LibunitType_trigger
• LibunitType_Unknown

Breakflags

These are values to use for the breakflags parameter to CONTINUE, in order to tell Probe what events are of interest to the client. These flags may be combined.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>break_next_line</td>
<td>Break at next source line (step over calls)</td>
</tr>
<tr>
<td>break_any_call</td>
<td>Break at next source line (step into calls)</td>
</tr>
<tr>
<td>break_any_return</td>
<td>Break after returning from current entrypoint (skip over any entrypoints called from the current routine)</td>
</tr>
<tr>
<td>break_return</td>
<td>Break the next time an entrypoint gets ready to return. (This includes entrypoints called from the current one. If interpreter is running Proc1, which calls Proc2, then break_return stops at the end of Proc2.)</td>
</tr>
<tr>
<td>break_exception</td>
<td>Break when an exception is raised</td>
</tr>
<tr>
<td>break_handler</td>
<td>Break when an exception handler is executed</td>
</tr>
<tr>
<td>abort_execution</td>
<td>Stop execution and force an 'exit' event as soon as DBMS_DEBUG.CONTINUE is called.</td>
</tr>
</tbody>
</table>

Information Flags

These are flags which may be passed as the info_requested parameter to SYNCHRONIZE, CONTINUE, and GET_RUNTIME_INFO.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info_getStackDepth</td>
<td>Get the current depth of the stack</td>
</tr>
<tr>
<td>info_getBreakpoint</td>
<td>Get the breakpoint number</td>
</tr>
<tr>
<td>Flag</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>info_getLineinfo</td>
<td>Get program unit information</td>
</tr>
</tbody>
</table>

**Reasons for Suspension**

After `CONTINUE` is run, the program either runs to completion or breaks on some line.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason_none</td>
<td>-</td>
</tr>
<tr>
<td>reason_interpreter_starting</td>
<td>Interpreter is starting</td>
</tr>
<tr>
<td>reason_breakpoint</td>
<td>Hit a breakpoint</td>
</tr>
<tr>
<td>reason_enter</td>
<td>Procedure entry</td>
</tr>
<tr>
<td>reason_return</td>
<td>Procedure is about to return</td>
</tr>
<tr>
<td>reason_finish</td>
<td>Procedure is finished</td>
</tr>
<tr>
<td>reason_line</td>
<td>Reached a new line</td>
</tr>
<tr>
<td>reason_interrupt</td>
<td>An interrupt occurred</td>
</tr>
<tr>
<td>reason_exception</td>
<td>An exception was raised</td>
</tr>
<tr>
<td>reason_exit</td>
<td>Interpreter is exiting (old form)</td>
</tr>
<tr>
<td>reason_knl_exit</td>
<td>Kernel is exiting</td>
</tr>
<tr>
<td>reason_handler</td>
<td>Start exception-handler</td>
</tr>
<tr>
<td>reason_timeout</td>
<td>A timeout occurred</td>
</tr>
<tr>
<td>reason_instantiate</td>
<td>Instantiation block</td>
</tr>
<tr>
<td>reason_abort</td>
<td>Interpreter is aborting</td>
</tr>
</tbody>
</table>

### 53.6 DBMS_DEBUG Data Structures

The `DBMS_DEBUG` package defines `RECORD` types and `TABLE` types.

**RECORD Types**

- `BREAKPOINT_INFO` Record Type
- `PROGRAM_INFO` Record Type
- `RUNTIME_INFO` Record Type

**TABLE Types**

- `BACKTRACE_TABLE` Table Type
- `BREAKPOINT_TABLE` Table Type
- `INDEX_TABLE` Table Type
- `VC2_TABLE` Table Type
53.6.1 BREAKPOINT_INFO Record Type

This type gives information about a breakpoint, such as its current status and the program unit in which it was placed.

Syntax

```sql
TYPE breakpoint_info IS RECORD (
   name        VARCHAR2(30),
   owner       VARCHAR2(30),
   dblink      VARCHAR2(30),
   line#       BINARY_INTEGER,
   libunittype BINARY_INTEGER,
   status      BINARY_INTEGER);
```

Fields

Table 53-9  BREAKPOINT_INFO Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the program unit</td>
</tr>
<tr>
<td>owner</td>
<td>Owner of the program unit</td>
</tr>
<tr>
<td>dblink</td>
<td>Database link, if remote</td>
</tr>
<tr>
<td>line#</td>
<td>Line number</td>
</tr>
<tr>
<td>libunittype</td>
<td>NULL, unless this is a nested procedure or function</td>
</tr>
<tr>
<td>status</td>
<td>See Constants for values of breakpoint_status_*</td>
</tr>
</tbody>
</table>

53.6.2 PROGRAM_INFO Record Type

The PROGRAM_INFO record type of the DBMS_DEBG package specifies a program location. It is a line number in a program unit.

This is used for stack backtraces and for setting and examining breakpoints. The read-only fields are currently ignored by Probe for breakpoint operations. They are set by Probe only for stack backtraces.

Syntax

```sql
TYPE program_info IS RECORD(
   -- The following fields are used when setting a breakpoint
   namespace        BINARY_INTEGER,
   name             VARCHAR2(30),
   owner            VARCHAR2(30),
   dblink           VARCHAR2(30),
   line#            BINARY_INTEGER,
   -- Read-only fields (set by Probe when doing a stack backtrace)
   libunittype      BINARY_INTEGER,
   entrypointname   VARCHAR2(30));
```
Fields

Table 53-10  PROGRAM_INFO Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>See DBMS_DEBUG Operational Notes for more information about namespaces.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the program unit</td>
</tr>
<tr>
<td>owner</td>
<td>Owner of the program unit</td>
</tr>
<tr>
<td>dblink</td>
<td>Database link, if remote</td>
</tr>
<tr>
<td>line#</td>
<td>Line number</td>
</tr>
<tr>
<td>libunittype</td>
<td>A read-only field, NULL, unless this is a nested procedure or function</td>
</tr>
<tr>
<td>entrypointname</td>
<td>A read-only field, to disambiguate among objects that share the same namespace (for example, procedure and package specifications). See DBMS_DEBUG Operational Notes for more information about the libunit types.</td>
</tr>
</tbody>
</table>

53.6.3 RUNTIME_INFO Record Type

This type gives context information about the running program.

Syntax

```plaintext
TYPE runtime_info IS RECORD(
    line#            BINARY_INTEGER,
    terminated       binary_integer,
    breakpoint       binary_integer,
    stackdepth       BINARY_INTEGER,
    interpreterdepth BINARY_INTEGER,
    reason           BINARY_INTEGER,
    program          program_info);
```

Fields

Table 53-11  RUNTIME_INFO Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line#</td>
<td>Duplicate of program.line#</td>
</tr>
<tr>
<td>terminated</td>
<td>Whether the program has terminated</td>
</tr>
<tr>
<td>breakpoint</td>
<td>Breakpoint number</td>
</tr>
<tr>
<td>stackdepth</td>
<td>Number of frames on the stack</td>
</tr>
<tr>
<td>interpreterdepth</td>
<td>[A reserved field]</td>
</tr>
<tr>
<td>reason</td>
<td>Reason for suspension</td>
</tr>
<tr>
<td>program</td>
<td>Source location</td>
</tr>
</tbody>
</table>
53.6.4 BACKTRACE_TABLE Table Type

This type is used by PRINT_BACKTRACE.

Syntax

```plaintext
TYPE backtrace_table IS TABLE OF program_info INDEX BY BINARY_INTEGER;
```

53.6.5 BREAKPOINT_TABLE Table Type

This type is used by SHOW_BREAKPOINTS.

Syntax

```plaintext
TYPE breakpoint_table IS TABLE OF breakpoint_info INDEX BY BINARY_INTEGER;
```

53.6.6 INDEX_TABLE Table Type

This type is used by GET_INDEXES to return the available indexes for an indexed table.

Syntax

```plaintext
TYPE index_table IS table of BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

53.6.7 VC2_TABLE Table Type

This type is used by SHOW_SOURCE.

Syntax

```plaintext
TYPE vc2_table IS TABLE OF VARCHAR2(90) INDEX BY BINARY_INTEGER;
```

53.7 Summary of DBMS_DEBUG Subprograms

This table lists the DBMS_DEBUG subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACH_SESSION Procedure</td>
<td>Notifies the debug session about the target debugID</td>
</tr>
<tr>
<td>CONTINUE Function</td>
<td>Continues execution of the target program</td>
</tr>
<tr>
<td>DEBUG_OFF Procedure</td>
<td>Turns debug-mode off</td>
</tr>
<tr>
<td>DEBUG_ON Procedure</td>
<td>Turns debug-mode on</td>
</tr>
<tr>
<td>DELETE_BREAKPOINT Function</td>
<td>Deletes a breakpoint</td>
</tr>
<tr>
<td>DELETE_OER_BREAKPOINT T Function</td>
<td>Deletes an OER breakpoint</td>
</tr>
</tbody>
</table>
Table 53-12 (Cont.) DBMS_DEBUG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETACH_SESSION Procedure</td>
<td>Stops debugging the target program</td>
</tr>
<tr>
<td>DISABLE_BREAKPOINT Function</td>
<td>Disables a breakpoint</td>
</tr>
<tr>
<td>ENABLE_BREAKPOINT Function</td>
<td>Activates an existing breakpoint</td>
</tr>
<tr>
<td>EXECUTE Procedure</td>
<td>Executes SQL or PL/SQL in the target session</td>
</tr>
<tr>
<td>GET_INDEXES Function</td>
<td>Returns the set of indexes for an indexed table</td>
</tr>
<tr>
<td>GET_MORE_SOURCE Procedure</td>
<td>Provides additional source in the event of buffer overflow when using SHOW_SOURCE</td>
</tr>
<tr>
<td>GET_LINE_MAP Function</td>
<td>Returns information about line numbers in a program unit</td>
</tr>
<tr>
<td>GET_RUNTIME_INFO Function</td>
<td>Returns information about the current program</td>
</tr>
<tr>
<td>GET_TIMEOUT_BEHAVIOR Function</td>
<td>Returns the current timeout behavior</td>
</tr>
<tr>
<td>GET_VALUE Function</td>
<td>Gets a value from the currently-running program</td>
</tr>
<tr>
<td>INITIALIZE Function</td>
<td>Sets debugID in target session</td>
</tr>
<tr>
<td>PING Procedure</td>
<td>Pings the target session to prevent it from timing out</td>
</tr>
<tr>
<td>PRINT_BACKTRACE Procedure</td>
<td>Prints a stack backtrace</td>
</tr>
<tr>
<td>PRINT_INSTANTIATIONS Procedure</td>
<td>Prints a stack backtrace</td>
</tr>
<tr>
<td>PROBE_VERSION Procedure</td>
<td>Returns the version number of DBMS_DEBUG on the server</td>
</tr>
<tr>
<td>SELF_CHECK Procedure</td>
<td>Performs an internal consistency check</td>
</tr>
<tr>
<td>SET_BREAKPOINT Function</td>
<td>Sets a breakpoint in a program unit</td>
</tr>
<tr>
<td>SET_OER_BREAKPOINT Function</td>
<td>Sets an OER breakpoint</td>
</tr>
<tr>
<td>SET_TIMEOUT Function</td>
<td>Sets the timeout value</td>
</tr>
<tr>
<td>SET_TIMEOUT_BEHAVIOR Procedure</td>
<td>Tells Probe what to do with the target session when a timeout occurs</td>
</tr>
<tr>
<td>SET_VALUE Function</td>
<td>Sets a value in the currently-running program</td>
</tr>
<tr>
<td>SHOW_BREAKPOINTS Procedures</td>
<td>Returns a listing of the current breakpoints</td>
</tr>
<tr>
<td>SHOW_FRAME_SOURCE Procedure</td>
<td>Fetches the frame source</td>
</tr>
<tr>
<td>SHOW_SOURCE Procedures</td>
<td>Fetches program source</td>
</tr>
<tr>
<td>SYNCHRONIZE Function</td>
<td>Waits for program to start running</td>
</tr>
<tr>
<td>TARGET_PROGRAM_RUNNING Procedure</td>
<td>Returns TRUE if the target session is currently executing a stored procedure, or FALSE if it is not</td>
</tr>
</tbody>
</table>
53.7.1 ATTACH_SESSION Procedure

This procedure notifies the debug session about the target program.

Syntax

```
DBMS_DEBUG.ATTACH_SESSION (
    debug_session_id  IN VARCHAR2,
    diagnostics       IN BINARY_INTEGER := 0);
```

Parameters

Table 53-13 ATTACH_SESSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug_session_id</td>
<td>Debug ID from a call to INITIALIZE in target session</td>
</tr>
<tr>
<td>diagnostics</td>
<td>Generate diagnostic output if nonzero</td>
</tr>
</tbody>
</table>

53.7.2 CONTINUE Function

This function passes the given breakflags (a mask of the events that are of interest) to Probe in the target process. It tells Probe to continue execution of the target process, and it waits until the target process runs to completion or signals an event.

If info_requested is not NULL, then calls GET_RUNTIME_INFO.

Syntax

```
DBMS_DEBUG.CONTINUE (
    run_info       IN OUT runtime_info,
    breakflags     IN     BINARY_INTEGER,
    info_requested IN     BINARY_INTEGER := NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 53-14 CONTINUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_info</td>
<td>Information about the state of the program</td>
</tr>
<tr>
<td>breakflags</td>
<td>Mask of events that are of interest (see the discussion about break flags under DBMS_DEBUG Operational Notes)</td>
</tr>
<tr>
<td>info_requested</td>
<td>Which information should be returned in run_info when the program stops (see the discussion of information flags under DBMS_DEBUG Operational Notes)</td>
</tr>
</tbody>
</table>
Return Values

Table 53-15  CONTINUE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td></td>
</tr>
<tr>
<td>error_timeout</td>
<td>Timed out before the program started running</td>
</tr>
<tr>
<td>error_communication</td>
<td>Other communication error</td>
</tr>
</tbody>
</table>

53.7.3 DEBUG_OFF Procedure

This procedure notifies the target session that debugging should no longer take place in that session. It is not necessary to call this function before ending the session.

⚠️ WARNING:

There must be a debug session waiting if immediate is **TRUE**.

Syntax

```
DBMS_DEBUG.DEBUG_OFF;
```

Usage Notes

The server does not handle this entrypoint specially. Therefore, it attempts to debug this entrypoint.

53.7.4 DEBUG_ON Procedure

This procedure marks the target session so that all PL/SQL is run in debug mode. This must be done before any debugging can take place.

Syntax

```
DBMS_DEBUG.DEBUG_ON (   no_client_side_plsql_engine  BOOLEAN := TRUE,   immediate                   BOOLEAN := FALSE);
```

Parameters

Table 53-16  DEBUG_ON Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_client_side_plsql_engine</td>
<td>Should be left to its default value unless the debugging session is taking place from a client-side PL/SQL engine</td>
</tr>
<tr>
<td>immediate</td>
<td>If this is <strong>TRUE</strong>, then the interpreter immediately switches itself into debug-mode, instead of continuing in regular mode for the duration of the call.</td>
</tr>
</tbody>
</table>
53.7.5 DELETE_BREAKPOINT Function

This function deletes a breakpoint.

Syntax

```sql
DBMS_DEBUG.DELETE_BREAKPOINT (  
    breakpoint IN BINARY_INTEGER)  
RETURN BINARY INTEGER;
```

Parameters

Table 53-17 DELETE_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakpoint</td>
<td>Breakpoint number from a previous call to SET_BREAKPOINT</td>
</tr>
</tbody>
</table>

Return Values

Table 53-18 DELETE_BREAKPOINT Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td></td>
</tr>
<tr>
<td>error_no_such_breakpt</td>
<td>No such breakpoint exists</td>
</tr>
<tr>
<td>error_idle_breakpt</td>
<td>Cannot delete an unused breakpoint</td>
</tr>
<tr>
<td>error_stale_breakpt</td>
<td>The program unit was redefined since the breakpoint was set</td>
</tr>
</tbody>
</table>

53.7.6 DELETE_OER_BREAKPOINT Function

This function deletes an OER breakpoint.

Syntax

```sql
DBMS_DEBUG.DELETE_OER_BREAKPOINT (  
    oer IN PLS_INTEGER)  
RETURN PLS INTEGER;
```

Parameters

Table 53-19 DELETE_OER_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oer</td>
<td>The OER (positive 4-byte number) to delete</td>
</tr>
</tbody>
</table>
53.7.7 DETACH_SESSION Procedure

This procedure stops debugging the target program.

This procedure may be called at any time, but it does not notify the target session that the debug session is detaching itself, and it does not terminate execution of the target session. Therefore, care should be taken to ensure that the target session does not hang itself.

Syntax

DBMS_DEBUG.DETACH_SESSION;

53.7.8 DISABLE_BREAKPOINT Function

This function makes an existing breakpoint inactive but leaves it in place.

Syntax

DBMS_DEBUG.DISABLE_BREAKPOINT (breakpoint IN BINARY_INTEGER) RETURN BINARY_INTEGER;

Parameters

Table 53-20 DISABLE_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakpoint</td>
<td>Breakpoint number from a previous call to SET_BREAKPOINT</td>
</tr>
</tbody>
</table>

Return Values

Table 53-21 DISABLE_BREAKPOINT Function Return Values

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td></td>
</tr>
<tr>
<td>error_no_such_breakpt</td>
<td>No such breakpoint exists</td>
</tr>
<tr>
<td>error_idle_breakpt</td>
<td>Cannot disable an unused breakpoint</td>
</tr>
</tbody>
</table>

53.7.9 ENABLE_BREAKPOINT Function

This function is the reverse of disabling. This enables a previously disabled breakpoint.

Syntax

DBMS_DEBUG.ENABLE_BREAKPOINT (breakpoint IN BINARY_INTEGER) RETURN BINARY_INTEGER;
Parameters

Table 53-22  ENABLE_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakpoint</td>
<td>Breakpoint number from a previous call to SET_BREAKPOINT</td>
</tr>
</tbody>
</table>

Return Values

Table 53-23  ENABLE_BREAKPOINT Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>Success</td>
</tr>
<tr>
<td>error_no_such_breakpt</td>
<td>No such breakpoint exists</td>
</tr>
<tr>
<td>error_idle_breakpt</td>
<td>Cannot enable an unused breakpoint</td>
</tr>
</tbody>
</table>

53.7.10 EXECUTE Procedure

This procedure executes SQL or PL/SQL code in the target session. The target session is assumed to be waiting at a breakpoint (or other event). The call to DBMS_DEBUG.EXECUTE occurs in the debug session, which then asks the target session to execute the code.

Syntax

```sql
DBMS_DEBUG.EXECUTE (
    what         IN VARCHAR2,
    frame#       IN BINARY_INTEGER,
    bind_results IN BINARY_INTEGER,
    results      IN OUT NOCOPY dbms_debug_vc2coll,
    errm         IN OUT NOCOPY VARCHAR2);
```

Parameters

Table 53-24  EXECUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>what</td>
<td>SQL or PL/SQL source to execute</td>
</tr>
<tr>
<td>frame#</td>
<td>The context in which to execute the code. Only -1 (global context) is supported at this time.</td>
</tr>
<tr>
<td>bind_results</td>
<td>Whether the source wants to bind to results in order to return values from the target session: 0 = No 1 = Yes</td>
</tr>
<tr>
<td>results</td>
<td>Collection in which to place results, if bind_results is not 0</td>
</tr>
<tr>
<td>errm</td>
<td>Error message, if an error occurred; otherwise, NULL</td>
</tr>
</tbody>
</table>
Examples

Example 1
This example executes a SQL statement. It returns no results.

```
DECLARE
    coll sys.dbms_debug_vc2coll; -- results (unused)
    errm VARCHAR2(100);
BEGIN
    dbms_debug.execute('insert into emp(ename,empno,deptno) ' ||
       'values(''LJE'', 1, 1),' ||
       '-1, 0, coll, errm);
END;
```

Example 2
This example executes a PL/SQL block, and it returns no results. The block is an autonomous transaction, which means that the value inserted into the table becomes visible in the debug session.

```
DECLARE
    coll sys.dbms_debug_vc2coll;
    errm VARCHAR2(100);
BEGIN
    dbms_debug.execute(
        'DECLARE PRAGMA autonomous_transaction; ' ||
        'BEGIN ' ||
        '   insert into emp(ename, empno, deptno) ' ||
        '   values(''LJE'', 1, 1); ' ||
        '   COMMIT; ' ||
        'END;',
        -1, 0, coll, errm);
END;
```

Example 3
This example executes a PL/SQL block, and it returns some results.

```
DECLARE
    coll sys.dbms_debug_vc2coll;
    errm VARCHAR2(100);
BEGIN
    dbms_debug.execute(
        'DECLARE ' ||
        '   pp SYS.dbms_debug_vc2coll := SYS.dbms_debug_vc2coll(); ' ||
        '   x PLS_INTEGER; ' ||
        '   i PLS_INTEGER := 1; ' ||
        'BEGIN ' ||
        '   SELECT COUNT(*) INTO x FROM emp; ' ||
        '   pp.EXEND(x * 6); ' ||
        '   FOR c IN (SELECT * FROM emp) LOOP ' ||
        '      pp(i) := ' ||
        '      pp(i) := ' ||
        '      pp(i) := ' ||
        '      pp(i) := ' ||
        '      pp(i) := ' ||
        '      pp(i) := null; i := i+1; ' ||
        '   END LOOP; ' ||
        '   :1 := pp;' ||
        'END;',
```

Summary of DBMS_DEBUG Subprograms
-l, 1, coll, erm);
each := coll.FIRST;
WHILE (each IS NOT NULL) LOOP
  dosomething(coll(each));
  each := coll.NEXT(each);
END LOOP;
END;

53.7.11 GET_INDEXES Function

Given a name of a variable or parameter, this function returns the set of its indexes, if it is an indexed table. An error is returned if it is not an indexed table.

Syntax

DBMS_DEBUG.GET_INDEXES (  
  varname   IN  VARCHAR2,  
  frame#    IN  BINARY_INTEGER,  
  handle    IN  program_info,  
  entries   OUT index_table)  
RETURN BINARY_INTEGER;

Parameters

Table 53-25 GET_INDEXES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varname</td>
<td>Name of the variable to get index information about</td>
</tr>
<tr>
<td>frame#</td>
<td>Number of frame in which the variable or parameter resides; NULL for a package variable</td>
</tr>
<tr>
<td>handle</td>
<td>Package description, if object is a package variable</td>
</tr>
<tr>
<td>entries</td>
<td>1-based table of the indexes: if non-NULL, then entries(1) contains the first index of the table, entries(2) contains the second index, and so on.</td>
</tr>
</tbody>
</table>

Return Values

Table 53-26 GET_INDEXES Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
</table>
| error_no_such_object  | One of the following:  
  - The package does not exist  
  - The package is not instantiated  
  - The user does not have privileges to debug the package  
  - The object does not exist in the package |
53.7.12 GET_MORE_SOURCE Procedure

When the source does not fit in the buffer provided by the SHOW_SOURCE Procedure version which produced a formatted buffer, this procedure provides additional source.

Syntax

```
DBMS_DEBUG.GET_MORE_SOURCE (
    buffer          IN OUT VARCHAR2,
    buflen          IN BINARY_INTEGER,
    piece#          IN BINARY_INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>The buffer</td>
</tr>
<tr>
<td>buflen</td>
<td>The length of the buffer</td>
</tr>
<tr>
<td>piece#</td>
<td>A value between 2 and the value returned in the parameter pieces from the call to the relevant version of the SHOW_SOURCE Procedures</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure should be called only after the version of SHOW_SOURCE that returns a formatted buffer.

Related Topics

- SHOW_SOURCE Procedures
  The procedure gets the source code. There are two overloaded SHOW_SOURCE procedures.

53.7.13 GET_LINE_MAP Function

This function finds line and entrypoint information about a program so that a debugger can determine the source lines at which it is possible to place breakpoints.

Syntax

```
DBMS_DEBUG.GET_LINE_MAP (
    program               IN   program_info,
    maxline               OUT  BINARY_INTEGER,
    number_of_entry_points OUT  BINARY_INTEGER,
    linemap               OUT  RAW)
RETURN BINARY_INTEGER;
```
Parameters

Table 53-28  GET_LINE_MAP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program</td>
<td>A top-level program unit (procedure / package / function / package body, and so on). Its Namespace, Name, and Owner fields must be initialized, the remaining fields are ignored.</td>
</tr>
<tr>
<td>maxline</td>
<td>The largest source code line number in 'program'</td>
</tr>
<tr>
<td>number_of_entry_points</td>
<td>The number of subprograms in 'program'</td>
</tr>
<tr>
<td>linemap</td>
<td>A bitmap representing the executable lines of 'program'. If line number N is executable, bit number N MOD 8 will be set to 1 at linemap position N / 8. The length of returned linemap is either maxline divided by 8 (plus one if maxline MOD 8 is not zero) or 32767 in the unlikely case of maxline being larger than 32767 * 8.</td>
</tr>
</tbody>
</table>

Return Values

Table 53-29  GET_LINE_MAP Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>A successful completion</td>
</tr>
<tr>
<td>error_no_debug_info</td>
<td>The program unit exists, but has no debug info</td>
</tr>
<tr>
<td>error_bad_handle</td>
<td>No such program unit exists</td>
</tr>
</tbody>
</table>

53.7.14 GET_RUNTIME_INFO Function

This function returns information about the current program. It is only needed if the info_requested parameter to SYNCHRONIZE or CONTINUE was set to 0.

Note:

This is currently only used by client-side PL/SQL.

Syntax

```sql
DBMS_DEBUG.GET_RUNTIME_INFO (  
    info_requested  IN  BINARY_INTEGER,  
    run_info        OUT runtime_info)  
RETURN BINARY_INTEGER;
```
Parameters

Table 53-30   GET_RUNTIME_INFO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info_requested</td>
<td>Which information should be returned in run_info when the program stops (see DBMS_DEBUG Operational Notes for information about information flags)</td>
</tr>
<tr>
<td>run_info</td>
<td>Information about the state of the program</td>
</tr>
</tbody>
</table>

53.7.15 GET_TIMEOUT_BEHAVIOUR Function

This procedure returns the current timeout behavior. This call is made in the target session.

Syntax

```sql
DBMS_DEBUG.GET_TIMEOUT_BEHAVIOUR
RETURN BINARY_INTEGER;
```

Parameters

Table 53-31   GET_TIMEOUT_BEHAVIOUR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oer</td>
<td>The OER (a 4-byte positive number)</td>
</tr>
</tbody>
</table>

Return Values

Table 53-32   GET_TIMEOUT_BEHAVIOUR Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>A successful completion</td>
</tr>
</tbody>
</table>

Information Flags

```sql
info_getOerInfo CONSTANT PLS_INTEGER:= 32;
```

Usage Notes

Less functionality is supported on OER breakpoints than on code breakpoints. In particular, note that:

- No "breakpoint number" is returned - the number of the OER is used instead. Thus it is impossible to set duplicate breakpoints on a given OER (it is a no-op).
- It is not possible to disable an OER breakpoint (although clients are free to simulate this by deleting it).
- OER breakpoints are deleted using delete_oer_breakpoint.
53.7.16 GET_VALUE Function

This function gets a value from the currently-running program. There are two overloaded GET_VALUE functions.

Syntax

DBMS_DEBUG.GET_VALUE (  
  variable_name  IN  VARCHAR2,  
  frame#         IN  BINARY_INTEGER,  
  scalar_value   OUT VARCHAR2,  
  format         IN  VARCHAR2 := NULL)  
RETURN BINARY_INTEGER;

Parameters

Table 53-33  GET_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>Name of the variable or parameter</td>
</tr>
<tr>
<td>frame#</td>
<td>Frame in which it lives; 0 means the current procedure</td>
</tr>
<tr>
<td>scalar_value</td>
<td>Value</td>
</tr>
<tr>
<td>format</td>
<td>Optional date format to use, if meaningful</td>
</tr>
</tbody>
</table>

Return Values

Table 53-34  GET_VALUE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>A successful completion</td>
</tr>
<tr>
<td>error_bogus_frame</td>
<td>Frame does not exist</td>
</tr>
<tr>
<td>error_no_debug_info</td>
<td>Entrypoint has no debug information</td>
</tr>
<tr>
<td>error_no_such_object</td>
<td>variable_name does not exist in frame#</td>
</tr>
<tr>
<td>error_unknown_type</td>
<td>The type information in the debug information is illegible</td>
</tr>
<tr>
<td>error_nullvalue</td>
<td>Value is NULL</td>
</tr>
<tr>
<td>error_indexed_table</td>
<td>The object is a table, but no index was provided</td>
</tr>
</tbody>
</table>

This form of GET_VALUE is for fetching package variables. Instead of a frame#, it takes a handle, which describes the package containing the variable.

Syntax

DBMS_DEBUG.GET_VALUE (  
  variable_name  IN  VARCHAR2,  
  handle         IN  program_info,  
  scalar_value   OUT VARCHAR2,  
  format         IN  VARCHAR2 := NULL)  
RETURN BINARY_INTEGER;
Parameters

Table 53-35  GET_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>Name of the variable or parameter</td>
</tr>
<tr>
<td>handle</td>
<td>Description of the package containing the variable</td>
</tr>
<tr>
<td>scalar_value</td>
<td>Value</td>
</tr>
<tr>
<td>format</td>
<td>Optional date format to use, if meaningful</td>
</tr>
</tbody>
</table>

Return Values

Table 53-36  GET_VALUE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_no_such_object</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- Package does not exist</td>
</tr>
<tr>
<td></td>
<td>- Package is not instantiated</td>
</tr>
<tr>
<td></td>
<td>- User does not have privileges to debug the package</td>
</tr>
<tr>
<td></td>
<td>- Object does not exist in the package</td>
</tr>
<tr>
<td>error_indexed_table</td>
<td>The object is a table, but no index was provided</td>
</tr>
</tbody>
</table>

Examples

This example illustrates how to get the value with a given package PACK in schema SCOTT, containing variable VAR:

```sql
DECLARE
    handle  dbms_debug.program_info;
    resultbuf  VARCHAR2(500);
    retval     BINARY_INTEGER;
BEGIN
    handle.Owner     := 'SCOTT';
    handle.Name      := 'PACK';
    handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
    retval           := dbms_debug.get_value('VAR', handle, resultbuf, NULL);
END;
```

53.7.17 INITIALIZE Function

This function initializes the target session for debugging.

Syntax

```sql
DBMS_DEBUG.INITIALIZE (  
dbg_session_id IN VARCHAR2 := NULL,
  diagnostics IN BINARY_INTEGER := 0)  
RETURN VARCHAR2;
```
Parameters

Table 53-37  INITIALIZE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug_session_id</td>
<td>Name of session ID. If NULL, then a unique ID is generated.</td>
</tr>
<tr>
<td>diagnostics</td>
<td>Indicates whether to dump diagnostic output to the tracefile:</td>
</tr>
<tr>
<td></td>
<td>0 = (default) no diagnostics</td>
</tr>
<tr>
<td></td>
<td>1 = print diagnostics</td>
</tr>
</tbody>
</table>

Return Values

The newly-registered debug session ID (debugID)

Usage Notes

You cannot use DBMS_DEBUG and the JDWP-based debugging interface simultaneously. This call will either fail with an ORA-30677 error if the session is currently being debugged with the JDWP-based debugging interface or, if the call succeeds, any further use of the JDWP-based interface to debug this session will be disallowed.

Calls to DBMS_DEBUG will succeed only if either the caller or the specified debug role carries the DEBUG CONNECT SESSION privilege. Failing that, an ORA-1031 error will be raised. Other exceptions are also possible if a debug role is specified but the password does not match, or if the calling user has not been granted the role, or the role is application-enabled and this call does not originate from within the role-enabling package.

The CREATE ANY PROCEDURE privilege does not affect the visibility of routines through the debugger. A privilege DEBUG for each object has been introduced with a corresponding DEBUG ANY PROCEDURE variant. These are required in order to see routines owned by users other than the session's login user.

Authentication of the debug role and the check for DEBUG CONNECT SESSION privilege will be done in the context of the caller to this routine. If the caller is a definer's rights routine or has been called from one, only privileges granted to the defining user, the debug role, or PUBLIC will be used to check for DEBUG CONNECT SESSION. If this call is from within a definer's rights routine, the debug role, if specified, must be one that has been granted to that definer, but it need not also have been granted to the session login user or be enabled in the calling session at the time the call is made.

The checks made by the debugger after this call is made looking for the DEBUG privilege on individual procedures will be done in the context of the session's login user, the roles that were enabled at session level at the moment this call was made (even if those roles were not available within a definer's rights environment of the call), and the debug role.
53.7.18 PING Procedure

This procedure pings the target session to prevent it from timing out. Use this procedure when execution is suspended in the target session, for example at a breakpoint.

If the `timeoutBehaviour` is set to `retry_on_timeout` then this procedure is not necessary.

**Syntax**

```
DBMS_DEBUG.PING;
```

**Exceptions**

Oracle will display the `no_target_program` exception if there is no target program or if the target session is not currently waiting for input from the debug session.

**Usage Notes**

Timeout options for the target session are registered with the target session by calling `set_timeoutBehaviour`:

- `retry_on_timeout` - Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.
- `continue_on_timeout` - Continue execution, using same event flags.
- `nodebug_on_timeout` - Turn debug-mode OFF (in other words, call `debug_off`) and then continue execution. No more events will be generated by this target session unless it is re-initialized by calling `debug_on`.
- `abort_on_timeout` - Continue execution, using the `abort_execution` flag, which should cause the program to terminate immediately. The session remains in debug-mode.

```
retry_on_timeout CONSTANT BINARY_INTEGER:= 0;
continue_on_timeout CONSTANT BINARY_INTEGER:= 1;
nodebug_on_timeout CONSTANT BINARY_INTEGER:= 2;
abort_on_timeout CONSTANT BINARY_INTEGER:= 3;
```

53.7.19 PRINT_BACKTRACE Procedure

This procedure prints a backtrace listing of the current execution stack. This should only be called if a program is currently running.

There are two overloaded `PRINT_BACKTRACE` procedures.

**Syntax**

```
DBMS_DEBUG.PRINT_BACKTRACE (listing IN OUT VARCHAR2);
DBMS_DEBUG.PRINT_BACKTRACE (backtrace OUT backtrace_table);
```
Parameters

Table 53-38 PRINT_BACKTRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listing</td>
<td>A formatted character buffer with embedded newlines</td>
</tr>
<tr>
<td>backtrace</td>
<td>1-based indexed table of backtrace entries. The currently-running procedure is the last entry in the table (that is, the frame numbering is the same as that used by GET_VALUE). Entry 1 is the oldest procedure on the stack.</td>
</tr>
</tbody>
</table>

53.7.20 PRINT_INSTANTIATIONS Procedure

This procedure returns a list of the packages that have been instantiated in the current session.

Syntax

DBMS_DEBUG.PRINT_INSTANTIATIONS (  
  pkgs   IN OUT NOCOPY backtrace_table,  
  flags  IN BINARY_INTEGER);

Parameters

Table 53-39 PRINT_INSTANTIATIONS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pkgs</td>
<td>The instantiated packages</td>
</tr>
</tbody>
</table>
| flags     | Bitmask of options:  
  - 1 - show specs  
  - 2 - show bodies  
  - 4 - show local instantiations  
  - 8 - show remote instantiations (NYI)  
  - 16 - do a fast job. The routine does not test whether debug information exists or whether the libunit is shrink-wrapped. |

Exceptions

no_target_program - target session is not currently executing

Usage Notes

On return, pkgs contains a program_info for each instantiation. The valid fields are: Namespace, Name, Owner, and LibunitType.

In addition, Line# contains a bitmask of:  
- 1 - the libunit contains debug info  
- 2 - the libunit is shrink-wrapped
53.7.21 PROBE_VERSION Procedure

This procedure returns the version number of DBMS_DEBUG on the server.

Syntax

DBMS_DEBUG.PROBE_VERSION (  
    major out BINARY_INTEGER,  
    minor out BINARY_INTEGER);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>Major version number</td>
</tr>
<tr>
<td>minor</td>
<td>Minor version number: increments as functionality is added</td>
</tr>
</tbody>
</table>

53.7.22 SELF_CHECK Procedure

This procedure performs an internal consistency check. SELF_CHECK also runs a communications test to ensure that the Probe processes are able to communicate.

If SELF_CHECK does not return successfully, then an incorrect version of DBMS_DEBUG was probably installed on this server. The solution is to install the correct version (pbload.sql loads DBMS_DEBUG and the other relevant packages).

Syntax

DBMS_DEBUG.SELF_CHECK (  
    timeout IN binary_integer := 60);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The timeout to use for the communication test. Default is 60 seconds.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OER-6516</td>
<td>Probe version is inconsistent</td>
</tr>
<tr>
<td>pipe_creation_failure</td>
<td>Could not create a pipe</td>
</tr>
<tr>
<td>pipe_send_failure</td>
<td>Could not write data to the pipe</td>
</tr>
<tr>
<td>pipe_receive_failure</td>
<td>Could not read data from the pipe</td>
</tr>
</tbody>
</table>
Table 53-42  (Cont.) SELF_CHECK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe_datatype_mismatch</td>
<td>Datatype in the pipe was wrong</td>
</tr>
<tr>
<td>pipe_data_error</td>
<td>Data got garbled in the pipe</td>
</tr>
</tbody>
</table>

All of these exceptions are fatal. They indicate a serious problem with Probe that prevents it from working correctly.

53.7.23 SET_BREAKPOINT Function

This function sets a breakpoint in a program unit, which persists for the current session.

Execution pauses if the target program reaches the breakpoint.

Syntax

```sql
DBMS_DEBUG.SET_BREAKPOINT (
    program     IN  program_info,
    line#       IN  BINARY_INTEGER,
    breakpoint# OUT BINARY_INTEGER,
    fuzzy       IN  BINARY_INTEGER := 0,
    iterations  IN  BINARY_INTEGER := 0
) RETURN BINARY_INTEGER;
```

Parameters

Table 53-43  SET_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program</td>
<td>Information about the program unit in which the breakpoint is to be set. (In version 2.1 and later, the namespace, name, owner, and dblink may be set to NULL, in which case the breakpoint is placed in the currently-running program unit.)</td>
</tr>
<tr>
<td>line#</td>
<td>Line at which the breakpoint is to be set</td>
</tr>
<tr>
<td>breakpoint#</td>
<td>On successful completion, contains the unique breakpoint number by which to refer to the breakpoint</td>
</tr>
<tr>
<td>fuzzy</td>
<td>Only applicable if there is no executable code at the specified line:</td>
</tr>
<tr>
<td></td>
<td>0 means return error_illegal_line</td>
</tr>
<tr>
<td></td>
<td>1 means search forward for an adjacent line at which to place the breakpoint</td>
</tr>
<tr>
<td></td>
<td>-1 means search backward for an adjacent line at which to place the breakpoint</td>
</tr>
<tr>
<td>iterations</td>
<td>Number of times to wait before signalling this breakpoint</td>
</tr>
</tbody>
</table>
Return Values

Note:
The `fuzzy` and `iterations` parameters are not yet implemented

Table 53-44  SET_BREAKPOINT Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>A successful completion</td>
</tr>
<tr>
<td>error_illegal_line</td>
<td>Cannot set a breakpoint at that line</td>
</tr>
<tr>
<td>error_bad_handle</td>
<td>No such program unit exists</td>
</tr>
</tbody>
</table>

53.7.24 SET_OER_BREAKPOINT Function

This function sets an OER breakpoint.

Syntax

```sql
DBMS_DEBUG.SET_OER_BREAKPOINT (  
    oer  IN PLS_INTEGER)  
RETURN PLS_INTEGER;
```

Parameters

Table 53-45  SET_OER_BREAKPOINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oer</td>
<td>The OER (positive 4-byte number) to set</td>
</tr>
</tbody>
</table>

Return Values

Table 53-46  SET_OER_BREAKPOINT Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>A successful completion</td>
</tr>
<tr>
<td>error_no_such_breakpt</td>
<td>No such OER breakpoint exists</td>
</tr>
</tbody>
</table>

53.7.25 SET_TIMEOUT Function

This function sets the timeout value and returns the new timeout value.

Syntax

```sql
DBMS_DEBUG.SET_TIMEOUT (  
    timeout BINARY_INTEGER)  
RETURN BINARY_INTEGER;
```
Parameters

Table 53-47  SET_TIMEOUT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The timeout to use for communication between the target and debug sessions</td>
</tr>
</tbody>
</table>

53.7.26 SET_TIMEOUT_BEHAVIOUR Procedure

This procedure tells Probe what to do with the target session when a timeout occurs. This call is made in the target session.

Syntax

```
DBMS_DEBUG.SET_TIMEOUT_BEHAVIOUR (  
    behaviour IN PLS_INTEGER);  
```

Parameters

Table 53-48  SET_TIMEOUT_BEHAVIOUR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>behaviour - One of the following:</td>
<td></td>
</tr>
<tr>
<td>retry_on_timeout</td>
<td>Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.</td>
</tr>
<tr>
<td>continue_on_timeout</td>
<td>Continue execution, using same event flags</td>
</tr>
<tr>
<td>nodebug_on_timeout</td>
<td>Turn debug-mode OFF (in other words, call debug_off) and continue execution. No more events will be generated by this target session unless it is re-initialized by calling debug_on.</td>
</tr>
<tr>
<td>abort_on_timeout</td>
<td>Continue execution, using the abort_execution flag, which should cause the program to terminate immediately. The session remains in debug-mode.</td>
</tr>
</tbody>
</table>

Exceptions

unimplemented - the requested behavior is not recognized

Usage Notes

The default behavior (if this procedure is not called) is continue_on_timeout, since it allows a debugger client to reestablish control (at the next event) but does not cause the target session to hang indefinitely.
53.7.27 SET_VALUE Function

This function sets a value in the currently-running program. There are two overloaded
SET_VALUE functions.

Syntax

DBMS_DEBUG.SET_VALUE (
  frame#               IN binary_integer,
  assignment_statement IN varchar2)
RETURN BINARY_INTEGER;

DBMS_DEBUG.SET_VALUE (
  handle               IN program_info,
  assignment_statement IN VARCHAR2)
RETURN BINARY_INTEGER;

Parameters

Table 53-49  SET_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame#</td>
<td>Frame in which the value is to be set; 0 means the currently executing frame.</td>
</tr>
<tr>
<td>handle</td>
<td>Description of the package containing the variable</td>
</tr>
<tr>
<td>assignment_statement</td>
<td>An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, ‘x := 3;’. Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.</td>
</tr>
</tbody>
</table>

Return Values

Table 53-50  SET_VALUE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>-</td>
</tr>
<tr>
<td>error_illegal_value</td>
<td>Not possible to set it to that value</td>
</tr>
<tr>
<td>error_illegal_null</td>
<td>Cannot set to NULL because object type specifies it as ‘not NULL’</td>
</tr>
<tr>
<td>error_value_malformed</td>
<td>Value is not a scalar</td>
</tr>
<tr>
<td>error_name_incomplete</td>
<td>The assignment statement does not resolve to a scalar. For example, ‘x := 3;’, if x is a record.</td>
</tr>
<tr>
<td>error_no_such_object</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- Package does not exist</td>
</tr>
<tr>
<td></td>
<td>- Package is not instantiated</td>
</tr>
<tr>
<td></td>
<td>- User does not have privileges to debug the package</td>
</tr>
<tr>
<td></td>
<td>- Object does not exist in the package</td>
</tr>
</tbody>
</table>
Usage Notes

In some cases, the PL/SQL compiler uses temporaries to access package variables, and does not guarantee to update such temporaries. It is possible, although unlikely, that modification to a package variable using SET_VALUE might not take effect for a line or two.

Examples

To set the value of SCOTT.PACK.var to 6:

DECLARE
    handle dbms_debug.program_info;
    retval BINARY_INTEGER;
BEGIN
    handle.Owner := 'SCOTT';
    handle.Name := 'PACK';
    handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
    retval := dbms_debug.set_value(handle, 'var := 6;');
END;

53.7.28 SHOW_BREAKPOINTS Procedures

There are two overloaded procedures that return a listing of the current breakpoints. There are three overloaded SHOW_BREAKPOINTS procedures.

Syntax

DBMS_DEBUG.SHOW_BREAKPOINTS (  
    listing    IN OUT VARCHAR2);

DBMS_DEBUG.SHOW_BREAKPOINTS (  
    listing    OUT breakpoint_table);

DBMS_DEBUG.SHOW_BREAKPOINTS (  
    code_breakpoints  OUT breakpoint_table,  
    oer_breakpoints   OUT oer_table);

Parameters

Table 53-51  SHOW_BREAKPOINTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listing</td>
<td>A formatted buffer (including newlines) of the breakpoints. Indexed table of breakpoint entries. The breakpoint number is indicated by the index into the table. Breakpoint numbers start at 1 and are reused when deleted.</td>
</tr>
<tr>
<td>code_breakpoints</td>
<td>The indexed table of breakpoint entries, indexed by breakpoint number</td>
</tr>
<tr>
<td>oer_breakpoints</td>
<td>The indexed table of OER breakpoints, indexed by OER</td>
</tr>
</tbody>
</table>
53.7.29 SHOW_FRAME_SOURCE Procedure

The procedure gets the source code. There are two overloaded SHOW_SOURCE procedures.

Syntax

```
DBMS_DEBUG.SHOW_FRAME_SOURCE (  
    first_line  IN            BINARY_INTEGER,
    last_line   IN            BINARY_INTEGER,
    source      IN OUT NOCOPY vc2_table,
    frame_num   IN            BINARY_INTEGER);
```

Parameters

Table 53-52  SHOW_FRAME_SOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_line</td>
<td>Line number of first line to fetch (PL/SQL programs always start at line 1 and have no holes)</td>
</tr>
<tr>
<td>last_line</td>
<td>Line number of last line to fetch. No lines are fetched past the end of the program.</td>
</tr>
<tr>
<td>source</td>
<td>The resulting table, which may be indexed by line#</td>
</tr>
<tr>
<td>frame_num</td>
<td>1-based frame number</td>
</tr>
</tbody>
</table>

Usage Notes

- You use this function only when backtrace shows an anonymous unit is executing at a given frame position and you need to view the source in order to set a breakpoint.
- If frame number is top of the stack and it’s an anonymous block then SHOW_SOURCE can also be used.
- If it’s a stored PL/SQL package/function/procedure then use SQL as described in the Usage Notes to SHOW_SOURCE Procedures.

53.7.30 SHOW_SOURCE Procedures

The procedure gets the source code. There are two overloaded SHOW_SOURCE procedures.

Syntax

```
DBMS_DEBUG.SHOW_SOURCE (  
    first_line  IN   BINARY_INTEGER,
    last_line   IN   BINARY_INTEGER,
    source      OUT  vc2_table);
```

```
DBMS_DEBUG.SHOW_SOURCE (  
    first_line  IN   BINARY_INTEGER,
    last_line   IN   BINARY_INTEGER,
    window      IN   BINARY_INTEGER,
    print_arrow  IN   BINARY_INTEGER,
    buffer      IN OUT VARCHAR2,
    buflen      IN   BINARY_INTEGER,
    pieces      OUT  BINARY_INTEGER);
```
Parameters

Table 53-53  SHOW_SOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_line</td>
<td>Line number of first line to fetch (PL/SQL programs always start at line 1 and have no holes)</td>
</tr>
<tr>
<td>last_line</td>
<td>Line number of last line to fetch. No lines are fetched past the end of the program.</td>
</tr>
<tr>
<td>source</td>
<td>The resulting table, which may be indexed by line#</td>
</tr>
<tr>
<td>window</td>
<td>'Window' of lines (the number of lines around the current source line)</td>
</tr>
<tr>
<td>print_arrow</td>
<td>Nonzero means to print an arrow before the current line</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer in which to place the source listing</td>
</tr>
<tr>
<td>buflen</td>
<td>Length of buffer</td>
</tr>
<tr>
<td>pieces</td>
<td>Set to nonzero if not all the source could be placed into the given buffer</td>
</tr>
</tbody>
</table>

Return Values

An indexed table of source-lines. The source lines are stored starting at first_line. If any error occurs, then the table is empty.

Usage Notes

The best way to get the source code (for a program that is being run) is to use SQL. For example:

```sql
DECLARE
    info DBMS_DEBUG.runtime_info;
BEGIN
    -- call DBMS_DEBUG.SYNCHRONIZE, CONTINUE,
    -- or GET_RUNTIME_INFO to fill in 'info'
    SELECT text INTO <buffer> FROM all_source
    WHERE owner = info.Program.Owner
    AND name  = info.Program.Name
    AND line  = info.Line#;
END;
```

However, this does not work for nonpersistent programs (for example, anonymous blocks and trigger invocation blocks). For nonpersistent programs, call SHOW_SOURCE. There are two flavors: one returns an indexed table of source lines, and the other returns a packed (and formatted) buffer.

The second overloading of SHOW_SOURCE returns the source in a formatted buffer, complete with line-numbers. It is faster than the indexed table version, but it does not guarantee to fetch all the source.

If the source does not fit in bufferlength (buflen), then additional pieces can be retrieved using the GET_MORE_SOURCE procedure (pieces returns the number of additional pieces that need to be retrieved).
53.7.31 SYNCHRONIZE Function

This function waits until the target program signals an event. If info_requested is not NULL, then it calls GET_RUNTIME_INFO.

Syntax

DBMS_DEBUG.SYNCHRONIZE (
    run_info OUT runtime_info,
    info_requested IN BINARY_INTEGER := NULL)
RETURN BINARY_INTEGER;

Parameters

Table 53-54 SYNCHRONIZE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_info</td>
<td>Structure in which to write information about the program. By default, this includes information about what program is running and at which line execution has paused.</td>
</tr>
<tr>
<td>info_requested</td>
<td>Optional bit-field in which to request information other than the default (which is info_getStackDepth + info_getLineInfo). 0 means that no information is requested at all (see DBMS_DEBUG Operational Notes for more about information flags).</td>
</tr>
</tbody>
</table>

Return Values

Table 53-55 SYNCHRONIZE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>A successful completion</td>
</tr>
<tr>
<td>error_timeout</td>
<td>Timed out before the program started execution</td>
</tr>
<tr>
<td>error_communication</td>
<td>Other communication error</td>
</tr>
</tbody>
</table>

53.7.32 TARGET_PROGRAM_RUNNING Procedure

This procedure returns TRUE if the target session is currently executing a stored procedure, or FALSE if it is not.

Syntax

DBMS_DEBUG.TARGET_PROGRAM_RUNNING
RETURN BOOLEAN;
The **DBMS_DEBUG_JDWP** provides the interface to initiate and control the debugging of PL/SQL stored procedures and Java stored procedures over Java Debug Wire Protocol (JDWP).

This chapter contains the following topics:

- **DBMS_DEBUG_JDWP Overview**
- **DBMS_DEBUG_JDWP Security Model**
- **Summary of DBMS_DEBUG_JDWP Subprograms**

### 54.1 DBMS_DEBUG_JDWP Overview

Oracle supports the debugging of PL/SQL stored procedures and Java stored procedures over JDWP. Using the **DBMS_DEBUG_JDWP** package, you can:

- Retrieve the session ID of the current session and serial number
- Connect a database session to a debugger over JDWP
- Set the NLS parameters to change the formats in which PL/SQL program values are represented over JDWP

### 54.2 DBMS_DEBUG_JDWP Security Model

The debugging user invoking a **DBMS_DEBUG_JDWP** subprogram requires a privilege to connect the target database session to the debugger.

**Table 54-1 System Privileges**

<table>
<thead>
<tr>
<th>System Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG CONNECT SESSION</td>
<td>Allows a user to connect his current session to a debugger</td>
</tr>
<tr>
<td>DEBUG CONNECT ANY</td>
<td>Allows a user to connect a session by any logon user to a debugger</td>
</tr>
</tbody>
</table>

In addition, the user can also be granted the following user privilege to debug another user’s session.

**Table 54-2 User Privileges**

<table>
<thead>
<tr>
<th>User Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG CONNECT ON USER &lt;user&gt;</td>
<td>Allows a user to connect any of the specified user’s logon sessions to a debugger to debug another user's session or his own</td>
</tr>
</tbody>
</table>
54.3 Summary DBMS_DEBUG_JDWP Subprograms

This table lists the DBMS_DEBUG_JDWP subprograms and briefly describes them.

### Table 54-3  DBMS_DEBUG_JDWP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT_TCP Procedure</td>
<td>Connects the specified session to the debugger waiting at host:port.</td>
</tr>
<tr>
<td>CURRENT_SESSION_ID Function</td>
<td>Gets the current session's session ID.</td>
</tr>
<tr>
<td>CURRENT_SESSION_SERIAL Function</td>
<td>Gets the current session's session serial number.</td>
</tr>
<tr>
<td>DISCONNECT Procedure</td>
<td>Disconnects the specified session from any debugger with which it is connected.</td>
</tr>
<tr>
<td>GET_NLS_PARAMETER Function</td>
<td>Gets the value of the specified NLS parameter affecting the format in which NUMBER, DATE, TIME (WITH TIME ZONE) and TIMESTAMP (WITH TIME ZONE) runtime values of PL/SQL programs are converted to strings as they are presented through JDWP</td>
</tr>
<tr>
<td>PROCESS_CONNECT_STRING Procedure</td>
<td>Connects a session to a debugger without having to directly modify an application's code.</td>
</tr>
<tr>
<td>SET_NLS_PARAMETER Procedure</td>
<td>Sets the value of the specified NLS parameter affecting the format in which NUMBER, DATE, TIME (WITH TIME ZONE) and TIMESTAMP (WITH TIME ZONE) runtime values of PL/SQL programs are converted to strings as they are presented through JDWP</td>
</tr>
</tbody>
</table>

54.3.1 CONNECT_TCP Procedure

This procedure connects the specified session to the debugger waiting at host:port.

**Syntax**

```sql
DBMS_DEBUG_JDWP.CONNECT_TCP(
    host IN VARCHAR2,
    port IN VARCHAR2,
    session_id IN PLS_INTEGER := NULL,
    session_serial IN PLS_INTEGER := NULL,
    debug_role IN VARCHAR2 := NULL,
    debug_role_pwd IN VARCHAR2 := NULL,
    option_flags IN PLS_INTEGER := 0,
    extensions_cmd_set IN PLS_INTEGER := 128);
```
Parameters

Table 54-4  CONNECT_TCP Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The host name the debugger is waiting at</td>
</tr>
<tr>
<td>port</td>
<td>The port number the debugger is waiting at</td>
</tr>
<tr>
<td>session_id</td>
<td>Session ID</td>
</tr>
<tr>
<td>session_serial</td>
<td>Session number</td>
</tr>
<tr>
<td>debug_role</td>
<td>Debug role</td>
</tr>
<tr>
<td>debug_role_pwd</td>
<td>Debug password</td>
</tr>
<tr>
<td>option_flags</td>
<td>Values:</td>
</tr>
<tr>
<td></td>
<td>• 1: Does not suspend the program until the next client/server request begins. This can be used to hide the startup sequence from end users, who may only want to see their own code</td>
</tr>
<tr>
<td></td>
<td>• 2: Forces the connection even if the session appears to be connected to a debugger. This should best only be specified after some human-interaction confirmation step has occurred; i.e., if an attempt without this option raised ORA-30677, then if the user confirms, retry with this bit set. These may be added together to select multiple option choices.</td>
</tr>
<tr>
<td>extensions_cmd_set</td>
<td>The ID of the Oracle JDWP extension command set</td>
</tr>
</tbody>
</table>

Exceptions

Table 54-5  CONNECT_TCP Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-00022</td>
<td>Invalid session ID</td>
</tr>
<tr>
<td>ORA-01031</td>
<td>Insufficient privilege</td>
</tr>
<tr>
<td>ORA-30677</td>
<td>Session is already connected to a debugger</td>
</tr>
<tr>
<td>ORA-30681</td>
<td>Improper value for argument EXTENSIONS_CMD_SET</td>
</tr>
<tr>
<td>ORA-30682</td>
<td>Improper value for argument OPTION_FLAGS</td>
</tr>
<tr>
<td>ORA-30683</td>
<td>Failure establishing connection to debugger</td>
</tr>
</tbody>
</table>

Usage Notes

- To connect the current session to a debugger, you can pass NULL to both the session_id and session_serial parameters.
To connect a different session, you need to find out its ID and serial. These are available in the `v$plsql_debuggable_sessions` view. The Instance Manager option of Oracle Enterprise Manager is one example of a user interface that displays these values to users. You can also find the values of these for your own session using the `CURRENT_SESSION_ID Function` and `CURRENT_SESSION_SERIAL Function`.

The `debug_role` and `debug_role_pwd` arguments allow the user to name any role as the "debug role", which will be available to privilege checking when checking for permissions to connect the session and when checking permissions available on objects within the debugged session. Both the role and its password are passed here as strings and not as identifiers, so double quotes should not be used but case matters. If the original role name wasn't double-quoted, it should be specified here in upper case.

An `ORA-30677` indicates that the requested session is already being debugged. It is suggested in this case that the user be asked to confirm that (s)he desires to steal the session from the existing connection, and then either an explicit disconnect call or the use of the `connect_force_connect` option bit can be used to allow the connection to succeed on a second attempt. Note that using the `connect_force_connect` bit will avoid the session being allowed to run freely if it is currently suspended through the debugger - in other words, this bit lets you steal a session from one debugger to another without actually disturbing the state of the session.

### 54.3.2 CURRENT_SESSION_ID Function

This function gets the current session's session ID.

**Syntax**

```sql
DBMS_DEBUG_JDWP.CURRENT_SESSION_ID
RETURN PLS_INTEGER;
```

### 54.3.3 CURRENT_SESSION_SERIAL Function

This function gets the current session's session number.

**Syntax**

```sql
DBMS_DEBUG_JDWP.CURRENT_SESSION_SERIAL
RETURN PLS_INTEGER;
```

### 54.3.4 DISCONNECT Procedure

This procedure disconnects the specified session from any debugger with which it is connected.

**Syntax**

```sql
DBMS_DEBUG_JDWP.DISCONNECT(
    session_id        IN  PLS_INTEGER  := NULL,
    session_serial    IN  PLS_INTEGER := NULL);
```
Parameters

Table 54-6  DISCONNECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>Session ID</td>
</tr>
<tr>
<td>session_serial</td>
<td>Session number</td>
</tr>
</tbody>
</table>

Usage Notes

- If the session to disconnect is the current session, the session will be allowed to run freely after disconnecting the debugger. Otherwise, the session will be terminated.
- The same rights are required for this call as for connect, except when disconnecting the current session and the effective user at the time of the call is the same as the login user of the session where no privilege is required.

54.3.5 GET_NLS_PARAMETER Function

This function gets the value of the specified NLS parameter affecting the format in which the NUMBER, DATE, TIME (WITH TIME ZONE) and TIMESTAMP (WITH TIME ZONE) runtime values of PL/SQL programs are converted to strings, as they are presented through JDWP.

These values are private to the current session, but further are private to the debugger mechanisms, separate from the values used to convert values within the debugged program itself.

Syntax

```sql
DBMS_DEBUG_JDWP.GET_NLS_PARAMETER(
    name   IN  VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table 54-7  GET_NLS_PARAMETER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specified NLS parameter</td>
</tr>
</tbody>
</table>

Usage Notes

- When any variable value is read or assigned through JDWP, or when either GET_NLS_PARAMETER Function or SET_NLS_PARAMETER Procedure is first invoked in a session, the debugger mechanisms make a private copy of the then-current NLS_LANGUAGE, NLS_TERRITORY, NLSCALENDAR, NLS_DATE_LANGUAGE, NLS_NUMERIC_CHARACTERS, NLS_TIMESTAMP_FORMAT, NLS_TIMESTAMP_TZ_FORMAT, NLS_TIME_FORMAT and NLS_TIME_TZ_FORMAT values. These private copies may be read using this GET_NLS_PARAMETER Function call and changed using the following call to the SET_NLS_PARAMETER Procedure.
- Once the debugger's private copy of the NLS parameters is established, changes made to the NLS parameters in the current session using the ALTER SESSION statement will
have no effect on the formatting of values as seen through JDWP. To modify the NLS parameters used for JDWP, one must use the `SET_NLS_PARAMETER` Procedure. By the same token, changes made to the debugger’s private copy of the NLS parameters using `SET_NLS_PARAMETER` Procedure will have no effect on the debugged program itself.

- Date values are always formatted for JDWP use using the `NLS_TIMESTAMP_FORMAT`. The default format for `DATE (NLS_DATE_FORMAT)` used in a session most often does not show the time information that is in fact present in the value, and for debugging purposes it seems beneficial to always display that information.

### 54.3.6 PROCESS_CONNECT_STRING Procedure

This procedure connects a session to a debugger in two ways, so that you do not have to directly modify the application code.

The two ways are:

- Using the `ORA_DEBUG_JDWP` environment variable, when running an OCI program
- Setting a web browser cookie called `OWA_DEBUG_<dad>`, when running an application through the PL/SQL Web Gateway

**Syntax**

```sql
DBMS_DEBUG_JDWP.PROCESS_CONNECT_STRING (
    connect_string          IN  VARCHAR2,
    connect_string_type     IN  PLS_INTEGER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect_string</td>
<td>The <code>ORA_DEBUG_JDWP</code> environment variable or <code>OWA_DEBUG_&lt;dad&gt;</code> cookie value that contains the JDWP connection information such as the host and port number of the debugger to connect to</td>
</tr>
<tr>
<td>connect_string_type</td>
<td>Can have the following two values:</td>
</tr>
<tr>
<td></td>
<td>• 1 if the connect string value is retrieved from the <code>ORA_DEBUG_JDWP</code> environment</td>
</tr>
<tr>
<td></td>
<td>• 2 if the value is from the <code>OWA_DEBUG_&lt;dad&gt;</code> cookie</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-00022</td>
<td>Invalid session ID</td>
</tr>
<tr>
<td>ORA-01031</td>
<td>Insufficient privilege</td>
</tr>
<tr>
<td>ORA-30677</td>
<td>Session is already connected to a debugger</td>
</tr>
</tbody>
</table>
Table 54-9  (Cont.) PROCESS_CONNECT_STRING Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-30681</td>
<td>Improper value for argument EXTENSIONS_CMD_SET</td>
</tr>
<tr>
<td>ORA-30682</td>
<td>Improper value for argument OPTION_FLAGS</td>
</tr>
<tr>
<td>ORA-30683</td>
<td>Failure establishing connection to debugger</td>
</tr>
<tr>
<td>ORA-30689</td>
<td>Improper value for environment variable ORA_DEBUG_JDWP</td>
</tr>
</tbody>
</table>

54.3.7 SET_NLS_PARAMETER Procedure

This function sets the value of the specified NLS parameter affecting the format in which NUMBER, DATE, TIME (WITH TIME ZONE) and TIMESTAMP (WITH TIME ZONE) runtime values of PL/SQL programs are converted to strings as they are presented through JDWP.

These values are private to the current session, but further are private to the debugger mechanisms, separate from the values used to convert values within the debugged program itself.

Syntax

```
DBMS_DEBUG_JDWP.SET_NLS_PARAMETER(
    name      IN  VARCHAR2,
    value     IN  VARCHAR2);
```

Parameters

Table 54-10  SET_NLS_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specified NLS parameter</td>
</tr>
<tr>
<td>value</td>
<td>Value of specified NLS parameter</td>
</tr>
</tbody>
</table>

Usage Notes

- When any variable value is read or assigned through JDWP, or when either GET_NLS_PARAMETER Function or SET_NLS_PARAMETER Procedure is first invoked in a session, the debugger mechanisms make a private copy of the then-current NLS_LANGUAGE, NLS_TERRITORY, NLSCALENDAR, NLS_DATE_LANGUAGE, NLS_NUMERIC_CHARACTERS, NLS_TIMESTAMP_FORMAT, NLS_TIMESTAMP_TZ_FORMAT, NLS_TIME_FORMAT and NLS_TIME_TZ_FORMAT values. These private copies may be read by calling the GET_NLS_PARAMETER Function and changed using the following call to the SET_NLS_PARAMETER Procedure.

- Once the debugger's private copy of the NLS parameters is established, changes made to the NLS parameters in the current session using the ALTER SESSION statement will have no effect on the formatting of values as seen through JDWP. To modify the NLS parameters used for JDWP, one must use the SET_NLS_PARAMETER Procedure. By the same token, changes made to the debugger's private copy of the NLS parameters using SET_NLS_PARAMETER Procedure will have no effect on the debugged program itself.
• Date values are always formatted for JDWP use using the `NLS_TIMESTAMP_FORMAT`. The default format for `DATE` (NLS_DATE_FORMAT) used in a session most often does not show the time information that is in fact present in the value, and for debugging purposes it seems beneficial to always display that information.
The `DBMS_DEBUG_JDWP_CUSTOM` package provides database users a means to perform custom handling of a debug connection request of a database session to a debugger using the Java Debug Wire Protocol (JDWP).

This chapter contains the following topics:

• `DBMS_DEBUG_JDWP_CUSTOM` Overview
• `DBMS_DEBUG_JDWP_CUSTOM` Security Model
• Summary of `DBMS_DEBUG_JDWP_CUSTOM` Subprograms

### 55.1 DBMS_DEBUG_JDWP_CUSTOM Overview

`DBMS_DEBUG_JDWP_CUSTOM` package is invoked when you attempt to connect your database session to a debugger, using the Java Debug Wire Protocol (JDWP). Before connecting to the database, you must set the `ORA_DEBUG_JDWP` environment variable in your client-side OCI application. Defining this package in your schema enables you to perform custom actions before connecting your database session to the debugger.

### 55.2 DBMS_DEBUG_JDWP_CUSTOM Security Model

This default implementation of the `DBMS_DEBUG_JDWP_CUSTOM` package does not perform any additional security checks and only invokes the `DBMS_DEBUG_JDWP` package for making the debug connection. When the `DBMS_DEBUG_JDWP` package is invoked, it checks for the necessary `DEBUG CONNECT` system or user privilege before making the connection.

A database user who wants to perform additional custom security checks, must perform the following steps to override this default implementation:

• Define the `DBMS_DEBUG_JDWP_CUSTOM` package in the user’s own schema.
• Implement the check before invoking the `DBMS_DEBUG_JDWP` package in the user’s local copy of the package.

### 55.3 Summary DBMS_DEBUG_JDWP_CUSTOM Subprograms

This table summarizes the `DBMS_DEBUG_JDWP_CUSTOM` subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT_DEBUGGER Procedure</td>
<td>This procedure is invoked to handle the debug connection request of the current database session to the debugger.</td>
</tr>
</tbody>
</table>
55.3.1 CONNECT_DEBUGGER Procedure

This procedure is invoked to handle the debug connection request of the current database session to the debugger.

Syntax

```sql
PROCEDURE CONNECT_DEBUGGER
(
    HOST                VARCHAR2,
    PORT                VARCHAR2,
    DEBUG_ROLE          VARCHAR2 := NULL,
    DEBUG_ROLE_PWD      VARCHAR2 := NULL,
    OPTION_FLAGS        PLS_INTEGER := 0,
    EXTENSIONS_CMD_SET  PLS_INTEGER := 128
)
```

Parameters

The preceding parameters are for the default implementation of the procedure for handling the debug connection request. A user, who wants to customize the handling of the request, must override this default implementation of the DBMS_DEBUG_JDWP_CUSTOM package procedure by defining the package (specification and body) with a procedure of the same name in the user's own schema. The user may customize the number and names of the arguments, but must have the same name for the package and the procedure, namely, DBMS_DEBUG_JDWP_CUSTOM and CONNECT_DEBUGGER respectively. The user's customized version of the package may contain overloaded versions of the CONNECT_DEBUGGER procedure with different arguments. All of the arguments to the custom package procedure must either be of VARCHAR2 type or of types that PL/SQL can implicitly convert from VARCHAR2.

For example, if a user wants to disallow debugging outside business hours, while applying optional NLS settings to the program values displayed during debugging, then the user can define a custom implementation of the package in the user's schema in the following way:

```sql
CREATE OR REPLACE PACKAGE DBMS_DEBUG_JDWP_CUSTOM AUTHID CURRENT_USER IS
    PROCEDURE CONNECT_DEBUGGER(HOST VARCHAR2,
        PORT VARCHAR2,
        NLS_LANGUAGE VARCHAR2 DEFAULT NULL,
        NLS_TERRITORY VARCHAR2 DEFAULT NULL);
END;
/

CREATE OR REPLACE PACKAGE BODY DBMS_DEBUG_JDWP_CUSTOM IS

    PROCEDURE CONNECT_DEBUGGER(HOST VARCHAR2,
        PORT VARCHAR2,
        NLS_LANGUAGE VARCHAR2 DEFAULT NULL,
        NLS_TERRITORY VARCHAR2 DEFAULT NULL) IS
    BEGIN
        SELECT EXTRACT(HOUR FROM LOCALTIMESTAMP) INTO HOUR FROM DUAL;
    END;
END;
```

Chapter 55

Summary DBMS_DEBUG_JDWP_CUSTOM Subprograms

55-2
IF (HOUR < 9 OR HOUR > 5) THEN
    RAISE_APPLICATION_ERROR(-20000,
        'Debug connection disallowed outside business hours');
ELSE
    IF (NLS_LANGUAGE IS NOT NULL) THEN
        DBMS_DEBUG_JDWP.SET_NLS_PARAMETER('NLS_LANGUAGE', NLS_LANGUAGE);
        END IF;
    IF (NLS_TERRITORY IS NOT NULL) THEN
        DBMS_DEBUG_JDWP.SET_NLS_PARAMETER('NLS_TERRITORY', NLS_TERRITORY);
        END IF;
    DBMS_DEBUG_JDWP.CONNECT_TCP(HOST, PORT);
    END IF;
END IF;
END;
/

Before executing the client application for passing arguments to the procedure parameters, the user must set the ORA_DEBUG_JDWP environment variable as follows:

> setenv ORA_DEBUG_JDWP "host=123.45.67.89;port=4000;nls_language=american"
> myapp -user HR/<password>
...
You can use the `DBMS_DESCRIBE` package to get information about a PL/SQL object. When you specify an object name, `DBMS_DESCRIBE` returns a set of indexed tables with the results. Full name translation is performed and security checking is also checked on the final object.

This chapter contains the following topics:

- Overview
- Security Model
- Types
- Exceptions
- Examples
- Summary of `DBMS_DESCRIBE` Subprograms

### 56.1 DBMS_DESCRIBE Overview

This package provides the same functionality as the Oracle Call Interface `OCIDescribeAny` call.

**See Also:**

`Oracle Call Interface Programmer's Guide`

### 56.2 DBMS_DESCRIBE Security Model

This package is available to `PUBLIC` and performs its own security checking based on the schema object being described.

### 56.3 DBMS_DESCRIBE Types

The `DBMS_DESCRIBE` package declares two PL/SQL table types, which are used to hold data returned by `DESCRIBE_PROCEDURE` in its `OUT` parameters.

The types are:

```sql
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(30)
   INDEX BY BINARY_INTEGER;

TYPE NUMBER_TABLE IS TABLE OF NUMBER
   INDEX BY BINARY_INTEGER;
```
56.4 DBMS_DESCRIBE Exceptions

DBMS_DESCRIBE can raise application errors in the range -20000 to -20004.

Table 56-1 DBMS_DESCRIBE Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000</td>
<td>ORU 10035: cannot describe a package ('X') only a procedure within a package.</td>
</tr>
<tr>
<td>ORA-20001</td>
<td>ORU-10032: procedure 'X' within package 'Y' does not exist.</td>
</tr>
<tr>
<td>ORA-20002</td>
<td>ORU-10033: object 'X' is remote, cannot describe; expanded name 'Y'.</td>
</tr>
<tr>
<td>ORA-20003</td>
<td>ORU-10036: object 'X' is invalid and cannot be described.</td>
</tr>
<tr>
<td>ORA-20004</td>
<td>Syntax error attempting to parse 'X'.</td>
</tr>
</tbody>
</table>

56.5 DBMS_DESCRIBE Examples

One use of the DESCRIBE_PROCEDURE procedure is as an external service interface.

For example, consider a client that provides an OBJECT_NAME of SCOTT.ACCOUNT_UPDATE, where ACCOUNT_UPDATE is an overloaded function with specification:

```
TABLE account (accnt_no NUMBER, person_id NUMBER,
                balance NUMBER(7,2))
TABLE person  (person_id number(4), person_nm varchar2(10))

CREATE OR REPLACE PACKAGE ACCOUNT_PKG is
  FUNCTION ACCOUNT_UPDATE (accnt_no NUMBER,
                           person person%rowtype,
                           amounts DBMS_DESCRIBE.NUMBER_TABLE,
                           trans_date DATE)
  return account.balance%type;

FUNCTION ACCOUNT_UPDATE (accnt_no NUMBER,
                         person person%rowtype,
                         amounts DBMS_DESCRIBE.NUMBER_TABLE,
                         trans_no NUMBER)
  return account.balance%type;
END;
```

This procedure might look similar to the following output:

```
overload position  argument level  datatype length prec scale rad
------------- ------ -------- ------ -------- ------ ---- ----- ---
1              0         2     22    7     2  10
1              1   ACCNT_NO  0         2     0     0     0     0
1              2   PERSON   0   250      0    0     0     0
1              1  PERSON_ID 1   2     22    4     0  10
1              2  PERSON_NM 1   10      0     0     0     0
1              3  AMOUNTS   0   251      0     0     0     0
1              1           2     22    7     2  10
1              4  TRANS_DATE 0   12      0     0     0     0
2              0         2     22    7     2  10
```
The following PL/SQL procedure has as its parameters all of the PL/SQL datatypes:

```sql
CREATE OR REPLACE PROCEDURE p1 (  
pvc2    IN     VARCHAR2,  
pvc     OUT    VARCHAR,  
pstr    IN OUT STRING,  
plong   IN     LONG,  
prowid  IN     ROWID,  
pchara  IN     CHARACTER,  
pchar   IN     CHAR,  
praw    IN     RAW,  
plraw   IN     LONG RAW,  
pbinint IN     BINARY_INTEGER,  
pplsiint IN     PLS_INTEGER,  
pbool   IN     BOOLEAN,  
pnat    IN     NATURAL,  
ppos    IN     POSITIVE,  
pposn   IN     POSITIVEN,  
pnatn   IN     NATURALN,  
pnum    IN     NUMBER,  
pintgr  IN     INTEGER,  
pint    IN     INT,  
psmall  IN     SMALLINT,  
pdec    IN     DECIMAL,  
preal   IN     REAL,  
pfloat  IN     FLOAT,  
pnumer  IN     NUMERIC,  
pdp     IN     DOUBLE PRECISION,  
pdate   IN     DATE,  
pmls    IN     MLSLABEL) AS
BEGIN  
    NULL;  
END;
```

If you describe this procedure using the following:

```sql
CREATE OR REPLACE PACKAGE describe_it AS  
PROCEDURE desc_proc (name VARCHAR2);  
END describe_it;  
CREATE OR REPLACE PACKAGE BODY describe_it AS  
PROCEDURE prt_value(val VARCHAR2, isize INTEGER) IS  
n INTEGER;  
BEGIN  
n := isize - LENGTHB(val);  
IF n < 0 THEN  
n := 0;  
END IF;  
DBMS_OUTPUT.PUT(val);  
FOR i in 1..n LOOP  
    DBMS_OUTPUT.PUT(' ');  
END LOOP;  
END prt_value;
```
PROCEDURE desc_proc (name VARCHAR2) IS

  overload    DBMS_DESCRIBE.NUMBER_TABLE;
  position    DBMS_DESCRIBE.NUMBER_TABLE;
  c_level     DBMS_DESCRIBE.NUMBER_TABLE;
  arg_name    DBMS_DESCRIBE.VARCHAR2_TABLE;
  dty         DBMS_DESCRIBE.NUMBER_TABLE;
  def_val     DBMS_DESCRIBE.NUMBER_TABLE;
  p_mode      DBMS_DESCRIBE.NUMBER_TABLE;
  length      DBMS_DESCRIBE.NUMBER_TABLE;
  precision   DBMS_DESCRIBE.NUMBER_TABLE;
  scale       DBMS_DESCRIBE.NUMBER_TABLE;
  radix       DBMS_DESCRIBE.NUMBER_TABLE;
  spare       DBMS_DESCRIBE.NUMBER_TABLE;
  idx         INTEGER := 0;

BEGIN
  DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
    name,
    null,
    null,
    overload,
    position,
    c_level,
    arg_name,
    dty,
    def_val,
    p_mode,
    length,
    precision,
    scale,
    radix,
    spare);

  DBMS_OUTPUT.PUT_LINE('Position    Name        DTY  Mode');
  LOOP
    idx := idx + 1;
    prt_value(TO_CHAR(position(idx)), 12);
    prt_value(arg_name(idx), 12);
    prt_value(TO_CHAR(dty(idx)), 5);
    prt_value(TO_CHAR(p_mode(idx)), 5);
    DBMS_OUTPUT.NEW_LINE;
  END LOOP;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.NEW_LINE;
  END desc_proc;
END describe_it;

Then the results list all the numeric codes for the PL/SQL datatypes:

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Datatype_Code</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PVC2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>PVC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>PSTR</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>PLONG</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>
56.6 Summary of DBMS_DESCRIBE Subprograms

The DBMS_DESCRIBE package includes the DESCRIBE_PROCEDURE procedure.

Table 56-2 DBMS_DESCRIBE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE_PROCEDURE</td>
<td>Provides a brief description of a PL/SQL stored procedure</td>
</tr>
</tbody>
</table>

56.6.1 DESCRIBE_PROCEDURE Procedure

The procedure DESCRIBE_PROCEDURE provides a brief description of a PL/SQL stored procedure.

It takes the name of a stored procedure and returns information about each parameter of that procedure.

Syntax

```sql
DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
  object_name       IN   VARCHAR2,
  reserved1         IN   VARCHAR2,
  reserved2         IN   VARCHAR2,
  overload          OUT  NUMBER_TABLE,
  position          OUT  NUMBER_TABLE,
  level             OUT  NUMBER_TABLE,
  argument_name     OUT  VARCHAR2_TABLE,
  datatype          OUT  NUMBER_TABLE,
  default_value     OUT  NUMBER_TABLE,
  in_out            OUT  NUMBER_TABLE,
  length            OUT  NUMBER_TABLE,
  precision         OUT  NUMBER_TABLE,
)```

---

56-5
Parameters

Table 56-3  DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>Name of the procedure being described. The syntax for this parameter follows the rules used for identifiers in SQL. The name can be a synonym. This parameter is required and may not be null. The total length of the name cannot exceed 197 bytes. An incorrectly specified OBJECT_NAME can result in one of the following exceptions: ORA-20000 - A package was specified. You can only specify a stored procedure, stored function, packaged procedure, or packaged function. ORA-20001 - The procedure or function that you specified does not exist within the given package. ORA-20002 - The object that you specified is a remote object. This procedure cannot currently describe remote objects. ORA-20003 - The object that you specified is invalid and cannot be described. ORA-20004 - The object was specified with a syntax error.</td>
</tr>
<tr>
<td>reserved1</td>
<td>Reserved for future use -- must be set to NULL or the empty string.</td>
</tr>
<tr>
<td>reserved2</td>
<td></td>
</tr>
<tr>
<td>overload</td>
<td>A unique number assigned to the procedure's signature. If a procedure is overloaded, then this field holds a different value for each version of the procedure.</td>
</tr>
<tr>
<td>position</td>
<td>Position of the argument in the parameter list. Position 0 returns the values for the return type of a function.</td>
</tr>
<tr>
<td>level</td>
<td>If the argument is a composite type, such as record, then this parameter returns the level of the datatype. See the Oracle Call Interface Programmer's Guide for a description of the ODESP call for an example.</td>
</tr>
<tr>
<td>argument_name</td>
<td>Name of the argument associated with the procedure that you are describing.</td>
</tr>
</tbody>
</table>
### Table 56-3  (Cont.) DBMS_DESCRIBE.DESCRIPTEDURE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datatype</td>
<td>Oracle datatype of the argument being described. The datatypes and their numeric type codes are:</td>
</tr>
<tr>
<td></td>
<td>0 placeholder for procedures with no arguments</td>
</tr>
<tr>
<td></td>
<td>1 VARCHAR, VARCHAR, STRING</td>
</tr>
<tr>
<td></td>
<td>2 NUMBER, INTEGER, SMALLINT, REAL, FLOAT, DECIMAL</td>
</tr>
<tr>
<td></td>
<td>3 BINARY_INTEGER, PLS_INTEGER, POSITIVE, NATURAL</td>
</tr>
<tr>
<td></td>
<td>8 LONG</td>
</tr>
<tr>
<td></td>
<td>11 ROWID</td>
</tr>
<tr>
<td></td>
<td>12 DATE</td>
</tr>
<tr>
<td></td>
<td>23 RAW</td>
</tr>
<tr>
<td></td>
<td>24 LONG RAW</td>
</tr>
<tr>
<td></td>
<td>58 OPAQUE TYPE</td>
</tr>
<tr>
<td></td>
<td>96 CHAR (ANSI FIXED CHAR), CHARACTER</td>
</tr>
<tr>
<td></td>
<td>106 MLSLABEL</td>
</tr>
<tr>
<td></td>
<td>121 OBJECT</td>
</tr>
<tr>
<td></td>
<td>122 NESTED TABLE</td>
</tr>
<tr>
<td></td>
<td>123 VARRAY</td>
</tr>
<tr>
<td></td>
<td>178 TIME</td>
</tr>
<tr>
<td></td>
<td>179 TIME WITH TIME ZONE</td>
</tr>
<tr>
<td></td>
<td>180 TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>181 TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td></td>
<td>231 TIMESTAMP WITH LOCAL TIME ZONE</td>
</tr>
<tr>
<td></td>
<td>250 PL/SQL RECORD</td>
</tr>
<tr>
<td></td>
<td>251 PL/SQL TABLE</td>
</tr>
<tr>
<td></td>
<td>252 PL/SQL BOOLEAN</td>
</tr>
<tr>
<td>default_value</td>
<td>1 if the argument being described has a default value; otherwise, the value is 0.</td>
</tr>
<tr>
<td>in_out</td>
<td>Describes the mode of the parameter:</td>
</tr>
<tr>
<td></td>
<td>0 IN</td>
</tr>
<tr>
<td></td>
<td>1 OUT</td>
</tr>
<tr>
<td></td>
<td>2 IN OUT</td>
</tr>
<tr>
<td>length</td>
<td>For &amp;rowtype formal arguments, the length constraint is returned, otherwise 0 is returned. If the include_string_constraints parameter is set to TRUE, the argument's formal length constraint is passed back if it is of the appropriate type. Those are the string types: 1;8;23;24;96</td>
</tr>
<tr>
<td>precision</td>
<td>If the argument being described is of datatype 2 (NUMBER), then this parameter is the precision of that number.</td>
</tr>
<tr>
<td>scale</td>
<td>If the argument being described is of datatype 2 (NUMBER), then this parameter is the scale of that number.</td>
</tr>
<tr>
<td>radix</td>
<td>If the argument being described is of datatype 2 (NUMBER), then this parameter is the radix of that number.</td>
</tr>
<tr>
<td>spare</td>
<td>Reserved for future functionality.</td>
</tr>
<tr>
<td>include_string_constraints</td>
<td>The default is FALSE. If the parameter is set to TRUE, the arguments' formal type constraints is passed back if it is of the appropriate type. Those are the string types: 1;8;23;24;96</td>
</tr>
</tbody>
</table>
Return Values

All values from `DESCRIBE_PROCEDURE` are returned in its `OUT` parameters. The datatypes for these are PL/SQL tables, to accommodate a variable number of parameters.
The `DBMS_DG` package allows applications to notify the primary database or the fast-start failover target database in an Oracle Data Guard broker environment to initiate a fast-start failover when the application encounters a condition that warrants a failover.

This chapter contains the following topics:

- Using `DBMS_DG`
- Security Model
- Summary of the `DBMS_DG` Subprogram

**Note:**

If you are working in a multitenant container database (CDB), then functions within `DBMS_DG` are only executed at the root level. Ensure you are connected at the root level, not at the individual pluggable database (PDB) level.

**57.2 DBMS_DG Security Model**

The `DBMS_DG` package runs with invoker’s rights and requires the `SYSDBA` privilege.
57.3 Summary of the DBMS_DG Subprogram

The DBMS_DG package contains one subprogram, the INITIATE_FS_FAILOVER procedure.

Table 57-1  DBMS_DG Package Subprogram

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIATE_FS_FAILOVER Procedure</td>
<td>Enables an application to notify either the primary or fast-start failover target standby database that a fast-start failover is necessary when the application encounters conditions that warrant a failover. This procedure can only be called while connected to a primary database or a fast-start failover standby database.</td>
</tr>
</tbody>
</table>

57.3.1 INITIATE_FS_FAILOVER Procedure

Use this procedure to specify a condition string that, when encountered by an application, allows the application to request that a fast-start failover be invoked.

Syntax

```sql
DBMS_DG.INITIATE_FS_FAILOVER (condstr IN VARCHAR2) RETURN BINARY_INTEGER;
```

Parameters

Table 57-2  INITIATE_FS_FAILOVER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>condstr</td>
<td>Specifies the condition string for which a fast-start failover should be requested. If no condition string argument is supplied, the default string of &quot;Application Failover Requested&quot; will be logged in the broker log file and in the database alert log of the database on which the procedure was called.</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure returns a binary integer.
- Query the V$FS_FAILOVER_STATS view to see the time of the last fast-start failover and the reason it was performed.
- This procedure can only be called while connected to a primary database or a fast-start failover standby database.
Errors

Table 57-3  INITIATE_FS_FAILOVER Procedure Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-00000: normal, successful completion</td>
<td>The request to initiate a fast-start failover has been posted to the observer.</td>
</tr>
<tr>
<td>ORA-16646: fast-start failover is disabled</td>
<td>Either a broker configuration does not exist or fast-start failover has not been enabled.</td>
</tr>
<tr>
<td>ORA-16666: unable to initiate fast-start failover on a bystander standby database</td>
<td>DBMS_DG.INITIATE_FS_FAILOVER was invoked on a bystander standby database. That is, it was not invoked on the primary or on the fast-start failover target standby database.</td>
</tr>
<tr>
<td>ORA-16817: unsynchronized fast-start failover configuration</td>
<td>DBMS_DG.INITIATE_FS_FAILOVER was invoked in a maximum available fast-start failover configuration when the configuration was not synchronized.</td>
</tr>
<tr>
<td>ORA-16819: fast-start failover observer not started</td>
<td>DBMS_DG.INITIATE_FS_FAILOVER was invoked but an observer had not yet been started.</td>
</tr>
<tr>
<td>ORA-16820: fast-start failover observer is no longer observing this database</td>
<td>DBMS_DG.INITIATE_FS_FAILOVER was invoked but the configuration detects that the observer may not be running.</td>
</tr>
<tr>
<td>ORA-16829: lagging fast-start failover configuration</td>
<td>DBMS_DG.INITIATE_FS_FAILOVER was invoked in a maximum performance fast-start failover configuration when the configuration was not in the user-specified redo lag limit.</td>
</tr>
</tbody>
</table>

Example

In this example, the program attempts to initiate a fast-start failover when fast-start failover is disabled. To use this example, connect as user SYSG with SYSDBA privileges.

```sql
set serveroutput on
declare
status integer;
begin
status := dbms_dg.initiate_fs_failover('Failover Requested');

dbms_output.put_line('Fast-Start Failover is disabled: Expected status = ORA-16646');
dbms_output.put_line('Actual Status = ORA-' || status);
end;
/
exit;
```
DBMS_DIMENSION enables you to verify dimension relationships and provides an alternative to the Enterprise Manager Dimension Wizard for displaying a dimension definition.

See Also:
Oracle Database Data Warehousing Guide for detailed conceptual and usage information about the DBMS_DIMENSION package

This chapter contains the following topics:

• Security Model
• Summary of DBMS_DIMENSION Subprograms

58.1 DBMS_DIMENSION Security Model

Security on this package can be controlled by granting EXECUTE to selected users or roles.

A user can validate or describe all the dimensions in his own schema. To validate or describe a dimension in another schema, you must have either an object privilege on the dimension or one of the following system privileges: CREATE ANY DIMENSION, ALTER ANY DIMENSION, and DROP ANY DIMENSION.

58.2 Summary of DBMS_DIMENSION Subprograms

This table lists the DBMS_DIMENSION subprograms and briefly describes them.

Table 58-1  DBMS_DIMENSION Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE_DIMENSION Procedure</td>
<td>Prints out the definition of the input dimension, including dimension owner and name, levels, hierarchies, and attributes</td>
</tr>
<tr>
<td>VALIDATE_DIMENSION Procedure</td>
<td>Verifies that the relationships specified in a dimension are correct</td>
</tr>
</tbody>
</table>
58.2.1 DESCRIBE_DIMENSION Procedure

This procedure displays the definition of the dimension, including dimension name, levels, hierarchies, and attributes. It displays the output using the `DBMS_OUTPUT` package.

**Syntax**

```sql
DBMS_DIMENSION.DESCRIBE_DIMENSION (
  dimension  IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>The owner and name of the dimension in the format of <code>owner.name</code>.</td>
</tr>
</tbody>
</table>

58.2.2 VALIDATE_DIMENSION Procedure

This procedure verifies that the relationships specified in a dimension are valid. The rowid for any row that is found to be invalid will be stored in the table `DIMENSION_EXCEPTIONS` in the user's schema.

**Syntax**

```sql
DBMS_DIMENSION.VALIDATE_DIMENSION (
  dimension               IN VARCHAR2,
  incremental             IN BOOLEAN := TRUE,
  check_nulls             IN BOOLEAN := FALSE,
  statement_id            IN VARCHAR2 := NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>The owner and name of the dimension in the format of <code>owner.name</code>.</td>
</tr>
<tr>
<td>incremental</td>
<td>If TRUE, check only the new rows for tables of this dimension. If FALSE, check all the rows.</td>
</tr>
<tr>
<td>check_nulls</td>
<td>If TRUE, then all level columns are verified to be non-null. If FALSE, this check is omitted. Specify FALSE when non-NULLness is guaranteed by other means, such as NOT NULL constraints.</td>
</tr>
<tr>
<td>statement_id</td>
<td>A client-supplied unique identifier to associate output rows with specific invocations of the procedure.</td>
</tr>
</tbody>
</table>
DBMS_DISTRIBUTED_TRUST_ADMIN procedures maintain the Trusted Servers List. Use these procedures to define whether a server is trusted. If a database is not trusted, Oracle refuses current user database links from the database.

This chapter contains the following topics:

- Overview
- Security Model
- Examples
- Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms

59.1 DBMS_DISTRIBUTED_TRUST_ADMIN Overview

Oracle uses local Trusted Servers Lists, along with enterprise domain membership lists stored in the enterprise LDAP directory service, to determine if another database is trusted. The LDAP directory service entries are managed with the Enterprise Security Manager Tool in Oracle Enterprise Manager.

Oracle considers another database to be "trusted" if it meets the following criteria:

1. It is in the same enterprise domain in the directory service as the local database.
2. The enterprise domain is marked as trusted in the directory service.
3. It is not listed as untrusted in the local Trusted Servers List. Current user database links will only be accepted from another database if both databases involved trust each other.

You can list a database server locally in the Trusted Servers List regardless of what is listed in the directory service. However, if you list a database that is not in the same domain as the local database, or if that domain is untrusted, the entry will have no effect.

This functionality is part of the Enterprise User Security feature of the Oracle Advanced Security Option.

59.2 DBMS_DISTRIBUTED_TRUST_ADMIN Security Model

To execute DBMS_DISTRIBUTED_TRUST_ADMIN, the EXECUTE_CATALOG_ROLE role must be granted to the DBA. To select from the view TRUSTED_SERVERS, the SELECT_CATALOG_ROLE role must be granted to the DBA.

It is important to know whether all servers are trusted or not trusted. Trusting a particular server with the ALLOW_SERVER procedure does not have any effect if the database already trusts all databases, or if that database is already trusted. Similarly, denying a particular server with the DENY_SERVER procedure does not have any effect if the database already does not trust any database or if that database is already untrusted.
The procedures `DENY_ALL` and `ALLOW_ALL` delete all entries (in other words, server names) that are explicitly allowed or denied using the `ALLOW_SERVER` procedure or `DENY_SERVER` procedure respectively.

59.3 DBMS_DISTRIBUTED_TRUST_ADMIN Examples

If you have not yet used the package `DBMS_DISTRIBUTED_TRUST_ADMIN` to change the trust listing, by default you trust all databases in the same enterprise domain if that domain it listed as trusted in the directory service:

```sql
SELECT * FROM TRUSTED_SERVERS;
```

```
NAME
--------- ---------------------
Trusted
All
```

Because all servers are currently trusted, you can execute the `DENY_SERVER Procedure` and specify that a particular server is not trusted:

```sql
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER ('SALES.US.AMERICAS.ACME_AUTO.COM');
```

PL/SQL procedure successfully completed.

```sql
SELECT * FROM TRUSTED_SERVERS;
```

```
NAME
--------- -----------------------------------------------
Untrusted SALES.US.AMERICAS.ACME_AUTO.COM
```

By executing the `DENY_ALL Procedure`, you can choose to not trust any database server:

```sql
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

PL/SQL procedure successfully completed.

```sql
SELECT * FROM TRUSTED_SERVERS;
```

```
NAME
--------- -----------------------------------------------
Untrusted
All
```

The `ALLOW_SERVER Procedure` can be used to specify that one particular database is to be trusted:

```sql
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER ('SALES.US.AMERICAS.ACME_AUTO.COM');
```

PL/SQL procedure successfully completed.

```sql
SELECT * FROM TRUSTED_SERVERS;
```

```
NAME
--------- ------------------------------------------------
Trusted   SALES.US.AMERICAS.ACME_AUTO.COM
```
59.4 Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms

This table lists and briefly describes the DBMS_DISTRIBUTED_TRUST_ADMIN subprograms.

Table 59-1  DBMS_DISTRIBUTED_TRUST_ADMIN Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOW_ALL Procedure</td>
<td>Empties the list and inserts a row indicating that all servers should be trusted</td>
</tr>
<tr>
<td>ALLOW_SERVER Procedure</td>
<td>Enables a specific server to be allowed access even though deny all is indicated in the list</td>
</tr>
<tr>
<td>DENY_ALL Procedure</td>
<td>Empties the list and inserts a row indicating that all servers should be untrusted</td>
</tr>
<tr>
<td>DENY_SERVER Procedure</td>
<td>Enables a specific server to be denied access even though allow all is indicated in the list</td>
</tr>
</tbody>
</table>

59.4.1 ALLOW_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers that are members of a trusted domain in an enterprise directory service and that are in the same domain are allowed access.

The view TRUSTED_SERVERS will show "TRUSTED ALL" indicating that the database trusts all servers that are currently trusted by the enterprise directory service.

Syntax

DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ALL;

Usage Notes

ALLOW_ALL only applies to servers listed as trusted in the enterprise directory service and in the same enterprise domain.

59.4.2 ALLOW_SERVER Procedure

This procedure ensures that the specified server is considered trusted (even if you have previously specified "deny all").

Syntax

DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER (server IN VARCHAR2);
### Parameters

#### Table 59-2  ALLOW_SERVER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Unique, fully-qualified name of the server to be trusted.</td>
</tr>
</tbody>
</table>

**Usage Notes**

If the Trusted Servers List contains the entry "deny all", then this procedure adds a specification indicating that a specific database (for example, DBx) is to be trusted.

If the Trusted Servers List contains the entry "allow all", and if there is no "deny DBx" entry in the list, then executing this procedure causes no change.

If the Trusted Servers List contains the entry "allow all", and if there is a "deny DBx" entry in the list, then that entry is deleted.

#### 59.4.3 DENY_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers are denied access.

The view **TRUSTED_SERVERS** will show "UNTRUSTED ALL" indicating that no servers are currently trusted.

**Syntax**

```sql
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

#### 59.4.4 DENY_SERVER Procedure

This procedure ensures that the specified server is considered untrusted (even if you have previously specified allow all).

**Syntax**

```sql
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER (server IN VARCHAR2);
```

### Parameters

#### Table 59-3  DENY_SERVER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Unique, fully-qualified name of the server to be untrusted.</td>
</tr>
</tbody>
</table>

**Usage Notes**

If the Trusted Servers List contains the entry allow all, then this procedure adds an entry indicating that the specified database (for example, DBx) is not to be trusted.
If the Trusted Servers List contains the entry "deny all", and if there is no "allow DBx" entry in the list, then this procedure causes no change.

If the Trusted Servers List contains the entry "deny all", and if there is an "allow DBx" entry, then this procedure causes that entry to be deleted.
The DBMS_DNFS package provides an interface to assist in creating a database using files in the backup set.

This chapter contains the following topics:

- Security Model
- Summary of DBMS_DNFS Subprograms

See Also:
Oracle Database Administrator's Guide

60.1 DBMS_DNFS Security Model

This package has to be executed by users with SYSDBA privileges.

60.2 Summary of DBMS_DNFS Subprograms

The DBMS_DNFS package includes the CLONEDB_RENAMEFILE, RESTORE_DATAFILE_PERMISSIONS, and UNMOUNTVOLUME procedures.

Table 60-1  DBMS_DNFS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLONEDB_RENAMEFILE Procedure</td>
<td>Renames datafiles that were pointing to the backup set to the actual file name in cloned database.</td>
</tr>
<tr>
<td>RESTORE_DATAFILE_PERMISSIONS Procedure</td>
<td>This procedure will restore the file permissions for the data files belonging to the PDB to read write. PDB data files are converted to read only when one or more clones are created using this source PDB.</td>
</tr>
<tr>
<td>UNMOUNTVOLUME Procedure</td>
<td>This procedure cleans up the cached Direct NFS file handles in SGA.</td>
</tr>
</tbody>
</table>

60.2.1 CLONEDB_RENAMEFILE Procedure

This procedure is used to rename datafiles that were pointing to the backup set to the actual file name in cloned database.

The srcfile is the file name that represents the data file in the backup image copy or a read-only storage snapshot. The destfile destination file path must point to a NFS volume where cloneDB datafiles will be created. When the procedure is run successfully, the control file record is updated with the new datafile name.
Syntax

DBMS_DNFS.CLONEDB_RENAMEFILE (  
    srcfile        IN         VARCHAR2,  
    destfile       IN         VARCHAR2);  

Parameters

Table 60-2  CLONEDB_RENAMEFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcfile</td>
<td>Source datafile name in the control file</td>
</tr>
<tr>
<td>destfile</td>
<td>New datafile name</td>
</tr>
</tbody>
</table>

60.2.2 RESTORE_DATAFILE_PERMISSIONS Procedure

This procedure restores the file permissions for the data files belonging to the PDB to read write. PDB data files are converted to read only when one or more clones are created using this source PDB. This procedure must be invoked after the last clone is dropped to restore the data file permissions for the source PDB files.

Syntax

DBMS_DNFS.RESTORE_DATAFILE_PERMISSIONS (  
    pdb_name       IN         VARCHAR2 DEFAULT NULL);  

Parameters

Table 60-3  RESTORE_DATAFILE_PERMISSIONS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdb_name</td>
<td>Name of pluggable database whose file permissions need to be restored.</td>
</tr>
</tbody>
</table>

60.2.3 UNMOUNTVOLUME Procedure

This procedure cleans up the cached Direct NFS file handles in SGA. This procedure must be used when any NFS mount changes are being made via the operating system and the OS unmount and mount commands are invoked.

The server is the host name or IP address of the NFS server. The volume is the exported path from the NFS server. This procedure will fail if the database has any open files in the volume that is being unmounted.

Syntax

DBMS_DNFS.UNMOUNTVOLUME (  
    server         IN         VARCHAR2,  
    volume         IN         VARCHAR2);
Parameters

Table 60-4  UNMOUNTVOLUME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>NFS server host name or IP address.</td>
</tr>
<tr>
<td>volume</td>
<td>Exported path from the NFS server.</td>
</tr>
</tbody>
</table>
The DBMS_DST package provides an interface to apply the Daylight Saving Time (DST) patch to the Timestamp with Time Zone datatype.

This chapter contains the following topics:

- Overview
- Security Model
- Views
- Summary of DBMS_DST Subprograms

See Also:

- Oracle Database Globalization Support Guide
- Oracle Database Reference

61.1 DBMS_DST Overview

The transition period during which Daylight Saving Time comes into effect, or stops being in effect, has the potential for problems, such as data loss, when handling timestamps with time zone data. The DBMS_DST package enables working with these transitions in the context of a set of rules.

61.2 DBMS_DST Security Model

The DBMS_DST package is an invoker's rights package.

See Also:

Oracle Database PL/SQL Language Reference for more information about using Invoker Rights or Definer Rights

The execute privilege on the package is granted to the EXECUTE_CATALOG_ROLE role. This role is normally granted to selected users to allow EXECUTE privileges for packages and procedures in the data dictionary.

The user that invokes the package must have the following privileges:

- CREATE ANY TABLE
- ALTER ANY TABLE
61.3 DBMS_DST Views

The DBMS_DST package uses views to display table information.

These views are shown in the following table. They are further described in the Oracle Database Reference:

<table>
<thead>
<tr>
<th>View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_TSTZ_TABLES</td>
<td>Displays information about all tables in the database, which have columns defined on TIMESTAMP WITH TIME ZONE datatypes or object types containing attributes of TIMESTAMP WITH TIME ZONE datatypes. Its columns are the same as those in ALL_TSTZ_TABLES.</td>
</tr>
<tr>
<td>USER_TSTZ_TABLES</td>
<td>Displays information about the tables owned by the current user, which have columns defined on TIMESTAMP WITH TIME ZONE datatypes or object types containing attributes of TIMESTAMP WITH TIME ZONE datatypes. Its columns (except for OWNER) are the same as those in ALL_TSTZ_TABLES.</td>
</tr>
<tr>
<td>ALL_TSTZ_TABLES</td>
<td>Displays information about the tables accessible to the current user, which have columns defined on TIMESTAMP WITH TIME ZONE datatypes or object types containing attributes of TIMESTAMP WITH TIME ZONE datatypes.</td>
</tr>
</tbody>
</table>

61.4 Summary of DBMS_DST Subprograms

This table lists and describes the DBMS_DST package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_PREPARE Procedure</td>
<td>Starts a prepare window</td>
</tr>
<tr>
<td>BEGIN_UPGRADE Procedure</td>
<td>Starts an upgrade window</td>
</tr>
<tr>
<td>CREATE_AFFECTED_TABLE Procedure</td>
<td>Creates a table that has the schema shown in the comments for the FIND_AFFECTED_TABLES Procedure</td>
</tr>
</tbody>
</table>
### Table 61-2  (Cont.) DBMS_DST Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_ERROR_TABLE Procedure</td>
<td>Creates a log error table</td>
</tr>
<tr>
<td>CREATE_TRIGGER_TABLE Procedure</td>
<td>Creates a table that is used to record active triggers disabled before performing upgrade on the table, having not been enabled due to fatal failure during the upgrading process</td>
</tr>
<tr>
<td>END_PREPARE Procedure</td>
<td>Ends a prepare window</td>
</tr>
<tr>
<td>END_UPGRADE Procedure</td>
<td>Ends an upgrade window</td>
</tr>
<tr>
<td>FIND_AFFECTED_TABLES Procedure</td>
<td>Finds all the tables that have affected TSTZ data due to the new timezone version</td>
</tr>
<tr>
<td>UPGRADE_DATABASE Procedure</td>
<td>Upgrades all tables in the database that have one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type</td>
</tr>
<tr>
<td>UPGRADE_SCHEMA Procedure</td>
<td>Upgrades tables in a specified list of schemas that has one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type</td>
</tr>
<tr>
<td>UPGRADE_TABLE Procedure</td>
<td>Upgrades a specified list of tables that has one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type</td>
</tr>
</tbody>
</table>

### 61.4.1 BEGIN_PREPARE Procedure

This procedure starts a prepare window. Once a prepare window is started successfully, the database property 'DST_UPGRADE_STATE' is set to 'PREPARE', and the database property 'SECONDARY_TT_VERSION' is set to a new timezone version.

The prepare window lets a DBA investigate data affected by the upgrade, and so judge when it is optimal to perform the upgrade. The prepare window can overlap normal database operation.

**Syntax**

```sql
DBMS_DST.BEGIN_PREPARE (new_version IN BINARY_INTEGER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_version</td>
<td>New timezone version to which the database is to be prepared to upgrade</td>
</tr>
</tbody>
</table>

### 61.4.2 BEGIN_UPGRADE Procedure

This procedure starts an upgrade window.

When an upgraded window is started successfully, the TSTZ data in the dictionary tables is upgraded to reflect the new timezone version, and the database property...
'DST_UPGRADE_STATE' is set to 'UPGRADE'. Once BEGIN_UPGRADE has been performed successfully, the user must re-start the database. After a successful restart, the database property 'PRIMARY_TT_VERSION' is the new timezone version, and 'SECONDARY_TT_VERSION' is the old timezone version.

The procedure operates atomically, and upgrades all or none of the dictionary tables and the database properties. It must be called in the database in OPEN MIGRATE mode.

**Syntax**

```sql
DBMS_DST.BEGIN_UPGRADE (new_version                IN  BINARY_INTEGER,
                          error_on_overlap_time      IN  BOOLEAN := FALSE,
                          error_on_nonexisting_time  IN  BOOLEAN := FALSE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_version</td>
<td>New timezone version to which the database is to be upgraded</td>
</tr>
<tr>
<td>error_on_overlap_time</td>
<td>Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error. The default is FALSE. For more information about boundary cases, see Oracle Database SQL Language Reference.</td>
</tr>
<tr>
<td>error_on_nonexisting_time</td>
<td>Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error. The default value is FALSE.</td>
</tr>
</tbody>
</table>

### 61.4.3 CREATE_AFFECTED_TABLE Procedure

This procedure creates a table that has the schema shown in the comments for the FIND_AFFECTED_TABLES Procedure.

**Syntax**

```sql
DBMS_DST.CREATE_AFFECTED_TABLE (table_name      IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table created</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedures takes a `table_name` without schema qualification, creating a table within the current user schema.
Related Topics

- **FIND_AFFECTED_TABLES Procedure**
  This procedure finds all the tables which have affected TSTZ data due to the new timezone version.

### 61.4.4 CREATE_ERROR_TABLE Procedure

This procedure creates a log error table.

The table has the following schema:

```sql
CREATE TABLE dst$error_table(
    table_owner   VARCHAR2(30),
    table_name    VARCHAR2(30),
    column_name   VARCHAR2(4000),
    rid           ROWID,
    error_number  NUMBER)
```

**Syntax**

```sql
DBMS_DST.CREATE_ERROR_TABLE (  
    table_name  IN VARCHAR2);
```

**Parameters**

**Table 61-6  CREATE_ERROR_TABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table created</td>
</tr>
</tbody>
</table>

**Usage Notes**

- This procedure takes a `table_name` without schema qualification, creating a table within the current user schema.
- The error number is found when upgrading time zone file and timestamp with time zone data. For more information about error handling when upgrading time zone file and timestamp with time zone data, see Oracle Database Globalization Support Guide

### 61.4.5 CREATE_TRIGGER_TABLE Procedure

This procedure creates a table to record active triggers that are disabled before performing upgrade on the table, having not been enabled due to fatal failure during the upgrading process.

The table that has the following schema.

```sql
CREATE TABLE dst_trigger_table (  
    trigger_owner  VARCHAR2(30),
    trigger_name   VARCHAR2(30));
```

**Syntax**

```sql
DBMS_DST.CREATE_TRIGGER_TABLE (  
    table_name  IN VARCHAR2);
```
Parameters

Table 61-7  CREATE_TRIGGER_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of table to be created</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure takes a table_name without schema qualification, creating a table within the current user schema.

61.4.6 END_PREPARE Procedure

This procedure ends a prepare window.

Syntax

DBMS_DST.BEGIN_PREPARE;

61.4.7 END_UPGRADE Procedure

This procedure ends an upgrade window. An upgraded window is ended if all the affected user tables have been upgraded. Otherwise, the OUT parameter num_of_failures indicates how many tables have not been converted.

Syntax

DBMS_DST.END_UPGRADE (  
    num_of_failures    OUT  BINARY_INTEGER);

Parameters

Table 61-8  END_UPGRADE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_of_failures</td>
<td>Number of tables that fail to complete</td>
</tr>
</tbody>
</table>

61.4.8 FIND_AFFECTED_TABLES Procedure

This procedure finds all the tables which have affected TSTZ data due to the new timezone version.

This procedure can only be invoked during a prepare window. The tables which have affected TSTZ data are recorded into a table indicated by parameter affected_tables. If semantic errors must be logged, they are recorded into a table indicated by parameter log_errors_table.

Syntax

DBMS_DST.FIND_AFFECTED_TABLES (  
    affected_tables             IN  VARCHAR2 =: 'sys.dst$affected_tables',  
    num_of_failures             OUT  BINARY_INTEGER,  
    log_errors                 IN  VARCHAR2 =: 'sys.dst$log_errors_table',  
    semanticErrors              OUT  VARCHAR2)
Parameters

Table 61-9 FIND_AFFECTED_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>affected_tables</td>
<td>Name of table with the following schema:</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE dst$affected_tables (</td>
</tr>
<tr>
<td></td>
<td>table_owner VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>table_name VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>column_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>row_count NUMBER,</td>
</tr>
<tr>
<td></td>
<td>error_count NUMBER)</td>
</tr>
<tr>
<td></td>
<td>The table can be created with the CREATE_AFFECTED_TABLE Procedure.</td>
</tr>
<tr>
<td>log_errors</td>
<td>Boolean flag indicating whether to log errors during upgrade. If FALSE,</td>
</tr>
<tr>
<td></td>
<td>no error is logged into the log_errors_table after aborting conversion of the</td>
</tr>
<tr>
<td></td>
<td>current table. If TRUE, the error is logged to the log_errors_table.</td>
</tr>
<tr>
<td></td>
<td>The default is FALSE.</td>
</tr>
<tr>
<td>log_errors_table</td>
<td>Table name with the following schema:</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE dst$error_table (</td>
</tr>
<tr>
<td></td>
<td>table_owner VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>table_name VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>column_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>rid ROWID,</td>
</tr>
<tr>
<td></td>
<td>error_number NUMBER)</td>
</tr>
<tr>
<td></td>
<td>The table can be created with the CREATE_ERROR_TABLE Procedure. The rid</td>
</tr>
<tr>
<td></td>
<td>column records the rowids of the offending rows,</td>
</tr>
<tr>
<td></td>
<td>and the error_number column records the corresponding error number.</td>
</tr>
<tr>
<td>parallel</td>
<td>Boolean flag indicating whether to find the affected tables using parallel</td>
</tr>
<tr>
<td></td>
<td>queries or serial queries. The default is FALSE.</td>
</tr>
</tbody>
</table>

61.4.9 UPGRADE_DATABASE Procedure

This procedure upgrades all tables in the database, which have one or more columns defined on the TSTZ type or an ADT containing the TSTZ type.

This procedure can only be invoked after an upgrade window has been started. Each table is upgraded in an atomic transaction. Note that, a base table and its materialized view log table are upgraded in an atomic transaction.

Syntax

```sql
DBMS_DST.UPGRADE_DATABASE  (  
    num_of_failures             OUT BINARY_INTEGER,  
    upgrade_data                IN  BOOLEAN := TRUE,  
    parallel                    IN  BOOLEAN := FALSE,  
)```
Parameters

Table 61-10  UPGRADE_DATABASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_of_failures</td>
<td>Number of tables that fail to complete</td>
</tr>
<tr>
<td>upgrade_data</td>
<td>Boolean flag indicating whether to convert TSTZ data using the new Time Zone patch File (TRUE), or to leave it unconverted (FALSE). The default is TRUE.</td>
</tr>
<tr>
<td>parallel</td>
<td>Boolean flag indicating whether to convert tables using PDML (Parallel DML) or Serial DML. The default is FALSE.</td>
</tr>
<tr>
<td>continue_after_errors</td>
<td>Boolean flag indicating whether to continue after upgrade fails on the current table. The default is TRUE.</td>
</tr>
<tr>
<td>log_errors</td>
<td>Boolean flag indicating whether to log errors during upgrade. If FALSE, no error is logged into the log_errors_table after aborting conversion of the current table. If TRUE, errors are logged to the log_errors_table. The default is FALSE.</td>
</tr>
<tr>
<td>log_errors_table</td>
<td>Table name with the following schema:</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE dst$error_table (table_owner VARCHAR2(30), table_name VARCHAR2(30), column_name VARCHAR2(4000), rid ROWID, error_number NUMBER)</td>
</tr>
<tr>
<td>error_on_overlap_time</td>
<td>Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error. The default is TRUE.</td>
</tr>
<tr>
<td>error_on_nonexisting_time</td>
<td>Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error. The default is TRUE.</td>
</tr>
<tr>
<td>log_triggers_table</td>
<td>Table to log triggers which are disabled before upgrade, having not been enabled due to a fatal failure when performing an upgrade.</td>
</tr>
</tbody>
</table>
61.4.10 UPGRADE_SCHEMA Procedure

This procedure upgrades tables in a specified list of schemas that have one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type.

This procedure can be invoked only after an upgrade window has been started. Each table is upgraded in an atomic transaction. Note that a base table and its materialized view log table are upgraded in an atomic transaction.

Syntax

```sql
DBMS_DST.UPGRADE_SCHEMA (  
  num_of_failures OUT BINARY_INTEGER,  
  schema_list IN VARCHAR2,  
  upgrade_data IN BOOLEAN := TRUE,  
  parallel IN BOOLEAN := FALSE,  
  continue_after_errors IN BOOLEAN := TRUE,  
  log_errors IN BOOLEAN := FALSE,  
  log_errors_table IN VARCHAR2 := 'sys.dst$error_table',  
  error_on_overlap_time IN BOOLEAN := FALSE,  
  error_on_nonexisting_time IN BOOLEAN := FALSE,  
  log_triggers_table IN VARCHAR2 := 'sys.dst$trigger_table');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_of_failures</td>
<td>Number of tables that fail to complete</td>
</tr>
<tr>
<td>schema_list</td>
<td>Schema name list (comma separated strings)</td>
</tr>
<tr>
<td>upgrade_data</td>
<td>Boolean flag indicating whether to convert TSTZ data using the new Time Zone patch File (TRUE) or to leave unconverted (FALSE). The default is TRUE.</td>
</tr>
<tr>
<td>parallel</td>
<td>Boolean flag indicating whether to convert tables using PDML (Parallel DML) or Serial DML. The default is FALSE.</td>
</tr>
<tr>
<td>continue_after_errors</td>
<td>Boolean flag indicating whether to continue after upgrade fails on the current table. The default is TRUE.</td>
</tr>
<tr>
<td>log_errors</td>
<td>Boolean flag indicating whether to log errors during upgrade. If FALSE, no error is logged into the log_errors_table after aborting conversion of the current table. If TRUE, the error is logged to the log_errors_table. The default is FALSE.</td>
</tr>
</tbody>
</table>
### Table 61-11  (Cont.) UPGRADE_SCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_errors_table</td>
<td>Table name with the following schema:</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE dst$error_table (</td>
</tr>
<tr>
<td></td>
<td>table_owner VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>table_name VARCHAR2(30),</td>
</tr>
<tr>
<td></td>
<td>column_name VARCHAR2(4000),</td>
</tr>
<tr>
<td></td>
<td>rid ROWID,</td>
</tr>
<tr>
<td></td>
<td>error_number NUMBER)</td>
</tr>
<tr>
<td></td>
<td>The table can be created with the CREATE_ERROR_TABLE Procedure. The rid column records the rowids of the offending rows, and the error_number column records the corresponding error number.</td>
</tr>
<tr>
<td>error_on_overlap_time</td>
<td>Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error.</td>
</tr>
<tr>
<td></td>
<td>The default is TRUE.</td>
</tr>
<tr>
<td>error_on_nonexisting_time</td>
<td>Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error.</td>
</tr>
<tr>
<td></td>
<td>The default is TRUE.</td>
</tr>
<tr>
<td>log_triggers_table</td>
<td>Table to log triggers that are disabled before upgrade, having not been enabled due to a fatal failure when performing an upgrade.</td>
</tr>
</tbody>
</table>

### 61.4.11 UPGRADE_TABLE Procedure

This procedure upgrades a specified list of tables that have one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type.

#### Syntax

```sql
DBMS_DST.UPGRADE_TABLE ( |
  num_of_failures OUT BINARY_INTEGER, |
  table_list IN VARCHAR2, |
  upgrade_data IN BOOLEAN := TRUE, |
  parallel IN BOOLEAN := FALSE, |
  continue_after_errors IN BOOLEAN := TRUE, |
  log_errors IN BOOLEAN := FALSE, |
  log_errors_table IN VARCHAR2 := 'sys.dst$error_table', |
  error_on_overlap_time IN BOOLEAN := FALSE, |
  error_on_nonexisting_time IN BOOLEAN := FALSE, |
  log_triggers_table IN VARCHAR2 := 'sys.dst$trigger_table', |
  atomic_upgrade IN BOOLEAN := FALSE); |
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_of_failures</td>
<td>Number of tables that fail to complete</td>
</tr>
<tr>
<td>table_list</td>
<td>Table name list (comma separated strings)</td>
</tr>
</tbody>
</table>
### Table 61-12  (Cont.) UPGRADE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>upgrade_data</td>
<td>Boolean flag indicating whether to convert TSTZ data using the new Time Zone patch File (TRUE), or to leave unconverted (FALSE).&lt;br&gt;The default is TRUE.</td>
</tr>
<tr>
<td>parallel</td>
<td>Boolean flag indicating whether to convert tables using PDML (Parallel DML), or Serial DML.&lt;br&gt;The default is FALSE.</td>
</tr>
<tr>
<td>continue_after_errors</td>
<td>Boolean flag indicating whether to continue after upgrade fails on the current table.&lt;br&gt;The default is TRUE.</td>
</tr>
<tr>
<td>log_errors</td>
<td>Boolean flag indicating whether to log errors during upgrade. If FALSE, no error is logged into the log_errors_table after aborting conversion of the current table. If TRUE, the error is logged to the log_errors_table.&lt;br&gt;The default is FALSE.</td>
</tr>
<tr>
<td>log_errors_table</td>
<td>Table name with the following schema:&lt;br&gt;CREATE TABLE dsterror_table (table_owner VARCHAR2(30), table_name VARCHAR2(30), column_name VARCHAR2(4000), rid ROWID, error_number NUMBER)&lt;br&gt;The table can be created with the CREATE_ERROR_TABLE Procedure. The rid parameter records the rowids of the offending rows and the corresponding error number.</td>
</tr>
<tr>
<td>error_on_overlap_time</td>
<td>Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error.&lt;br&gt;The default is TRUE.</td>
</tr>
<tr>
<td>error_on_nonexisting_time</td>
<td>Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error.&lt;br&gt;The default is TRUE.</td>
</tr>
<tr>
<td>log_triggers_table</td>
<td>Table to log triggers that are disabled before upgrade, having not been enabled due to a fatal failure when performing an upgrade atomic_upgrade</td>
</tr>
</tbody>
</table>

### Usage Notes

This procedure can only be invoked after an upgrade window has been started. The table list has to satisfy the following partial ordering:

1. If a base table has a materialized view log table, the log table must be the next item in the list.
2. If the container table for a materialized view appears in the list, the materialized view's 'non-upgraded' base tables and log tables must appear in the table list and before the container table.

A base table and its materialized view log table need to be upgraded in an atomic transaction by specifying `atomic_upgrade` to `TRUE`.
The DBMS_EDITIONS_UTILITIES package provides helper functions for edition-related operations.

The chapter contains the following topics:

- Overview
- Security Model
- Exceptions
- Summary of DBMS_EDITIONS_UTILITIES Subprograms

62.1 DBMS_EDITIONS_UTILITIES Overview

The DBMS_EDITIONS_UTILITIES package implements an interface which provides helper functions for edition-related operations.

62.2 DBMS_EDITIONS_UTILITIES Security Model

This package is owned by SYS with execute access granted to PUBLIC. It runs with invoker's rights, that is, with the security profile of the caller.

62.3 DBMS_EDITIONS_UTILITIES Exceptions

The table in this topic lists exception messages created by DBMS_EDITIONS_UTILITIES.

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-38817</td>
<td>Insufficient privileges</td>
</tr>
<tr>
<td>ORA-942</td>
<td>Missing table</td>
</tr>
</tbody>
</table>

62.4 Summary of DBMS_EDITIONS_UTILITIES Subprograms

This table lists and describes the DBMS_EDITIONS_UTILITIES package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAN_UNUSABLE_EDITIONS</td>
<td>Drops covered objects in unusable editions, and drops empty unusable editions if possible.</td>
</tr>
</tbody>
</table>
Table 62-2  (Cont.) DBMS_EDITIONS_UTILITIES Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_EDITIONING_VIEWS_READ_ONLY Procedure</td>
<td>Given the schema name and table name, this procedure sets the corresponding editioning views in all editions to READ ONLY or READ/WRITE</td>
</tr>
<tr>
<td>SET_NULL_COLUMN_VALUES_TO_EXPR Procedure</td>
<td>For use only during an edition-based redefinition (EBR) exercise</td>
</tr>
</tbody>
</table>

62.4.1 CLEAN_UNUSABLE_EDITIONS Procedure

Drops all covered objects in any unusable editions and drops empty unusable editions.

The CLEAN_UNUSABLE_EDITIONS procedure executes immediately and in its entirety.

Syntax

```sql
DBMS_EDITIONS_UTILITIES.CLEAN_UNUSABLE_EDITIONS ();
```

Usage Notes

The COMPATIBLE parameter must be set to 12.2.0 or higher for this procedure to execute.

See Also:

- Oracle Database SQL Language Reference for information about DROP EDITION
- Oracle Database Development Guide for more information about using edition-based redefinition

62.4.2 SET_EDITIONING_VIEWS_READ_ONLY Procedure

Given the schema name and table name, this procedure sets the corresponding editioning views in all editions to READ ONLY or READ/WRITE.

Syntax

```sql
DBMS_EDITIONS_UTILITIES.SET_EDITIONING_VIEWS_READ_ONLY ( 
  table_name IN VARCHAR2, 
  owner IN VARCHAR2 DEFAULT NULL, 
  read_only IN BOOLEAN DEFAULT TRUE);
```
62.3 SET_EDITIONING_VIEWS_READ_ONLY Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Base table of the editioning views</td>
</tr>
<tr>
<td>owner</td>
<td>Base table schema. The default (or NULL) is the current schema.</td>
</tr>
<tr>
<td>read_only</td>
<td>TRUE to set the views to read-only; FALSE (or NULL) sets the views to READ/WRITE. Default is TRUE.</td>
</tr>
</tbody>
</table>

Usage Notes

The user must have the following privileges:

- Owner of the table, or have the ALTER ANY TABLE system privileges
- USE object privilege on all the editions for which the views are defined

62.4.3 SET_NULL_COLUMN_VALUES_TO_EXPR Procedure

This procedure replaces NULL values in a replacement column with the value of an expression.

The expression evaluation cost is deferred to future updates and queries. The procedure is intended for use only during an edition-based redefinition (EBR) exercise.

See Also:

- Oracle Database Development Guide regarding transforming pre- to post-upgrade representation

Syntax

```sql
DBMS_EDITIONS_UTILITIES.SET_NULL_COLUMN_VALUES_TO_EXPR;
  table_name  IN  VARCHAR2,
  column_name IN  VARCHAR2,
  expression  IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>A potentially schema-qualified table name</td>
</tr>
<tr>
<td>column_name</td>
<td>Name of the column to be updated</td>
</tr>
<tr>
<td>expression</td>
<td>An expression composed of columns in the same table, constants, and SQL functions</td>
</tr>
</tbody>
</table>
The **DBMS_EPG** package implements the embedded PL/SQL gateway that enables a Web browser to invoke a PL/SQL stored procedure through an HTTP listener.

This chapter contains the following topics:

- Overview
- Security Model
- Exceptions
- Data Structures
  - VARCHAR2_TABLE Table Type
- Subprogram Groups
- Summary of DBMS_EPG Subprograms

### 63.1 DBMS_EPG Overview

The **DBMS_EPG** package is a platform on which PL/SQL users develop and deploy PL/SQL Web applications. The embedded PL/SQL gateway is an embedded version of the gateway that runs in the XML database HTTP server in the Oracle database. It provides the core features of **mod_plsql** in the database but does not require the Oracle HTTP server powered by Apache.

In order to make a PL/SQL application accessible from a browser by way of HTTP, a Database Access Descriptor (DAD) must be created and mapped to a virtual path. A DAD is a set of configuration values used for database access and the virtual path mapping makes the application accessible under a virtual path of the XML DB HTTP Server. A DAD is represented as a servlet in XML DB HTTP Server.

### 63.2 DBMS_EPG Security Model

The **XDBADMIN** role is required to invoke the configuration interface. It may invoked by the database user "XDB".

The authorization interface can be invoked by any user.

### 63.3 DBMS_EPG Exceptions

The table in this topic lists the exceptions raised by the **DBMS_EPG** package.
### Table 63-1  DBMS_EPG Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAD_NOT_FOUND</td>
<td>20000</td>
<td>Database Access Descriptor (DAD) %s not found. Ensure that the name of the DAD is correct and that it exists.</td>
</tr>
</tbody>
</table>

### 63.4 DBMS_EPG Data Structures

The **DBMS_EPG** package defines a **TABLE** type.

**VARCHAR2_TABLE** Table Type

This type is used by the procedures **GET_ALL_GLOBAL_ATTRIBUTES**, **GET_ALL_DAD_ATTRIBUTES**, **GET_ALL_DAD_MAPPINGS**, and **GET_DAD_LIST** to return lists of attribute names, attribute values, virtual paths, and database access descriptors (DAD).

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

### 63.5 DBMS_EPG Subprogram Groups

The DBMS_EPG consists of two interfaces: configuration subprograms and authorization subprograms.

- **Configuration Subprograms**
- **Authorization Subprograms**

#### 63.5.1 DBMS_EPG Configuration Subprograms

The Configuration subprogram group contain the subprogram interfaces to examine and modify the global and database access descriptor (DAD) specific settings of the embedded PL/SQL gateway.

### Table 63-2  Configuration Subprogram Group

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_DAD Procedure</td>
<td>Creates a new DAD</td>
</tr>
<tr>
<td>DELETE_DAD_ATTRIBUTE Procedure</td>
<td>Deletes a DAD attribute</td>
</tr>
<tr>
<td>DELETE_GLOBAL_ATTRIBUTE Procedure</td>
<td>Deletes a global attribute</td>
</tr>
<tr>
<td>DROP_DAD Procedure</td>
<td>Drops a DAD</td>
</tr>
<tr>
<td>GET_ALL_DAD_ATTRIBUTES Procedure</td>
<td>Retrieves all the attributes of a DAD.</td>
</tr>
<tr>
<td>GET_ALL_DAD_MAPPINGS Procedure</td>
<td>Retrieves all virtual paths to which the specified DAD is mapped.</td>
</tr>
<tr>
<td>GET_ALL_GLOBAL_ATTRIBUTES Procedure</td>
<td>Retrieves all global attributes and values</td>
</tr>
<tr>
<td>GET_DAD_ATTRIBUTE Function</td>
<td>Retrieves the value of a DAD attribute</td>
</tr>
</tbody>
</table>
### Table 63-2 (Cont.) Configuration Subprogram Group

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_DAD_LIST Procedure</td>
<td>Retrieves a list of all DADs for an Embedded Gateway instance.</td>
</tr>
<tr>
<td>GET_GLOBAL_ATTRIBUTE Function</td>
<td>Retrieves the value of a global attribute</td>
</tr>
<tr>
<td>MAP_DAD Procedure</td>
<td>Maps a DAD to the specified virtual path.</td>
</tr>
<tr>
<td>SET_DAD_ATTRIBUTE Procedure</td>
<td>Sets the value for a DAD</td>
</tr>
<tr>
<td>SET_GLOBAL_ATTRIBUTE Procedure</td>
<td>Sets the value of a global attribute</td>
</tr>
<tr>
<td>UNMAP_DAD Procedure</td>
<td>Unmaps a DAD from the specified virtual path</td>
</tr>
</tbody>
</table>

### 63.5.2 DBMS_EPG Authorization Subprograms

The Authorization subprogram group contains the subprogram interfaces to authorize and deauthorize the use of a database user's privileges by the embedded PL/SQL gateway through a specific database access descriptor (DAD)

### Table 63-3 Authorization Subprogram Group

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHORIZE_DAD Procedure</td>
<td>Authorizes a DAD to invoke procedures and access document tables with a database user's privileges</td>
</tr>
<tr>
<td>DEAUTHORIZE_DAD Procedure</td>
<td>Deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges</td>
</tr>
</tbody>
</table>

### 63.6 Summary of DBMS_EPG Subprograms

This table lists the DBMS_ALERT subprograms and briefly describes them.

### Table 63-4 DBMS_EPG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHORIZE_DAD Procedure</td>
<td>authorizes a DAD to invoke procedures and access document tables with a database user's privileges</td>
</tr>
<tr>
<td>CREATE_DAD Procedure</td>
<td>Creates a new DAD</td>
</tr>
<tr>
<td>DEAUTHORIZE_DAD Procedure</td>
<td>Deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges</td>
</tr>
<tr>
<td>DELETE_DAD_ATTRIBUTE Procedure</td>
<td>Deletes a DAD attribute</td>
</tr>
<tr>
<td>DELETE_GLOBAL_ATTRIBUTE Procedure</td>
<td>Deletes a global attribute</td>
</tr>
<tr>
<td>DROP_DAD Procedure</td>
<td>Drops a DAD</td>
</tr>
<tr>
<td>GET_ALL_DAD_ATTRIBUTES Procedure</td>
<td>Retrieves all the attributes of a DAD.</td>
</tr>
<tr>
<td>GET_ALL_DAD_MAPPINGS Procedure</td>
<td>Retrieves all virtual paths to which the specified DAD is mapped.</td>
</tr>
</tbody>
</table>
Table 63-4  (Cont.) DBMS_EPG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_ALL_GLOBAL_ATTRIBUTES Procedure</td>
<td>Retrieves all global attributes and values</td>
</tr>
<tr>
<td>GET_DAD_ATTRIBUTE Function</td>
<td>Retrieves the value of a DAD attribute</td>
</tr>
<tr>
<td>GET_DAD_ATTRIBUTE Procedure</td>
<td>Retrieves a list of all DADs for an Embedded Gateway instance.</td>
</tr>
<tr>
<td>GET_DAD_ATTRIBUTE Function</td>
<td>Retrieves the value of a global attribute</td>
</tr>
<tr>
<td>MAP_DAD Procedure</td>
<td>Maps a DAD to the specified virtual path</td>
</tr>
<tr>
<td>SET_DAD_ATTRIBUTE Procedure</td>
<td>Sets the value for a DAD</td>
</tr>
<tr>
<td>SET_GLOBAL_ATTRIBUTE Procedure</td>
<td>Sets the value of a global attribute</td>
</tr>
<tr>
<td>UNMAP_DAD Procedure</td>
<td>Unmaps a DAD from the specified virtual path</td>
</tr>
</tbody>
</table>

63.6.1 AUTHORIZE_DAD Procedure

This procedure authorizes a DAD to invoke procedures and access document tables with a database user’s privileges. The invoker can always authorize the use of her/his own privileges.

![See Also:](Authorization Subprograms for other subprograms in this group)

Syntax

```sql
DBMS_EPG.AUTHORIZE_DAD (  
    dad_name  IN  VARCHAR2,  
    path     IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 63-5  AUTHORIZE_DAD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD to create</td>
</tr>
<tr>
<td>user</td>
<td>The user whose privileges to deauthorize. If use, the invoker is assumed.</td>
</tr>
</tbody>
</table>

Usage Notes

- To authorize the use of another user’s privileges, the invoker must have the `ALTER USER` system privilege.
- The DAD must exist but its "database-username" DAD attribute does not have to be set to user to authorize.
- Multiple users can authorize the same DAD and it is up to the DAD’s "database-username" setting to decide which user’s privileges to use.
Exception

Raises an error if the DAD or user does not exist, or the invoker does not have the needed system privilege.

Examples

DBMS_EPG.AUTHORIZE_DAD('HR');

63.6.2 CREATE_DAD Procedure

This procedure creates a new DAD.

| See Also: |
| Configuration Subprograms for other subprograms in this group |

Syntax

DBMS_EPG.CREATE_DAD (
    dad_name  IN  VARCHAR2,
    path      IN  VARCHAR2 DEFAULT NULL);

Parameters

Table 63-6  CREATE_DAD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD to create</td>
</tr>
<tr>
<td>path</td>
<td>The virtual path to which to map the DAD</td>
</tr>
</tbody>
</table>

63.6.3 DEAUTHORIZE_DAD Procedure

This procedure deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges. The invoker can always deauthorize the use of his own privileges.

| See Also: |
| Authorization Subprograms for other subprograms in this group |

Syntax

DBMS_EPG.DEAUTHORIZE_DAD (
    dad_name  IN  VARCHAR2,
    path      IN  VARCHAR2 DEFAULT NULL);
Parameters

Table 63-7  DEAUTHORIZE_DAD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD for which to deauthorize use</td>
</tr>
<tr>
<td>user</td>
<td>The user whose privileges to deauthorize. If use, the invoker is assumed.</td>
</tr>
</tbody>
</table>

Usage Notes

To deauthorize the use of another user's privileges, the invoker must have the ALTER USER system privilege.

Exceptions

Raises an error if the DAD or user does not exist, or the invoker does not have the needed system privilege.

Examples

DBMS_EPG.DEAUTHORIZE_DAD('HR');

63.6.4 DELETE_DAD_ATTRIBUTE Procedure

This procedure deletes a DAD attribute.

See Also:

Configuration Subprograms for other subprograms in this group

Syntax

DBMS_EPG.DELETE_DAD_ATTRIBUTE (  
    dad_name      IN  VARCHAR2,  
    attr_name     IN  VARCHAR2);  

Parameters

Table 63-8  DELETE_DAD_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD for which to delete a DAD attribute</td>
</tr>
<tr>
<td>attr_name</td>
<td>The name of the DAD attribute to delete</td>
</tr>
</tbody>
</table>

Exceptions

Raises an error if DAD does not exist
### 63.6.5 DELETE_GLOBAL_ATTRIBUTE Procedure

This procedure deletes a global attribute.

#### Syntax

```sql
DBMS_EPG.DELETE_GLOBAL_ATTRIBUTE (
    attr_name  IN  VARCHAR2);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_name</td>
<td>The global attribute to delete</td>
</tr>
</tbody>
</table>

### 63.6.6 DROP_DAD Procedure

This procedure drops a DAD. All the virtual-path mappings of the DAD will be dropped also.

#### Syntax

```sql
DBMS_EPG.DROP_DAD (
    dadname  IN  VARCHAR2);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The DAD to drop</td>
</tr>
</tbody>
</table>

#### Exceptions

Raises an error if the DAD does not exist.
63.6.7 GET_ALL_DAD_ATTRIBUTES Procedure

This procedure retrieves all the attributes of a DAD. The outputs are 2 correlated index-by tables of the name/value pairs.

```sql
DBMS_EPG.GET_ALL_DAD_ATTRIBUTES (
    dad_name      IN          VARCHAR2,
    attr_names    OUT NOCOPY  VARCHAR2_TABLE,
    attr_values   OUT NOCOPY  VARCHAR2_TABLE);
```

**Parameters**

**Table 63-11   GET_ALL_DAD_ATTRIBUTES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD</td>
</tr>
<tr>
<td>attr_names</td>
<td>The attribute names</td>
</tr>
<tr>
<td>attr_values</td>
<td>The attribute values</td>
</tr>
</tbody>
</table>

**Exceptions**

Raises an error if DAD does not exist.

**Usage Notes**

If the DAD has no attributes set, then `attr_names` and `attr_values` will be set to empty arrays.

63.6.8 GET_ALL_DAD_MAPPINGS Procedure

This procedure retrieves all the virtual paths to which the specified DAD is mapped.

```sql
DBMS_EPG.GET_ALL_DAD_MAPPINGS (
    dad_name      IN          VARCHAR2,
    paths         OUT NOCOPY  VARCHAR2_TABLE);
```
Parameters

Table 63-12 GET_ALL_DAD_MAPPINGS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_names</td>
<td>The name of the DAD</td>
</tr>
<tr>
<td>paths</td>
<td>The virtual paths to which the DAD is mapped</td>
</tr>
</tbody>
</table>

Exceptions

Raises an error if DAD does not exist.

Usage Notes

If the DAD is not mapped to any virtual path, paths will be set to empty arrays.

63.6.9 GET_ALL_GLOBAL_ATTRIBUTES Procedure

This procedure retrieves all global attributes and values. The outputs are 2 correlated index-by-tables of the name/value pairs.

See Also:

Configuration Subprograms for other subprograms in this group

Syntax

```sql
DBMS_EPG.GET_ALL_GLOBAL_ATTRIBUTES (     attr_names   OUT NOCOPY VARCHAR2_TABLE,     attr_values   OUT NOCOPY VARCHAR2_TABLE);
```

Parameters

Table 63-13 GET_ALL_GLOBAL_ATTRIBUTES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_names</td>
<td>The global attribute names</td>
</tr>
<tr>
<td>attr_values</td>
<td>The values of the global attributes</td>
</tr>
</tbody>
</table>

Usage Notes

If the gateway instance has no global attributes set, then attr_names and attr_values will be set to empty arrays.
63.6.10 GET_DAD_ATTRIBUTE Function

This procedure retrieves the value of a DAD attribute.

See Also:

Configuration Subprograms for other subprograms in this group

Syntax

DBMS_EPG.GET_DAD_ATTRIBUTE (
   dad_name     IN  VARCHAR2,
   attr_name    IN  VARCHAR2)
RETURN VARCHAR2;

Parameters

Table 63-14   GET_DAD_ATTRIBUTE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD for which to delete an attribute</td>
</tr>
<tr>
<td>attr_name</td>
<td>The name of the attribute to delete</td>
</tr>
</tbody>
</table>

Return values

Returns the DAD attribute value. Returns NULL if attribute is unknown or has not been set.

Exceptions

Raises an error if DAD does not exist.

63.6.11 GET_DAD_LIST Procedure

This procedure retrieves a list of all DADs for an Embedded Gateway instance.

See Also:

Configuration Subprograms for other subprograms in this group

Syntax

DBMS_EPG.GET_DAD_LIST (
   dad_names     OUT NOCOPY  VARCHAR2_TABLE);
Parameters

Table 63-15  GET_DAD_LIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_names</td>
<td>The list of all DADs</td>
</tr>
</tbody>
</table>

Usage Notes

If no DADs exist then \texttt{dad_names} will be set to an empty array.

63.6.12 GET_GLOBAL_ATTRIBUTE Function

This function retrieves the value of a global attribute.

\begin{quote}
\textbf{See Also:}

\textit{Configuration Subprograms} for other subprograms in this group
\end{quote}

Syntax

\begin{verbatim}
DBMS_EPG.GET_GLOBAL_ATTRIBUTE ( \\
    attr_name  IN  VARCHAR2) \\
RETURN VARCHAR2;
\end{verbatim}

Parameters

Table 63-16  GET_GLOBAL_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_name</td>
<td>The global attribute to retrieve</td>
</tr>
</tbody>
</table>

Return Values

Returns the global attribute value. Returns \texttt{NULL} if attribute has not been set or is not a valid attribute.

63.6.13 MAP_DAD Procedure

This procedure maps a DAD to the specified virtual path. If the virtual path exists already, the old virtual-path mapping will be overridden.

\begin{quote}
\textbf{See Also:}

\textit{Configuration Subprograms} for other subprograms in this group
\end{quote}
Syntax

DBMS_EPG.MAP_DAD (  
    dad_name  IN  VARCHAR2,  
    path      IN  VARCHAR2);  

Parameters

Table 63-17  MAP_DAD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD to map</td>
</tr>
<tr>
<td>path</td>
<td>The virtual path to map</td>
</tr>
</tbody>
</table>

Exceptions

 Raises and error if the DAD does not exist.

63.6.14 SET_DAD_ATTRIBUTE Procedure

This procedure sets the value for a DAD.

See Also:

Configuration Subprograms for other subprograms in this group

Syntax

DBMS_EPG.SET_DAD_ATTRIBUTE (  
    dad_name    IN  VARCHAR2,  
    attr_name   IN  VARCHAR2,  
    attr_value  IN  VARCHAR2);  

Parameters

Table 63-18  SET_DAD_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD for which to set the attribute</td>
</tr>
<tr>
<td>attr_name</td>
<td>The name of the attribute to set</td>
</tr>
<tr>
<td>attr_value</td>
<td>The attribute value to set</td>
</tr>
<tr>
<td>mod_plsql DAD Attribute</td>
<td>Embedded PL/SQL Gateway DAD Attribute</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>PlsqlAfterProcedure</td>
<td>after-procedure</td>
</tr>
<tr>
<td>PlsqlAlwaysDescribeProcedure</td>
<td>always-describe-procedure</td>
</tr>
<tr>
<td>PlsqlAuthenticationMode</td>
<td>authentication-mode</td>
</tr>
<tr>
<td>PlsqlBeforeProcedure</td>
<td>before-procedure</td>
</tr>
<tr>
<td>PlsqlBindBucketLengths</td>
<td>bind-bucket-lengths</td>
</tr>
<tr>
<td>PlsqlBindBucketWidths</td>
<td>bind-bucket-widths</td>
</tr>
<tr>
<td>PlsqlCGIEnvironmentList</td>
<td>cgi-environment-list</td>
</tr>
<tr>
<td>PlsqlCompatibilityMode</td>
<td>compatibility-mode</td>
</tr>
<tr>
<td>PlsqlDatabaseUsername</td>
<td>database-username</td>
</tr>
<tr>
<td>PlsqlDefaultPage</td>
<td>default-page</td>
</tr>
<tr>
<td>PlsqlDocumentPath</td>
<td>document-path</td>
</tr>
<tr>
<td>PlsqlDocumentProcedure</td>
<td>document-procedure</td>
</tr>
<tr>
<td>PlsqlDocumentTablename</td>
<td>document-table-name</td>
</tr>
<tr>
<td>PlsqlErrorStyle</td>
<td>error-style</td>
</tr>
<tr>
<td>PlsqlExclusionList</td>
<td>exclusion-list</td>
</tr>
<tr>
<td>PlsqlFetchBufferSize</td>
<td>fetch-buffer-size</td>
</tr>
<tr>
<td>PlsqlInfoLogging</td>
<td>info-logging</td>
</tr>
<tr>
<td>PlsqlOWADebugEnable</td>
<td>owa-debug-enable</td>
</tr>
<tr>
<td>PlsqlMaxRequestsPerSession</td>
<td>max-requests-per-session</td>
</tr>
<tr>
<td>PlsqlNLSLanguage</td>
<td>nls-language</td>
</tr>
<tr>
<td>PlsqlPathAlias</td>
<td>path-alias</td>
</tr>
<tr>
<td>PlsqlPathAliasProcedure</td>
<td>path-alias-procedure</td>
</tr>
<tr>
<td>PlsqlRequestValidationFunction</td>
<td>request-validation-function</td>
</tr>
<tr>
<td>PlsqlSessionCookieName</td>
<td>session-cookie-name</td>
</tr>
</tbody>
</table>
Table 63-19  (Cont.) Mapping Between mod_plsql and Embedded PL/SQL Gateway DAD Attributes

<table>
<thead>
<tr>
<th>mod_plsql DAD Attribute</th>
<th>Embedded PL/SQL Gateway DAD Attribute</th>
<th>Allows Multiple Occurrences</th>
<th>Legal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlsqlSessionStateManagement</td>
<td>session-state-management</td>
<td>No</td>
<td>Enumeration of StatelessWithResetPackageState, StatelessWithFastResetPackageState, StatelessWithPreservePackageState</td>
</tr>
<tr>
<td>PlsqlTransferMode</td>
<td>transfer-mode</td>
<td>No</td>
<td>Enumeration of Char, Raw</td>
</tr>
<tr>
<td>PlsqlUploadAsLongRaw</td>
<td>upload-as-long-raw</td>
<td>No</td>
<td>String</td>
</tr>
</tbody>
</table>

Exceptions

Raises an error if DAD does not exist or the attribute is unknown.

Usage Notes

- If attr_name attribute has been set before, then the old value will be overwritten with the new attr_value argument.
- The embedded gateway assumes default values when the attributes are not set. The default values of the DAD attributes should be sufficient for most users of the embedded gateway. mod_plsql users should note the following
  - The PlsqlDatabasePassword attribute is not needed.
  - The PlsqlDatabaseConnectString attribute is not needed because the embedded gateway does not support logon to external databases.

Examples

```
DBMS_EPG.SET_DAD_ATTRIBUTE('HR', 'default-page', 'HRApp.home');
```

63.6.15 SET_GLOBAL_ATTRIBUTE Procedure

This procedure sets the value of a global attribute.

See Also:

Configuration Subprograms for other subprograms in this group
Syntax

```sql
DBMS_EPG.SET_GLOBAL_ATTRIBUTE(
    attr_name    IN VARCHAR2,
    attr_value   IN VARCHAR2);
```

Parameters

Table 63-20  SET_GLOBAL_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_name</td>
<td>The global attribute to set</td>
</tr>
<tr>
<td>attr_value</td>
<td>The attribute value to set</td>
</tr>
</tbody>
</table>

Table 63-21  Mapping Between mod_plsql and Embedded PL/SQL Gateway Global Attributes

<table>
<thead>
<tr>
<th>mod_plsql Global Attribute</th>
<th>Embedded PL/SQL Gateway Global Attribute</th>
<th>Allows Multiple Occurrences</th>
<th>Legal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlsqlLogLevel</td>
<td>log-level</td>
<td>No</td>
<td>Unsigned integer</td>
</tr>
<tr>
<td>PlsqlMaxParameters</td>
<td>max-parameters</td>
<td>No</td>
<td>Unsigned integer</td>
</tr>
</tbody>
</table>

Usage Notes

- The attribute name is case sensitive. The value may or may not be case-sensitive depending on the attribute.
- If `attr_name` attribute has been set before, then the old value will be overwritten with the new `attr_value` argument.

Exceptions

Raises an error if the attribute is unknown.

Examples

```sql
dbms_epg.set_global_attribute('max-parameters', '100');
```

63.6.16 UNMAP_DAD Procedure

This procedure unmaps a DAD from the specified virtual path. If path is `NULL`, the procedure removes all virtual-path mappings for the DAD but keeps the DAD.

See Also:

- Configuration Subprograms for other subprograms in this group
Syntax

DBMS_EPG.UNMAP_DAD (  
    dad_name IN VARCHAR2,  
    path IN VARCHAR2 DEFAULT NULL);

Parameters

Table 63-22  UNMAP_DAD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dad_name</td>
<td>The name of the DAD to unmap</td>
</tr>
<tr>
<td>path</td>
<td>The virtual path to unmap</td>
</tr>
</tbody>
</table>

Usage Notes

Raises an error if the DAD does not exist.
64

DBMS_ERRLOG

The DBMS_ERRLOG package provides a procedure that enables you to create an error logging table so that DML operations can continue after encountering errors rather than abort and rollback. This enables you to save time and system resources.

This chapter contains the following topics:

• Security Model
• Summary of DBMS_ERRLOG Subprograms

See Also:
Oracle Database Data Warehousing Guide for more information regarding how to use DBMS_ERRLOG and Oracle Database SQL Language Reference for error_logging_clause syntax

64.1 DBMS_DIMENSION Security Model

Security on this package can be controlled by granting EXECUTE on this package to selected users or roles. The EXECUTE privilege is granted publicly. However, to create an error logging table, you need SELECT access on the base table or view, the CREATE TABLE privilege, as well as tablespace quota for the target tablespace.

64.2 Summary of DBMS_ERRLOG Subprograms

The DBMS_ERRORLOG package includes the CREATE_ERROR_LOG procedure subprogram.

Table 64-1 DBMS_ERRLOG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_ERROR_LOG Procedure</td>
<td>Creates the error logging table used in DML error logging</td>
</tr>
</tbody>
</table>

64.2.1 CREATE_ERROR_LOG Procedure

This procedure creates the error logging table needed to use the DML error logging capability.

LONG, CLOB, BLOB, BFILE, and ADT datatypes are not supported in the columns.
Syntax

DBMS_ERRLOG.CREATE_ERROR_LOG (
    dml_table_name            IN VARCHAR2,
    err_log_table_name        IN VARCHAR2 := NULL,
    err_log_table_owner       IN VARCHAR2 := NULL,
    err_log_table_space       IN VARCHAR2 := NULL,
    skip_unsupported          IN BOOLEAN := FALSE);

Parameters

Table 64-2 CREATE_ERROR_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dml_table_name</td>
<td>The name of the DML table to base the error logging table on. The name can be fully qualified (for example, emp, scott.emp, &quot;EMP&quot;, &quot;SCOTT&quot;.&quot;EMP&quot;). If a name component is enclosed in double quotes, it will not be upper cased.</td>
</tr>
<tr>
<td>err_log_table_name</td>
<td>The name of the error logging table you will create. The default is the first 25 characters in the name of the DML table prefixed with 'ERR$_'. Examples are the following: dml_table_name: 'EMP', err_log_table_name: 'ERR$_EMP' dml_table_name: '&quot;Emp2&quot;', err_log_table_name: 'ERR$_Emp2'</td>
</tr>
<tr>
<td>err_log_table_owner</td>
<td>The name of the owner of the error logging table. You can specify the owner in dml_table_name. Otherwise, the schema of the current connected user is used.</td>
</tr>
<tr>
<td>err_log_table_space</td>
<td>The tablespace the error logging table will be created in. If not specified, the default tablespace for the user owning the DML error logging table will be used.</td>
</tr>
<tr>
<td>skip_unsupported</td>
<td>When set to TRUE, column types that are not supported by error logging will be skipped over and not added to the error logging table. When set to FALSE, an unsupported column type will cause the procedure to terminate. The default is FALSE.</td>
</tr>
</tbody>
</table>

Examples

First, create an error log table for the channels table in the SH schema, using the default name generation.

Then, see all columns of the table channels:

SQL> DESC channels
Name                         Null?       Type
---------------------------            -------     -----
CHANNEL_ID                        NOT NULL    CHAR(1)
CHANNEL_DESC                     NOT NULL    VARCHAR2 (20)
CHANNEL_CLASS                    NOT NULL    VARCHAR2 (20)

Finally, see all columns of the generated error log table. Note the mandatory control columns that are created by the package:
### ERR$_CHANNELS Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA_ERR_NUMBER$</td>
<td>-----</td>
<td>NUMBER</td>
</tr>
<tr>
<td>ORA_ERR_MESG$</td>
<td></td>
<td>VARCHAR2(2000)</td>
</tr>
<tr>
<td>ORA_ERR_ROWID$</td>
<td></td>
<td>ROWID</td>
</tr>
<tr>
<td>ORA_ERR_OPTYP$</td>
<td></td>
<td>VARCHAR2(2)</td>
</tr>
<tr>
<td>ORA_ERR_TAG$</td>
<td></td>
<td>VARCHAR2(2000)</td>
</tr>
<tr>
<td>CHANNEL_ID</td>
<td></td>
<td>VARCHAR2(4000)</td>
</tr>
<tr>
<td>CHANNEL_DESC</td>
<td></td>
<td>VARCHAR2(4000)</td>
</tr>
<tr>
<td>CHANNEL_CLASS</td>
<td></td>
<td>VARCHAR2(4000)</td>
</tr>
</tbody>
</table>

See *Oracle Database Administrator's Guide* for more information regarding control columns.
The DBMS_FGA package provides fine-grained security functions.

This chapter contains the following topics:

- Security Model
- Operational Notes
- Summary of DBMS_FGA Subprograms

### 65.1 DBMS_FGA Security Model

You must have the **AUDIT_ADMIN** role or the **EXECUTE** privilege on the **DBMS_FGA** package to create audit policies. **DBMS_FGA** is an invoker rights package.

To analyze and audit data, you must have the **AUDIT_VIEWER** role. Because the audit function can potentially capture all user environment and application context values, policy administration should be executable by privileged users only. The policy event handler module is executed with the module owner’s privilege.

### 65.2 DBMS_FGA Operational Notes

This package is available for only cost-based optimization. The rule-based optimizer may generate unnecessary audit records since audit monitoring can occur before row filtering.

For both the rule-based optimizer and the cost-based optimizer, you can query the **SQL_TEXT** and **SQL_BINDS** columns of the **UNIFIED_AUDIT_TRAIL** view to analyze the SQL text and corresponding bind variables that are issued.

### 65.3 Summary of DBMS_FGA Subprograms

This table lists the **DBMS_FGA** subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_POLICY Procedure</td>
<td>Creates an audit policy using the supplied predicate as the audit condition</td>
</tr>
<tr>
<td>DISABLE_POLICY Procedure</td>
<td>Disables an audit policy</td>
</tr>
<tr>
<td>DROP_POLICY Procedure</td>
<td>Drops an audit policy</td>
</tr>
<tr>
<td>ENABLE_POLICY Procedure</td>
<td>Enables an audit policy</td>
</tr>
</tbody>
</table>
65.3.1 ADD_POLICY Procedure

This procedure creates an audit policy using the supplied predicate as the audit condition.

Syntax

```sql
DBMS_FGA.ADD_POLICY(
    object_schema      IN  VARCHAR2 DEFAULT NULL,
    object_name        IN  VARCHAR2,
    policy_name        IN  VARCHAR2,
    audit_condition    IN  VARCHAR2 DEFAULT NULL,
    audit_column       IN  VARCHAR2 DEFAULT NULL,
    handler_schema     IN  VARCHAR2 DEFAULT NULL,
    handler_module     IN  VARCHAR2 DEFAULT NULL,
    enable             IN  BOOLEAN DEFAULT TRUE,
    statement_types    IN  VARCHAR2 DEFAULT SELECT,
    audit_trail        IN  BINARY_INTEGER DEFAULT NULL,
    audit_column_opts  IN  BINARY_INTEGER DEFAULT ANY_COLUMNS,
    policy_owner       IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 65-2 ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema of the object to be audited. If NULL, the current schema is assumed.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>Unique name of the policy. Do not enter special characters such as spaces or commas. If you want to use special characters for the policy name, then enclose the name in quotation marks.</td>
</tr>
<tr>
<td>audit_condition</td>
<td>A condition in a row that indicates a monitoring condition. NULL is allowed and acts as TRUE.</td>
</tr>
<tr>
<td>audit_column</td>
<td>Columns to be checked for access. These can include OLS hidden columns or object type columns. The default, NULL, causes audit if any column is accessed or affected.</td>
</tr>
<tr>
<td>handler_schema</td>
<td>Schema that contains the event handler. The default, NULL, causes the current schema to be used.</td>
</tr>
<tr>
<td>handler_module</td>
<td>Function name of the event handler; includes the package name if necessary. This function is invoked only after the first row that matches the audit condition in the query is processed. If the procedure fails with an exception, the user SQL statement will fail as well.</td>
</tr>
<tr>
<td>enable</td>
<td>Enables the policy if TRUE, which is the default</td>
</tr>
<tr>
<td>statement_types</td>
<td>SQL statement types to which this policy is applicable: INSERT, UPDATE, DELETE, or SELECT only</td>
</tr>
<tr>
<td>audit_trail</td>
<td>In an environment that has not yet migrated to unified auditing, the destination (DB or XML) of fine-grained audit records. Also specifies whether to populate the LSQLTEXT and LSQLBIND columns in the FGA_LOG$ system table.</td>
</tr>
</tbody>
</table>
Table 65-2 (Cont.) ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_column_opts</td>
<td>Establishes whether a statement is audited when the query references any column specified in the audit_column parameter or only when all such columns are referenced.</td>
</tr>
<tr>
<td>policy_owner</td>
<td>User who owns the fine-grained auditing policy. However, this setting is not a user-supplied argument. The Oracle Data Pump client uses this setting internally to recreate the fine-grained audit policies appropriately.</td>
</tr>
</tbody>
</table>

Usage Notes

- A table or view can have a maximum of 256 fine-grained audit policies applied to it.
- If object_schema is not specified, the current schema is assumed.
- An FGA policy should not be applied to out-of-line columns such as LOB columns.
- Each audit policy is applied to the query individually. However, at most one audit record may be generated for each policy, no matter how many rows being returned satisfy that policy's audit_condition. In other words, whenever any number of rows being returned satisfy an audit condition defined on the table, a single audit record will be generated for each such policy.
- If a table with an FGA policy defined on it receives a Fast Path insert or a vectored update, the hint is automatically disabled before any such operations. Disabling the hint allows auditing to occur according to the policy's terms. (One example of a Fast Path insert is the statement `INSERT-WITH-APPEND-hint`.)
- The audit_condition must be a boolean expression that can be evaluated using the values in the row being inserted, updated, or deleted. The expression can also use functions, such as the USER or SYS_CONTEXT functions. The expression must not combine conditions using operators such as AND and OR. audit_condition can be NULL (or omitted), which is interpreted as TRUE, but it cannot contain the following elements:
  - Subqueries or sequences
  - The following attributes of the USERENV namespace when accessed using the SYS_CONTEXT function:
    * CURRENT_SQL
    * CURRENT_SQL_LENGTH
    * CURRENT_BIND
  - Any use of the pseudo columns LEVEL, PRIOR, or ROWNUM.

Specifying an audit condition of "1=1" to force auditing of all specified statements ("statement_types") affecting the specified column ("audit_column") is no longer needed to achieve this purpose. A NULL value for audit_condition causes audit to happen even if no rows are processed, so that all actions on a table with this policy are audited.
- The audit_condition is evaluated using the privileges of the user who creates the policy.
- For the audit_condition setting, do not include functions, which execute the auditable statement on the same base table, in the audit_condition setting. For example,
suppose you create a function that executes an `INSERT` statement on the `HR.EMPLOYEES` table. The policy `audit_condition` contains this function and it is for `INSERT` statements (as set by the `statement_types` parameter). When the policy is used, the function executes recursively until the system has run out of memory. This can raise the error ORA-1000: maximum open cursors exceeded or ORA-00036: maximum number of recursive SQL levels (50) exceeded.

- Do not issue the `DBMS_FGA.ENABLE_POLICY` or `DBMS_FGA.DISABLE_POLICY` statement from a policy function in a condition.

- The audit function (`handle_module`) is an alerting mechanism for the administrator. The required interface for such a function is as follows:

```sql
PROCEDURE fname ( object_schema VARCHAR2, object_name VARCHAR2, policy_name VARCHAR2 ) AS ...
```

where `fname` is the name of the procedure, `object_schema` is the name of the schema of the table audited, `object_name` is the name of the table to be audited, and `policy_name` is the name of the policy being enforced. The audit function will be executed with the function owner’s privilege.

- If you have migrated to unified auditing, then omit the `audit_trail` parameter because the audit records will automatically be written to the unified audit trail.

- Be aware that sensitive data, such as credit card information, can be recorded in clear text.

- The `audit_trail` parameter, if used, specifies both where the fine-grained audit trail will be written and whether it is to include the query’s SQL Text and SQL Bind variable information (typically in columns named `LSQLTEXT` and `LSQLBIND`):
  - If `audit_trail` includes XML, then fine-grained audit records are written to XML-format operating system files stored in the directory specified by an `AUDIT_FILE_DEST` statement in SQL. (The default `AUDIT_FILE_DEST` is `$ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump` on Unix-based systems, and `$ORACLE_BASE\admin\$DB_UNIQUE_NAME\adump` on Windows systems.)
  - If `audit_trail` includes `DB` instead, then the audit records are written to the `SYS.FGA_LOG$` table in the database. However, for read-only databases, Oracle Database writes the fine-grained audit records to XML files, regardless of the `audit_trail` settings.
  - If `audit_trail` includes `EXTENDED`, then the query’s SQL Text and SQL Bind variable information are included in the audit trail.
  - For example:
    - Setting `audit_trail` to `DBMS_FGA.DB` sends the audit trail to the `SYS.FGA_LOG$` table in the database and omits SQL Text and SQL Bind.
    - Setting `audit_trail` to `DBMS_FGA.DB + DBMS_FGA.EXTENDED` sends the audit trail to the `SYS.FGA_LOG$` table in the database and includes SQL Text and SQL Bind.
    - Setting `audit_trail` to `DBMS_FGA.XML` writes the audit trail in XML files sent to the operating system and omits SQL Text and SQL Bind.
    - Setting `audit_trail` to `DBMS_FGA.XML + DBMS_FGA.EXTENDED` writes the audit trail in XML files sent to the operating system and includes SQL Text and SQL Bind.
The `audit_trail` parameter appears in the `ALL_AUDIT_POLICIES` view.

- You can change the operating system destination using the following command:
  
  ```sql
  ALTER SYSTEM SET AUDIT_FILE_DEST = New Directory DEFERRED
  ```

- On many platforms, XML audit files are named `process_name_processId.xml`, for example, `ora_2111.xml`. Alternatively, on Windows, the XML audit files are named `process_name_ThreadId.xml` (or `process_name_ProcessId.xml` if the process is not running as a thread).

- The `audit_column_opts` parameter establishes whether a statement is audited
  - when the query references *any* column specified in the `audit_column` parameter
    `(audit_column_opts = DBMS_FGA.ANY_COLUMNS)`, or
  - only when *all* such columns are referenced `(audit_column_opts = DBMS_FGA.ALL_COLUMNS)`. The default is `DBMS_FGA.ANY_COLUMNS`.

  The `ALL_AUDIT_POLICIES` view also shows `audit_column_opts`.

- When `audit_column_opts` is set to `DBMS_FGA.ALL_COLUMNS`, a SQL statement is audited only when all the columns mentioned in `audit_column` have been explicitly referenced in the statement. And these columns must be referenced in the same SQL-statement or in the sub-select.

  All these columns must refer to a single table/view or alias.

  If a SQL statement selects the columns from different table aliases, the statement will not be audited.

- Every XML audit record contains the elements `AUDIT_TYPE` and `EXTENDED_TIMESTAMP`, with the latter printed in UTC zone (with no timezone information). Values retrieved using `V$XML_AUDIT_TRAIL` view are converted to session timezone and printed.

- For `SQL_TEXT` and `SQL_BIND` element values (`CLOB` type columns), the dynamic view shows only the first 4000 characters. The underlying XML file may have more than 4000 characters for such `SQL_TEXT` and `SQL_BIND` values.

- For large numbers of XML audit files, querying `V$XML_AUDIT_TRAIL` is faster when they are loaded into a database table using SQL*Loader or a similar tool. XML audit files are larger than the equivalent written to OS files when `AUDIT_TRAIL=OS`.

- Error handling is the same as when `AUDIT_TRAIL=OS`. If any error occurs in writing an audit record to disk, including the directory identified by `AUDIT_FILE_DEST` being full, the auditing operation fails. An alert message is logged.

- The policy event handler module will be executed with the module owner's privilege.

- Do not create recursive fine-grained audit handlers. For example, suppose you create a handler that executes an INSERT statement on the `HR.EMPLOYEES` table. The policy that is associated with this handler is for INSERT statements (as set by the `statement_types` parameter). When the policy is used, the handler executes recursively until the system has run out of memory. This can raise the error ORA-1000: maximum open cursors exceeded or ORA-00036: maximum number of recursive SQL levels (50) exceeded. See also Oracle Database Security Guide with regard to Creating a Fine-Grained Audit Policy.

- The fine-grained audit handler module should not have explicit `COMMIT`, `ROLLBACK`, and `DDL` statements mentioned in it.
The values for the `audit_trail` parameter (XML and XML+EXTENDED) cause fine-grained auditing records to be written to operating system files in XML format. A dynamic view, V$XML_AUDIT_TRAIL, makes such audit records from XML files available to DBAs through SQL query, providing enhanced usability. Querying this view causes all XML files (all files with an .xml extension) in the AUDIT_FILE_DEST directory to be parsed and presented in relational table format.

Audit records stored in operating system files can be more secure than database-stored audit records because access can require file permissions that DBAs do not have. Operating system storage for audit records also offers higher availability, since such records remain available even if the database is temporarily inaccessible.

The DBA_COMMON_AUDIT_TRAIL view includes the contents of the V$XML_AUDIT_TRAIL dynamic view for standard and fine-grained audit records.

Note that the V$XML_AUDIT_TRAIL view is populated only if unified auditing is not enabled. If you have enabled unified auditing, then you can query the UNIFIED_AUDIT_TRAIL data dictionary view for the audit trail records.

**See Also:**

*Oracle Database Security Guide* for an example of creating an email alert handler for a fine-grained audit policy

### Examples

DBMS_FGA.ADD_POLICY (  
    object_schema      =>  'scott',  
    object_name        =>  'emp',  
    policy_name        =>  'mypolicy1',  
    audit_condition    =>  'sal < 100',  
    audit_column       =>  'comm,sal',  
    handler_schema     =>   NULL,  
    handler_module     =>   NULL,  
    enable             =>   TRUE,  
    statement_types    =>  'INSERT, UPDATE',  
    audit_column_opts  =>   DBMS_FGA.ANY_COLUMNS,  
    policy_owner       =>  'sec_admin');

### 65.3.2 DISABLE_POLICY Procedure

This procedure disables an audit policy.

**Syntax**

DBMS_FGA.DISABLE_POLICY (  
    object_schema IN VARCHAR2,  
    object_name IN VARCHAR2,  
    policy_name IN VARCHAR2);
### Parameters

#### Table 65-3  DISABLE_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema of the object to be audited. If NULL, the current schema is assumed.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>Unique name of the policy</td>
</tr>
</tbody>
</table>

The default value for object_schema is NULL. If NULL, the current schema is assumed.

#### Examples

```sql
DBMS_FGA.DISABLE_POLICY (
    object_schema => 'scott',
    object_name   => 'emp',
    policy_name   => 'mypolicy1');
```

### 65.3.3 DROP_POLICY Procedure

This procedure drops an audit policy.

#### Syntax

```sql
DBMS_FGA.DROP_POLICY(
    object_schema IN VARCHAR2,
    object_name   IN VARCHAR2,
    policy_name   IN VARCHAR2);
```

#### Parameters

#### Table 65-4  DROP_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema of the object to be audited. If NULL, the current schema is assumed.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>Unique name of the policy</td>
</tr>
</tbody>
</table>

#### Usage Notes

The **DBMS_FGA** procedures cause current DML transactions, if any, to commit before the operation unless they are inside a DDL event trigger. With DDL transactions, the **DBMS_FGA** procedures are part of the DDL transaction. The default value for object_schema is **NULL**. If **NULL**, the current schema is assumed.
Note:
Oracle Database automatically drops the audit policy if you remove the object specified in the `object_name` parameter of the `DBMS_FGA.ADD_POLICY` procedure, or if you drop the user who created the audit policy.

Examples

```sql
DBMS_FGA.DROP_POLICY (
    object_schema   =>  'scott',
    object_name     =>  'emp',
    policy_name     =>  'mypolicy1');
```

65.3.4 ENABLE_POLICY Procedure

This procedure enables an audit policy.

Syntax

```sql
DBMS_FGA.ENABLE_POLICY(
    object_schema  IN  VARCHAR2,
    object_name    IN  VARCHAR2,
    policy_name    IN  VARCHAR2,
    enable         IN  BOOLEAN);
```

Parameters

Table 65-5  ENABLE_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema of the object to be audited. If <code>NULL</code>, the current schema is assumed.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object to be audited</td>
</tr>
<tr>
<td>policy_name</td>
<td>Unique name of the policy</td>
</tr>
<tr>
<td>enable</td>
<td>Defaults to <code>TRUE</code> to enable the policy</td>
</tr>
</tbody>
</table>

Examples

```sql
DBMS_FGA.ENABLE_POLICY (
    object_schema   =>  'scott',
    object_name     =>  'emp',
    policy_name     =>  'mypolicy1',
    enable          =>   TRUE);
```
The DBMS_FILE_GROUP package, one of a set of Oracle Streams packages, provides administrative interfaces for managing file groups, file group versions, and files. A file group repository is a collection of all of the file groups in a database and can contain multiple versions of a particular file group. You can use this package to create and manage file group repositories.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Examples
- Summary of DBMS_FILE_GROUP Subprograms

66.1 DBMS_FILE_GROUP Overview

It is helpful to understand terminology before using the DBMS_FILE_GROUP package.

The following terms pertain to the DBMS_FILE_GROUP package:

File

A file is a reference to a file stored on hard disk. A file is composed of a file name, a directory object, and a file type. The directory object references the directory in which the file is stored on hard disk. For example, a file might have the following components:

- The file name is expdat.dmp.
- The directory object that contains the file is db_files.
- The file type is DBMS_FILE_GROUP.EXPORT_DUMP_FILE.

Version

A version is a collection of related files. For example, a version might consist of a set of data files and a Data Pump export dump file generated by a Data Pump transportable tablespace export. Only one Data Pump export dump file is allowed in a version.

File Group

A file group is a collection of versions. A file group can logically group a set of versions. For example, a file group named financial_quarters can keep track of quarterly financial data by logically grouping versions of files related to a tablespace set. The tablespaces containing the data can be exported at the end of each quarter and versioned under names such as Q1FY04, Q2FY04, and so on.
66.2 DBMS_FILE_GROUP Security Model

There are two ways to define control on the DBMS_FILE_GROUP package.

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

66.3 DBMS_FILE_GROUP Constants

The DBMS_FILE_GROUP package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS_FILE_GROUP.EXPORT_DUMP_FILE.

The following table lists the parameters and enumerated constants.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_type</td>
<td>· DATAFILE</td>
<td>VARCHAR2(30)</td>
<td>DATAFILE is a datafile for a database. This constant can be specified as 'DATAFILE'.</td>
</tr>
<tr>
<td></td>
<td>· EXPORT_DUMP_FILE</td>
<td></td>
<td>EXPORT_DUMP_FILE is a Data Pump export dump file. This constant can be specified as 'DUMPSET'.</td>
</tr>
<tr>
<td></td>
<td>· DATAPUMP_LOG_FILE</td>
<td></td>
<td>DATAPUMP_LOG_FILE is a Data Pump export log file. This constant can be specified as 'DATAPUMPLOG'.</td>
</tr>
<tr>
<td>new_file_type</td>
<td>· DATAFILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>· EXPORT_DUMP_FILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>· DATAPUMP_LOG_FILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_versions</td>
<td>· INFINITE</td>
<td>NUMBER</td>
<td>INFINITE specifies no limit. The max_versions or retention_days can increase without reaching a limit.</td>
</tr>
<tr>
<td>retention_days</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 66-1  (Cont.) DBMS_FILE_GROUP Parameters with Enumerated Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>System privilege specified in the GRANT_SYSTEM_PRIVILEGE procedure:</td>
<td>BINARY_INTEGER</td>
<td>READ_ANY_FILE_GROUP grants the privilege to view information about any file group in any schema in the data dictionary.</td>
</tr>
<tr>
<td></td>
<td>· READ_ANY_FILE_GROUP</td>
<td></td>
<td>MANAGE_ANY_FILE_GROUP grants the privilege to create, manage, and drop any file group in any schema.</td>
</tr>
<tr>
<td></td>
<td>· MANAGE_FILE_GROUP</td>
<td></td>
<td>MANAGE_FILE_GROUP grants the privilege to create, manage, and drop file groups in the user's schema.</td>
</tr>
<tr>
<td>Object privilege specified in the GRANT_OBJECT_PRIVILEGE procedure:</td>
<td></td>
<td></td>
<td>READ_ON_FILE_GROUP grants the privilege to view information about a specific file group in the data dictionary.</td>
</tr>
<tr>
<td></td>
<td>· READ_ON_FILE_GROUP</td>
<td></td>
<td>MANAGE_ON_FILE_GROUP grants the privilege to manage a specific file group in a schema other than the user's schema.</td>
</tr>
<tr>
<td></td>
<td>· MANAGE_ON_FILE_GROUP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

66.4 DBMS_FILE_GROUP Examples

DBMS_FILE_GROUP examples are in another book.

Oracle Streams Concepts and Administration includes an example of a business that sells books and music over the internet. The business runs weekly reports on the sales data in the inst1.example.com database and stores these reports in two HTML files on a computer file system. The book_sales.htm file contains the report for book sales, and the music_sales.htm file contains the report for music sales.

The business wants to store these weekly reports in a file group repository at the inst2.example.com remote database. Every week, the two reports are generated on the inst1.example.com database, transferred to the computer system running the inst2.example.com database, and added to the repository as a file group version. The file group repository stores all of the file group versions that contain the reports for each week.

66.5 Summary of DBMS_FILE_GROUP Subprograms

This table lists the DBMS_FILE_GROUP subprograms and briefly describes them.

Table 66-2  DBMS_FILE_GROUP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_FILE Procedure</td>
<td>Adds a file to a version of a file group</td>
</tr>
<tr>
<td>ALTER_FILE Procedure</td>
<td>Alters a file in a version of a file group</td>
</tr>
<tr>
<td>ALTER_FILE_GROUP Procedure</td>
<td>Alters a file group</td>
</tr>
<tr>
<td>ALTER_VERSION Procedure</td>
<td>Alters a version of a file group</td>
</tr>
</tbody>
</table>
Table 66-2  (Cont.) DBMS_FILE_GROUP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_FILE_GROUP Procedure</td>
<td>Creates a file group</td>
</tr>
<tr>
<td>CREATE_VERSION Procedure</td>
<td>Creates a version of a file group</td>
</tr>
<tr>
<td>DROP_FILE_GROUP Procedure</td>
<td>Drops a file group</td>
</tr>
<tr>
<td>DROP_VERSION Procedure</td>
<td>Drops a version of a file group</td>
</tr>
<tr>
<td>GRANT_OBJECT_PRIVILEGE Procedure</td>
<td>Grants object privileges on a file group to a user</td>
</tr>
<tr>
<td>GRANT_SYSTEM_PRIVILEGE Procedure</td>
<td>Grants system privileges for file group operations to a user</td>
</tr>
<tr>
<td>PURGE_FILE_GROUP Procedure</td>
<td>Purges a file group using the file group's retention policy</td>
</tr>
<tr>
<td>REMOVE_FILE Procedure</td>
<td>Removes a file from a version of a file group</td>
</tr>
<tr>
<td>REVOKE_OBJECT_PRIVILEGE Procedure</td>
<td>Revokes object privileges on a file group from a user</td>
</tr>
<tr>
<td>REVOKE_SYSTEM_PRIVILEGE Procedure</td>
<td>Revokes system privileges for file group operations from a user</td>
</tr>
</tbody>
</table>

Note:
All subprograms commit unless specified otherwise.

66.5.1 ADD_FILE Procedure

This procedure adds a file to a version of a file group.

See Also:
An example that uses this procedure for a file group repository in *Oracle Streams Concepts and Administration*

Syntax

```
DBMS_FILE_GROUP.ADD_FILE(
    file_group_name  IN  VARCHAR2,
    file_name        IN  VARCHAR2,
    file_type        IN  VARCHAR2  DEFAULT NULL,
    file_directory   IN  VARCHAR2  DEFAULT NULL,
    version_name     IN  VARCHAR2  DEFAULT NULL,
    comments         IN  VARCHAR2  DEFAULT NULL);
```
Parameters

Table 66-3  ADD_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group that contains the version, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file being added to the version. Each file name in a version must be unique.</td>
</tr>
<tr>
<td>file_type</td>
<td>The file type. The following are reserved file types:</td>
</tr>
<tr>
<td></td>
<td>- If the file is a datafile, then enter the following:</td>
</tr>
<tr>
<td></td>
<td>'DATAFILE'</td>
</tr>
<tr>
<td></td>
<td>- If the file is a Data Pump export dump file, then enter the following:</td>
</tr>
<tr>
<td></td>
<td>'DUMPSET'</td>
</tr>
<tr>
<td></td>
<td>Data Pump metadata is populated when a Data Pump export dump file is imported.</td>
</tr>
<tr>
<td></td>
<td>- If the file is a Data Pump export log file, then enter the following:</td>
</tr>
<tr>
<td></td>
<td>'DATAPUMPLOG'</td>
</tr>
<tr>
<td></td>
<td>If the file type is not one of the reserved file types, then either enter a text description of the file type, or specify NULL to omit a file type description.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Constants&quot; for more information about the reserved file types.</td>
</tr>
<tr>
<td>file_directory</td>
<td>The name of the directory object that corresponds to the directory containing the file.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure uses the default directory object for the version.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no default directory object exists for the version, then the procedure uses the default directory object for the file group.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no default directory object exists for the version or file group, then the procedure raises an error.</td>
</tr>
<tr>
<td>version_name</td>
<td>The name of the version to which the file is added.</td>
</tr>
<tr>
<td></td>
<td>If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file is added to version 1 of the file group.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure uses the version with the latest creation time for the file group.</td>
</tr>
<tr>
<td>comments</td>
<td>Comments about the file being added</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure with either DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET' specified for the file_type parameter, a user must meet the following requirements:

- Have the appropriate privileges to import the Data Pump export dump file
- Have READ privilege on the directory object that contains the Data Pump export dump file
66.5.2 ALTER_FILE Procedure

This procedure alters a file in a version of a file group.

Syntax

```
DBMS_FILE_GROUP.ALTER_FILE(
    file_group_name     IN  VARCHAR2,
    file_name           IN  VARCHAR2,
    version_name        IN  VARCHAR2  DEFAULT NULL,
    new_file_name       IN  VARCHAR2  DEFAULT NULL,
    new_file_directory  IN  VARCHAR2  DEFAULT NULL,
    new_file_type       IN  VARCHAR2  DEFAULT NULL,
    remove_file_type    IN  VARCHAR2  DEFAULT 'N',
    new_comments        IN  VARCHAR2  DEFAULT NULL,
    remove_comments     IN  VARCHAR2  DEFAULT 'N');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group that contains the version, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file being altered in the version.</td>
</tr>
<tr>
<td>version_name</td>
<td>The name of the version that contains the file being altered. If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file in version 1 of the file group is altered. If NULL, then the procedure uses the version with the latest creation time for the file group.</td>
</tr>
<tr>
<td>new_file_name</td>
<td>The new name of the file if the file name is being changed. Each file name in a version must be unique. If NULL, then the procedure does not change the file name. Note: When a non-NULL new file name is specified, this procedure changes the metadata for the file name in the data dictionary, but it does not change the file name on the hard disk.</td>
</tr>
<tr>
<td>new_file_directory</td>
<td>The new name of the directory object that corresponds to the directory containing the file, if the directory object is being changed. If NULL, then the procedure does not change the directory object name. Note: When a non-NULL new file directory is specified, this procedure changes the metadata for the file directory in the data dictionary, but it does not change the file directory on the hard disk.</td>
</tr>
</tbody>
</table>
Table 66-4  (Cont.) ALTER_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_file_type</td>
<td>The file type. The following are reserved file types:</td>
</tr>
<tr>
<td></td>
<td>• If the file is a datafile, then enter the following:</td>
</tr>
<tr>
<td></td>
<td>‘DATAFILE’</td>
</tr>
<tr>
<td></td>
<td>• If the file is a Data Pump export dump file, then enter the following:</td>
</tr>
<tr>
<td></td>
<td>‘DUMPSET’</td>
</tr>
<tr>
<td></td>
<td>• If the file is a Data Pump export log file, then enter the following:</td>
</tr>
<tr>
<td></td>
<td>‘DATAPUMPLOG’</td>
</tr>
<tr>
<td></td>
<td>If the file type is not one of the reserved file types, then enter a text description of the file type.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not change the file type.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;Constants&quot; for more information about the reserved file types.</td>
</tr>
<tr>
<td>remove_file_type</td>
<td>If Y, then the procedure removes the file type. If Y and the new_file_type parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If N, then the procedure does not remove the file type.</td>
</tr>
<tr>
<td>new_comments</td>
<td>New comments about the file being altered. If non-NULL, then the procedure replaces the existing comments with the specified comments.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not change the existing comments.</td>
</tr>
<tr>
<td>remove_comments</td>
<td>If Y, then the procedure removes the comments for the file. If Y and the new_comments parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If N, then the procedure does not change the existing comments.</td>
</tr>
</tbody>
</table>

Usage Notes

If the file type is changed to DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET', then Data Pump metadata for the file is populated. If the file type is changed from DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET', then Data Pump metadata for the file is purged.

To run this procedure with DBMS_FILE_GROUP.EXPORT_DUMP_FILE or 'DUMPSET' specified for the new_file_type parameter, a user must meet the following requirements:

• Have the appropriate privileges to import the Data Pump export dump file
• Have READ privilege on the directory object that contains the Data Pump export dump file

See Also:

Oracle Database Utilities for more information about Data Pump privileges
66.5.3 ALTER_FILE_GROUP Procedure

This procedure alters a file group.

Syntax

```
DBMS_FILE_GROUP.ALTER_FILE_GROUP(
    file_group_name           IN  VARCHAR2,
    keep_files                IN  VARCHAR2  DEFAULT NULL,
    min_versions              IN  NUMBER    DEFAULT NULL,
    max_versions              IN  NUMBER    DEFAULT NULL,
    retention_days            IN  NUMBER    DEFAULT NULL,
    new_default_directory     IN  VARCHAR2  DEFAULT NULL,
    remove_default_directory  IN  VARCHAR2  DEFAULT 'N',
    new_comments              IN  VARCHAR2  DEFAULT NULL,
    remove_comments           IN  VARCHAR2  DEFAULT 'N');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group being altered, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>keep_files</td>
<td>If Y, then the files in the file group are retained on hard disk if the file group or a version of the file group is dropped or purged.</td>
</tr>
<tr>
<td></td>
<td>If N, then the files in the file group are deleted from hard disk if the file group or a version of the file group is dropped or purged.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then this parameter is not changed.</td>
</tr>
<tr>
<td>min_versions</td>
<td>The minimum number of versions to retain. The specified value must be greater than or equal to 1.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not change the min_versions setting for the file group.</td>
</tr>
<tr>
<td>max_versions</td>
<td>The maximum number of versions to retain. The specified value must be greater than or equal to the value specified for min_versions. When the number of versions exceeds the specified max_versions, the oldest version is purged. Specify DBMS_FILE_GROUP.INFINITE for no limit to the number of versions.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not change the max_versions setting for the file group.</td>
</tr>
</tbody>
</table>
### Table 66-5  (Cont.) ALTER_FILE_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retention_days</td>
<td>The maximum number of days to retain a version. The specified value must be greater than or equal to 0 (zero). When the age of a version exceeds the specified retention_days and there are more versions than the number specified in \texttt{min_versions}, the version is purged. The age of a version is calculated by subtracting the creation time from the current time. A decimal value can specify a fraction of a day. For example, 1.25 specifies one day and six hours. Specify \texttt{DBMS_FILE_GROUP.INFINITE} for no limit to the number of days a version can exist. If NULL, then the procedure does not change the retention_days setting for the file group.</td>
</tr>
<tr>
<td>new_default_directory</td>
<td>The default directory object used when files are added to a file group if no directory is specified when the files are added, and no default directory object is specified for the version. If NULL, then the procedure does not change the default directory.</td>
</tr>
<tr>
<td>remove_default_directory</td>
<td>If \texttt{Y}, then the procedure removes the default directory for the file group. If \texttt{Y} and the new_default_directory parameter is set to a non-NULL value, then the procedure raises an error. If \texttt{N}, then the procedure does not remove the default directory for the file group.</td>
</tr>
<tr>
<td>new_comments</td>
<td>Comments about the file group. If non-NULL, then the new comments replace the existing comments for the file group. If NULL, then the procedure does not change the existing comments.</td>
</tr>
<tr>
<td>remove_comments</td>
<td>If \texttt{Y}, then the comments for the file group are removed. If \texttt{Y} and the new_comments parameter is set to a non-NULL value, then the procedure raises an error. If \texttt{N}, then the procedure does not change the comments for the file group.</td>
</tr>
</tbody>
</table>

### Usage Notes

If \texttt{min\_versions} is set to 1, then the only version of the file group can be purged when a new version is added. If the addition of the new version is not complete when the existing version is purged, then there can be a period of time when no version of the file group is available. Therefore, set \texttt{min\_versions} to at least 2 if a version of the file group must be available at all times.
66.5.4 ALTER_VERSION Procedure

This procedure alters a version of a file group.

Syntax

```
DBMS_FILE_GROUP.ALTER_VERSION(
  file_group_name            IN  VARCHAR2,
  version_name               IN  VARCHAR2  DEFAULT NULL,
  new_version_name           IN  VARCHAR2  DEFAULT NULL,
  remove_version_name        IN  VARCHAR2  DEFAULT 'N',
  new_default_directory      IN  VARCHAR2  DEFAULT NULL,
  remove_default_directory   IN  VARCHAR2  DEFAULT 'N',
  new_comments               IN  VARCHAR2  DEFAULT NULL,
  remove_comments            IN  VARCHAR2  DEFAULT 'N');
```

Parameters

Table 66-6  ALTER_VERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group that contains the version, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>version_name</td>
<td>The name of the version being altered. If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then version 1 of the file group is altered. If '*' is specified, then the procedure alters all versions, and the new_version_name parameter must be NULL. If NULL, then the procedure uses the version with the latest creation time for the file group.</td>
</tr>
<tr>
<td>new_version_name</td>
<td>The new name of the version. Do not specify a schema. The specified version name cannot be a positive integer or an asterisk ('*'). If NULL, then the procedure does not change the version name.</td>
</tr>
<tr>
<td>remove_version_name</td>
<td>If Y, then the procedure removes the version name. If the version name is removed, then the version number must be used to manage the version. If Y and the new_version_name parameter is set to a non-NULL value, then the procedure raises an error. If N, then the procedure does not remove the version name.</td>
</tr>
<tr>
<td>new_default_directory</td>
<td>The default directory object used when files are added to a version if no directory is specified when the files are added. If NULL, then the procedure does not change the default directory.</td>
</tr>
</tbody>
</table>
Table 66-6  (Cont.) ALTER_VERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| remove_default_directory | If Y, then the procedure removes the default directory. If Y and the new_default_directory parameter is set to a non-NULL value, then the procedure raises an error.  
If N, then the procedure does not remove the default directory. |
| new_comments            | Comments about the version. If non-NULL, then the new comments replace the existing comments for the version.  
If NULL, then the procedure does not change the comments. |
| remove_comments         | If Y, then the procedure removes the comments for the version. If Y and the new_comments parameter is set to a non-NULL value, then the procedure raises an error.  
If N, then the procedure does not remove the comments for the version. |

66.5.5 CREATE_FILE_GROUP Procedure

This procedure creates a file group.

See Also:

An example that uses this procedure for a file group repository in *Oracle Streams Concepts and Administration*

Syntax

```
DBMS_FILE_GROUP.CREATE_FILE_GROUP(
    file_group_name    IN  VARCHAR2,
    keep_files         IN  VARCHAR2  DEFAULT 'Y',
    min_versions       IN  NUMBER    DEFAULT 2,
    max_versions       IN  NUMBER    DEFAULT DBMS_FILE_GROUP.INFINITE,
    retention_days     IN  NUMBER    DEFAULT DBMS_FILE_GROUP.INFINITE,
    default_directory  IN  VARCHAR2  DEFAULT NULL,
    comments           IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 66-7  CREATE_FILE_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
Table 66-7  (Cont.) CREATE_FILE_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep_files</td>
<td>If Y, then the files in the file group are retained on hard disk if the file group or a version of the file group is dropped or purged. If N, then the files in the file group are deleted from hard disk if the file group or a version of the file group is dropped or purged. <strong>Note:</strong> If the file group is dropped because of a DROP USER CASCADE statement, then the setting of this parameter determines whether the files are dropped from the hard disk.</td>
</tr>
<tr>
<td>min_versions</td>
<td>The minimum number of versions to retain. The specified value must be greater than or equal to 1.</td>
</tr>
<tr>
<td>max_versions</td>
<td>The maximum number of versions to retain. The specified value must be greater than or equal to the value specified for min_versions. When the number of versions exceeds the specified max_versions, the oldest version is purged. Specify DBMS_FILE_GROUP.INFINITE for no limit to the number of versions.</td>
</tr>
<tr>
<td>retention_days</td>
<td>The maximum number of days to retain a version. The specified value must be greater than or equal to 0 (zero). When the age of a version exceeds the specified retention_days and there are more versions than the number specified in min_versions, the version is purged. The age of a version is calculated by subtracting the creation time from the current time. A decimal value can specify a fraction of a day. For example, 1.25 specifies one day and six hours. Specify DBMS_FILE_GROUP.INFINITE for no limit to the number of days a version can exist.</td>
</tr>
<tr>
<td>default_directory</td>
<td>The default directory object used when files are added to a file group if no directory is specified when the files are added, and no default directory object is specified for the version.</td>
</tr>
<tr>
<td>comments</td>
<td>Comments about the file group being created.</td>
</tr>
</tbody>
</table>

**Usage Notes**

If min_versions is set to 1, then the only version of the file group can be purged when a new version is added. If the addition of the new version is not complete when the existing version is purged, then there can be a period of time when no version of the file group is available. Therefore, set min_versions to at least 2 if a version of the file group must be available at all times.

66.5.6 CREATE_VERSION Procedure

This procedure creates a version of a file group.

This procedure automatically runs the PURGE_FILE_GROUP procedure. Therefore, versions can be purged based on the file group’s retention policy.

This procedure is overloaded. One version of the procedure contains the OUT parameter version_out, and the other does not.
See Also:

- PURGE_FILE_GROUP Procedure
- An example that uses this procedure for a file group repository in Oracle Streams Concepts and Administration

Syntax

```sql
DBMS_FILE_GROUP.CREATE_VERSION(
    file_group_name    IN  VARCHAR2,
    version_name       IN  VARCHAR2 DEFAULT NULL,
    default_directory  IN  VARCHAR2 DEFAULT NULL,
    comments           IN  VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_FILE_GROUP.CREATE_VERSION(
    file_group_name    IN   VARCHAR2,
    version_name       IN   VARCHAR2 DEFAULT NULL,
    default_directory  IN   VARCHAR2 DEFAULT NULL,
    comments           IN   VARCHAR2 DEFAULT NULL,
    version_out        OUT  VARCHAR2);
```

Parameters

Table 66-8  CREATE_VERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group to which the new version is added, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>version_name</td>
<td>The name of the version being created. Do not specify a schema. The specified version name cannot be a positive integer because, when a version is created, a version number is generated automatically. The specified version name cannot be an asterisk ('*').</td>
</tr>
<tr>
<td>default_directory</td>
<td>The default directory object used when files are added to a version if no directory is specified when the files are added.</td>
</tr>
<tr>
<td>comments</td>
<td>Comments about the version being created</td>
</tr>
<tr>
<td>version_out</td>
<td>If the version_name parameter is set to a non-NULL value, then this parameter contains the specified version name. If the version_name parameter is set to NULL, then this parameter contains the generated version number.</td>
</tr>
</tbody>
</table>
66.5.7 DROP_FILE_GROUP Procedure

This procedure drops a file group.

Syntax

```
DBMS_FILE_GROUP.DROP_FILE_GROUP(
    file_group_name  IN  VARCHAR2,
    keep_files       IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 66-9  DROP_FILE_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group being dropped, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>keep_files</td>
<td>If Y, then the procedure retains the files in the file group on hard disk. If N, then the procedure deletes the files in the file group from hard disk. If NULL, then the procedure uses the default keep files property of the file group.</td>
</tr>
</tbody>
</table>

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have WRITE privilege on the directory object that contains the files.

66.5.8 DROP_VERSION Procedure

This procedure drops a version of a file group.

Syntax

```
DBMS_FILE_GROUP.DROP_VERSION(
    file_group_name  IN  VARCHAR2,
    version_name     IN  VARCHAR2 DEFAULT NULL,
    keep_files       IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 66-10  DROP_VERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group that contains the version, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
Table 66-10  (Cont.) DROP_VERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| version_name | The name of the version being dropped.  
  If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then version 1 of the file group is dropped.  
  If NULL, then the procedure uses the version with the oldest creation time for the file group.  
  If '*', then the procedure drops all versions. |
| keep_files  | If Y, then the procedure retains the files in the version on hard disk.  
  If N, then the procedure deletes the files in the version from hard disk.  
  If NULL, then the procedure uses the default keep files property of the file group. |

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have WRITE privilege on the directory object that contains the files.

66.5.9 GRANT_OBJECT_PRIVILEGE Procedure

This procedure grants object privileges on a file group to a user.

Syntax

```
DBMS_FILE_GROUP.GRANT_OBJECT_PRIVILEGE(  
  object_name IN VARCHAR2,  
  privilege IN BINARY_INTEGER,  
  grantee IN VARCHAR2,  
  grant_option IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 66-11  GRANT_OBJECT_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the file group on which the privilege is granted, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>privilege</td>
<td>The constant that specifies the privilege. See &quot;Constants&quot; for valid privileges.</td>
</tr>
<tr>
<td>grantee</td>
<td>The name of the user or role for which the privilege is granted. The specified user cannot be the owner of the object.</td>
</tr>
</tbody>
</table>
| grant_option | If TRUE, then the specified user granted the specified privilege can grant this privilege to others.  
  If FALSE, then the specified user granted the specified privilege cannot grant this privilege to others. |
Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the object on which the privilege is granted
- Have the same privilege as the privilege being granted with the grant option

66.5.10 GRANT_SYSTEM_PRIVILEGE Procedure

This procedure grants system privileges for file group operations to a user.

Note:

When you grant a privilege on "ANY" object (for example, ALTER_ANY_RULE), and the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE, you give the user access to that type of object in all schemas, except the SYS schema. By default, the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to FALSE.

If you want to grant access to an object in the SYS schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the O7_DICTIONARY_ACCESSIBILITY initialization parameter to TRUE. Then privileges granted on "ANY" object allows access to any schema, including SYS. Set the O7_DICTIONARY_ACCESSIBILITY initialization parameter with caution.

Syntax

DBMS_FILE_GROUP.GRANT_SYSTEM_PRIVILEGE(
    privilege IN BINARY_INTEGER,
    grantee IN VARCHAR2,
    grant_option IN BOOLEAN DEFAULT FALSE);

Parameters

Table 66-12  GRANT_SYSTEM_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The constant that specifies the privilege. See &quot;Constants&quot; for valid privileges.</td>
</tr>
<tr>
<td>grantee</td>
<td>The name of the user or role for which the privilege is granted. The user who runs the procedure cannot be specified.</td>
</tr>
<tr>
<td>grant_option</td>
<td>If TRUE, then the specified user granted the specified privilege can grant this privilege to others. If FALSE, then the specified user granted the specified privilege cannot grant this privilege to others.</td>
</tr>
</tbody>
</table>
66.5.11 PURGE_FILE_GROUP Procedure

This procedure purges a file group using the file group’s retention policy.

A file group’s retention policy is determined by its settings for the max_versions, min_versions, and retention_days parameters. The following versions of a file group are removed when a file group is purged:

- All versions greater than the max_versions setting for the file group when versions are ordered in descending order by creation time. Therefore, the older versions are purged before the newer versions.
- All versions older than the retention_days setting for the file group unless purging a version would cause the number of versions to drop below the min_versions setting for the file group.

A job named SYS.FGRSAUTOPURGE_JOB automatically purges all file groups in a database periodically according to the job’s schedule. You can adjust this job’s schedule using the DBMS_SCHEDULER package. Alternatively, you can create a job that runs the PURGE_FILE_GROUP procedure periodically.

Syntax

```sql
DBMS_FILE_GROUP.PURGE_FILE_GROUP(
    file_group_name  IN  VARCHAR2);
```

Parameter

Table 66-13  PURGE_FILE_GROUP Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default. If NULL and this procedure is run by SYS user, then the procedure purges all file groups.</td>
</tr>
</tbody>
</table>

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have WRITE privilege on the directory object that contains the files. Files are deleted when a version is purged and the keep_files parameter is set to N for the version’s file group.

66.5.12 REMOVE_FILE Procedure

This procedure removes a file from a version of a file group.

Syntax

```sql
DBMS_FILE_GROUP.REMOVE_FILE(
    file_group_name  IN  VARCHAR2,
    file_name        IN  VARCHAR2,
    version_name     IN  VARCHAR2 DEFAULT NULL,
    keep_file        IN  VARCHAR2 DEFAULT NULL);
```
Parameters

Table 66-14 REMOVE_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_group_name</td>
<td>The name of the file group that contains the version, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file being removed from the version</td>
</tr>
<tr>
<td>version_name</td>
<td>The name of the version from which the file is removed. If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file is removed from version 1 of the file group. If NULL, then the procedure uses the version with the latest creation time for the file group. If '*', then the procedure removes the file from all versions.</td>
</tr>
<tr>
<td>keep_file</td>
<td>If Y, then the procedure retains the file on hard disk. If N, then the procedure deletes the file from hard disk. If NULL, then the procedure uses the default keep files property of the file group.</td>
</tr>
</tbody>
</table>

Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have WRITE privilege on the directory object that contains the files.

66.5.13 REVOKE_OBJECT_PRIVILEGE Procedure

This procedure revokes object privileges on a file group from a user.

Syntax

```
DBMS_FILE_GROUP.REVOKE_OBJECT_PRIVILEGE(
    object_name  IN  VARCHAR2,
    privilege    IN  BINARY_INTEGER,
    revokee      IN  VARCHAR2);
```

Parameters

Table 66-15 REVOKE_OBJECT_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the file group on which the privilege is revoked, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales_tbs, then specify hq_dba.sales_tbs. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>privilege</td>
<td>The constant that specifies the privilege. See &quot;Constants&quot; for valid privileges.</td>
</tr>
</tbody>
</table>
### REVOKE_OBJECT_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>revokee</code></td>
<td>The name of the user or role from which the privilege is revoked. The user who owns the object cannot be specified.</td>
</tr>
</tbody>
</table>

### REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure revokes system privileges for file group operations from a user.

#### Syntax

```sql
DBMS_FILE_GROUP.REVOKE_SYSTEM_PRIVILEGE(
  privilege  IN  BINARY_INTEGER,
  revokee    IN  VARCHAR2);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>privilege</code></td>
<td>The constant that specifies the privilege. See &quot;Constants&quot; for valid privileges.</td>
</tr>
<tr>
<td><code>revokee</code></td>
<td>The name of the user or role from which the privilege is revoked. The user who runs the procedure cannot be specified.</td>
</tr>
</tbody>
</table>
The `DBMS_FILE_TRANSFER` package provides procedures to copy a binary file within a database or to transfer a binary file between databases.

This chapter contains the following topics:

- Overview
- Security Model
- Operating Notes
- Summary of `DBMS_FILE_TRANSFER` Subprograms

**See Also:**
- *Oracle Database Administrator's Guide* for instructions about using file transfer
- *Oracle Streams Concepts and Administration* for applications of file transfer.

### 67.1 DBMS_FILE_TRANSFER Overview

The `DBMS_FILE_TRANSFER` package provides procedures to copy a binary file within a database or to transfer a binary file between databases.

The destination database converts each block when it receives a file from a platform with different endianness. Datafiles can be imported after they are moved to the destination database as part of a transportable operation without RMAN conversion. Both `GET` and `PUT` operations will converted the file across platform difference at the destination. However, `COPY` is a local operation and therefore no conversion is required.

### 67.2 DBMS_FILE_TRANSFER Security Model

The `DBMS_FILE_TRANSFER` package must be created under `SYS (CONNECT INTERNAL)`. Operations provided by this package are performed under the current calling user, not the package owner (`SYS`).

To use this interface the following users must have the following privileges:

- The current user at the local database must have `READ` privilege on the directory object specified in the `source_directory_object` parameter.
- The connected user at the destination database must have `WRITE` privilege to the directory object specified in the `destination_directory_object` parameter.
67.3 DBMS_FILE_TRANSFER Operational Notes

DBMS_FILE_TRANSFER supports online backup. You should therefore be careful in copying or transferring a file that is being modified by the database because this can result in an inconsistent file, and require recovery. To guarantee consistency, bring files offline when the database is in use.

If you want to use DBMS_FILE_TRANSFER for performing backups, note that you are implementing self-managed backups, and should therefore put the files in hot backup mode.

67.4 Summary of DBMS_FILE_TRANSFER Subprograms

This table lists the DBMS_FILE_TRANSFER subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY_FILE Procedure</td>
<td>Reads a file from a source directory and creates a copy of it in a destination directory. The source and destination directories can both be in a local file system, or both be in an Automatic Storage Management (ASM) disk group, or between local file system and ASM with copying in either direction.</td>
</tr>
<tr>
<td>GET_FILE Procedure</td>
<td>Contacts a remote database to read a remote file and then creates a copy of the file in the local file system or ASM.</td>
</tr>
<tr>
<td>PUT_FILE Procedure</td>
<td>Reads a local file or ASM and contacts a remote database to create a copy of the file in the remote file system.</td>
</tr>
</tbody>
</table>

67.4.1 COPY_FILE Procedure

This procedure reads a file from a source directory and creates a copy of it in a destination directory. The source and destination directories can both be in a local file system, or both be in an Automatic Storage Management (ASM) disk group, or between local file system and ASM with copying in either direction.

You can copy any type of file to and from a local file system. However, you can copy only database files (such as datafiles, tempfiles, controlfiles, and so on) to and from an ASM disk group.

The destination file is not closed until the procedure completes successfully.

**Syntax**

```sql
DBMS_FILE_TRANSFER.COPY_FILE(
    source_directory_object       IN  VARCHAR2,
    source_file_name              IN  VARCHAR2,
    destination_directory_object  IN  VARCHAR2,
    destination_file_name         IN  VARCHAR2);
```
Parameters

Table 67-2  COPY_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_directory_object</td>
<td>The directory object that designates the source directory. The directory object must already exist. You create directory objects with the CREATE DIRECTORY command.</td>
</tr>
<tr>
<td>source_file_name</td>
<td>The name of the file to copy. This file must exist in the source directory.</td>
</tr>
<tr>
<td>destination_directory_object</td>
<td>The directory object that designates the destination directory. The directory object must already exist. If the destination is ASM, the directory object must designate either a disk group name (for example, +diskgroup1) or a directory created for alias names. In the case of a directory, the full path to the directory must be specified (for example: +diskgroup1/dbs/control).</td>
</tr>
</tbody>
</table>
| destination_file_name      | The name to assign to the file in the destination directory. A file with the same name must not exist in the destination directory. If the destination is ASM:  
  • The file is given a fully qualified ASM filename and created in the appropriate directory (depending on the database name and file type)  
  • The file type tag assigned to the file is COPY_FILE  
  • The value of the destination_file_name argument becomes the file's alias name in the designated destination directory  
  The file name can be followed by an ASM template name in parentheses. The file is then given the attributes specified by the template. |

Usage Notes

To run this procedure successfully, the current user must have the following privileges:

- READ privilege on the directory object specified in the source_directory_object parameter
- WRITE privilege on directory object specified in the destination_directory_object parameter

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

The source_file_name parameter must specify a file that is in the directory specified by the source_directory_object parameter before running the procedure, and the destination_file_name parameter must specify the new name of the file in the new location specified in the destination_directory_object parameter. Relative paths and symbolic
links are not allowed in the directory objects for the `source_directory_object` and `destination_directory_object` parameters.

Transferring the file is not transactional. To monitor the progress of a long file copy, query the `V$SESSION_LONGOPS` dynamic performance view.

See Also:

Oracle Automatic Storage Management Administrator’s Guide for instructions about using file transfer

Examples

```sql
SQL> create directory DGROUP as '+diskgroup1/dbs/backup';
Directory created.

SQL> BEGIN
2    DBMS_FILE_TRANSFER.COPY_FILE('SOURCEDIR','t_xdbtmp.f', 'DGROUP',
                                      't_xdbtmp.f');
3  END;
4  /

PL/SQL procedure successfully completed.

SQL> EXIT
$ASMCMD
ASMCMD> ls
DISKGROUP1/
ASMCMD> cd diskgroup1/dbs/backup
ASMCMD> ls
  t_xdbtmp.f => +DISKGROUP1/ORCL/TEMPFILE/COPY_FILE.267.546546525

67.4.2 GET_FILE Procedure

This procedure contacts a remote database to read a remote file and then creates a copy of the file in the local file system or ASM. The file that is copied is the source file, and the new file that results from the copy is the destination file. The destination file is not closed until the procedure completes successfully.

Syntax

```sql
DBMS_FILE_TRANSFER.GET_FILE
    source_directory_object      IN  VARCHAR2,
    source_file_name             IN  VARCHAR2,
    source_database              IN  VARCHAR2,
    destination_directory_object IN  VARCHAR2,
    destination_file_name        IN  VARCHAR2};
```
Parameters

Table 67-3 GET_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_directory_object</td>
<td>The directory object from which the file is copied at the source site. This directory object must exist at the source site.</td>
</tr>
<tr>
<td>source_file_name</td>
<td>The name of the file that is copied in the remote file system. This file must exist in the remote file system in the directory associated with the source directory object.</td>
</tr>
<tr>
<td>source_database</td>
<td>The name of a database link to the remote database where the file is located.</td>
</tr>
<tr>
<td>destination_directory_object</td>
<td>The directory object into which the file is placed at the destination site. This directory object must exist in the local file system.</td>
</tr>
<tr>
<td>destination_file_name</td>
<td>The name of the file copied to the local file system. A file with the same name must not exist in the destination directory in the local file system.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure successfully, the following users must have the following privileges:

- The connected user at the source database must have read privilege on the directory object specified in the source_directory_object parameter.
- The current user at the local database must have write privilege on the directory object specified in the destination_directory_object parameter.

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. To monitor the progress of a long file transfer, query the V$SESSION_LONGOPS dynamic performance view.

Examples

```sql
CREATE OR REPLACE DIRECTORY df AS '+datafile' ;
GRANT WRITE ON DIRECTORY df TO "user";
CREATE DIRECTORY DSK_FILES AS '"^t_work^' ;
GRANT WRITE ON DIRECTORY dsk_files TO "user" ;

-- assumes that dbs2 link has been created and we are connected to the instance.
-- dbs2 could be a loopback or point to another instance.
BEGIN
  -- asm file to an os file
  -- get an asm file from dbs1.asm/a1 to dbs2.^t_work^/oa5.dat
  DBMS_FILE_TRANSFER.GET_FILE ( 'df' , 'a1' , 'dbs1' , 'dsk_files' , 'oa5.dat' ) ;
```
67.4.3 PUT_FILE Procedure

This procedure reads a local file or ASM and contacts a remote database to create a copy of the file in the remote file system.

The file that is copied is the source file, and the new file that results from the copy is the destination file. The destination file is not closed until the procedure completes successfully.

Syntax

```
DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object       IN  VARCHAR2,
    source_file_name              IN  VARCHAR2,
    destination_directory_object  IN  VARCHAR2,
    destination_file_name         IN  VARCHAR2,
    destination_database          IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_directory_object</td>
<td>The directory object from which the file is copied at the local source site.</td>
</tr>
<tr>
<td>source_file_name</td>
<td>The name of the file that is copied from the local file system.</td>
</tr>
<tr>
<td>destination_directory_object</td>
<td>The directory object into which the file is placed at the destination site.</td>
</tr>
<tr>
<td>destination_file_name</td>
<td>The name of the file placed in the remote file system.</td>
</tr>
<tr>
<td>destination_database</td>
<td>The name of a database link to the remote database to which the file is copied.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure successfully, the following users must have the following privileges:

- The current user at the local database must have read privilege on the directory object specified in the `source_directory_object` parameter.
• The connected user at the destination database must have write privilege to the directory object specified in the `destination_directory_object` parameter.

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

• The size of the copied file must be a multiple of 512 bytes.
• The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. To monitor the progress of a long file transfer, query the `V$SESSION_LONGOPS` dynamic performance view.

**Examples**

```sql
CREATE OR REPLACE DIRECTORY df AS '+datafile' ;
GRANT WRITE ON DIRECTORY df TO "user";
CREATE OR REPLACE DIRECTORY ft1 AS '+datafile/ft1' ;
GRANT READ,WRITE ON DIRECTORY ft1 TO "user";
CREATE OR REPLACE DIRECTORY ft1_1 AS '+datafile/ft1/ft1_1' ;

CONNECT user;
Enter password: password

-- - put a1.dat to a4.dat (using dbs2 dblink)
-- - level 2 sub dir to parent dir
-- - user has read privs on ft1_1 at dbs1 and write on df in dbs2
BEGIN
DBMS_FILE_TRANSFER.PUT_FILE ( 'ft1_1' , 'a2.dat' , 'df' , 'a4.dat' ,
                      'dbs2' ) ;
END ;
```
Using DBMS_FLASHBACK, you can flash back to a version of the database at a specified time or a specified system change number (SCN).

This chapter contains the following topics:

- Overview
- Security Model
- Types
- Exceptions
- Operational Notes
- Examples
- Summary of DBMS_FLASHBACK Subprograms

**See Also:**

For detailed information about DBMS_FLASHBACK:

- Oracle Database Development Guide
- Oracle Database SQL Language Reference.

### 68.1 DBMS_FLASHBACK Overview

DBMS_FLASHBACK provides an interface for the user to view the database at a particular time in the past, with the additional capacity provided by transaction back out features that allow for selective removal of the effects of individual transactions. This is different from a flashback database which moves the database back in time.

When DBMS_FLASHBACK is enabled, the user session uses the Flashback version of the database, and applications can execute against the Flashback version of the database.

You may want to use DBMS_FLASHBACK for the following reasons:

- Self-service repair: If you accidentally delete rows from a table, you can recover the deleted rows.
- Packaged applications such as email and voicemail: You can use Flashback to restore deleted email by re-inserting the deleted message into the current message box.
- Decision support system (DSS) and online analytical processing (OLAP) applications: You can perform data analysis or data modeling to track seasonal demand.
68.2 DBMS_FLASHBACK Security Model

To use the DBMS_FLASHBACK package, you must have the EXECUTE privilege on it.

68.3 DBMS_FLASHBACK Types

The following table describes the types used by DBMS_FLASHBACK.

Table 68-1  DBMS_FLASHBACK

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXNAME_ARRAY</td>
<td>Creates a VARRAY for holding Transaction Names or Identifiers (XIDs)</td>
</tr>
</tbody>
</table>

68.4 DBMS_FLASHBACK Exceptions

DBMS_FLASHBACK creates the following error messages.

Table 68-2  DBMS_FLASHBACK Error Messages

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-08180</td>
<td>Time specified is too old</td>
</tr>
<tr>
<td>ORA-08181</td>
<td>Invalid system change number specified</td>
</tr>
<tr>
<td>ORA-08182</td>
<td>User cannot begin read-only or serializable transactions in Flashback mode</td>
</tr>
<tr>
<td>ORA-08183</td>
<td>User cannot enable Flashback within an uncommitted transaction</td>
</tr>
<tr>
<td>ORA-08184</td>
<td>User cannot enable Flashback within another Flashback session</td>
</tr>
<tr>
<td>ORA-08185</td>
<td>SYS cannot enable Flashback mode</td>
</tr>
</tbody>
</table>

68.5 DBMS_FLASHBACK Operational Notes

DBMS_FLASHBACK is automatically turned off when the session ends, either by disconnection or by starting another connection.

PL/SQL cursors opened in Flashback mode return rows as of the flashback time or SCN. Different concurrent sessions (connections) in the database can perform Flashback to different wall-clock times or SCNs. DML and DDL operations and distributed operations are not allowed while a session is running in Flashback mode. You can use PL/SQL cursors opened before disabling Flashback to perform DML.

Under Automatic Undo Management (AUM) mode, you can use retention control to control how far back in time to go for the version of the database you need. If you need to perform a Flashback over a 24-hour period, the DBA must set the undo_retention parameter to 24 hours. This way, the system retains enough undo information to regenerate the older versions of the data.
You can set the RETENTION GUARANTEE clause for the undo tablespace to ensure that unexpired undo is not discarded. UNDO_RETENTION is not in itself a guarantee because, if the system is under space pressure, unexpired undo may be overwritten with freshly generated undo. In such cases, RETENTION GUARANTEE prevents this. For more information, see the Oracle Database Administrator's Guide.

In a Flashback-enabled session, SYSDATE is not affected; it continues to provide the current time.

DBMS_FLASHBACK can be used within logon triggers to enable Flashback without changing the application code.

See Also:
Oracle Database Administrator's Guide for information on setting the minimum undo retention period.

68.6 DBMS_FLASHBACK Examples

The following example illustrates how Flashback can be used when the deletion of a senior employee triggers the deletion of all the personnel reporting to him. Using the Flashback feature, you can recover and re-insert the missing employees.

```sql
DROP TABLE employee;
DROP TABLE keep SCN;
REM -- Keep_scn is a temporary table to store scns that we are interested in
CREATE TABLE keep SCN (scn number);
SET ECHO ON
CREATE TABLE employee (employee_no number(5) PRIMARY KEY,
employee_name varchar2(20),
employee_mgr number(5)
CONSTRAINT mgr_fkey REFERENCES EMPLOYEE ON DELETE CASCADE,
salary number,
hiredate date);
REM -- Populate the company with employees
INSERT INTO employee VALUES (1, 'John Doe', null, 1000000, '5-jul-81');
INSERT INTO employee VALUES (10, 'Joe Johnson', 1, 500000, '12-aug-84');
INSERT INTO employee VALUES (20, 'Susie Tiger', 10, 250000, '13-dec-90');
INSERT INTO employee VALUES (100, 'Scott Tiger', 20, 200000, '3-feb-86');
INSERT INTO employee VALUES (200, 'Charles Smith', 100, 150000, '22-mar-88');
INSERT INTO employee VALUES (210, 'Jane Johnson', 100, 100000, '11-apr-87');
INSERT INTO employee VALUES (220, 'Nancy Doe', 100, 100000, '18-sep-93');
INSERT INTO employee VALUES (300, 'Gary Smith', 210, 75000, '4-nov-96');
INSERT INTO employee VALUES (310, 'Bob Smith', 210, 65000, '3-may-95');
COMMIT;
REM -- Show the entire org
SELECT lpad( ' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
```

Chapter 68
DBMS_FLASHBACK Examples
ORDER BY LEVEL;

REM -- Sleep for a short time (approximately 10 to 20 seconds) to avoid
REM -- querying close to table creation
EXECUTE DBMS_LOCK.SLEEP(10);

REM -- Store this snapshot for later access through Flashback
DECLARE
I NUMBER;
BEGIN
I := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
INSERT INTO keep_scn VALUES (I);
COMMIT;
END;
/

REM -- Scott decides to retire but the transaction is done incorrectly
DELETE FROM EMPLOYEE WHERE employee_name = 'Scott Tiger';
COMMIT;

REM -- notice that all of scott's employees are gone
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM EMPLOYEE
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
ORDER BY LEVEL;

REM -- Flashback to see Scott's organization
DECLARE
restore_scn number;
BEGIN
SELECT scn INTO restore_scn FROM keep_scn;
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER (restore_scn);
END;
/

REM -- Show Scott's org.
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no =
(SELECT employee_no FROM employee WHERE employee_name = 'Scott Tiger')
ORDER BY LEVEL;

REM -- Restore scott's organization.
DECLARE
scotts_emp NUMBER;
scotts_mgr NUMBER;
CURSOR c1 IS
SELECT employee_no, employee_name, employee_mgr, salary, hiredate
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no =
(SELECT employee_no FROM employee WHERE employee_name = 'Scott Tiger');
cl_rec c1 % ROWTYPE;
BEGIN
SELECT employee_no, employee_mgr INTO scotts_emp, scotts_mgr FROM employee
WHERE employee_name = 'Scott Tiger';
/* Open c1 in flashback mode */
OPEN c1;
/* Disable Flashback */
DBMS_FLASHBACK.DISABLE;
LOOP
    FETCH c1 INTO c1_rec;
    EXIT WHEN c1%NOTFOUND;
    /*
    Note that all the DML operations inside the loop are performed
    with Flashback disabled
    */
    IF (c1_rec.employee_mgr = scotts_emp) then
        INSERT INTO employee VALUES (c1_rec.employee_no,
        c1_rec.employee_name,
        scotts_mgr,
        c1_rec.salary,
        c1_rec.hiredate);
    ELSE
        IF (c1_rec.employee_no != scotts_emp) THEN
            INSERT INTO employee VALUES (c1_rec.employee_no,
            c1_rec.employee_name,
            c1_rec.employee_mgr,
            c1_rec.salary,
            c1_rec.hiredate);
        END IF;
    END IF;
END LOOP;
END;
/

REM -- Show the restored organization.
select lpad(' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
ORDER BY LEVEL;

68.7 Summary of DBMS_FLASHBACK Subprograms

This table lists the DBMS_FLASHBACK subprograms and briefly describes them.

Table 68-3 DBMS_FLASHBACK Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE Procedure</td>
<td>Disables the Flashback mode for the entire session</td>
</tr>
<tr>
<td>ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure</td>
<td>Enables Flashback for the entire session. Takes an SCN as an Oracle number and sets the session snapshot to the specified number. Inside the Flashback mode, all queries return data consistent as of the specified wall-clock time or SCN</td>
</tr>
<tr>
<td>ENABLE_AT_TIME Procedure</td>
<td>Enables Flashback for the entire session. The snapshot time is set to the SCN that most closely matches the time specified in query_time</td>
</tr>
<tr>
<td>GET_SYSTEM_CHANGE_NUMBER Function</td>
<td>Returns the current SCN as an Oracle number. You can use the SCN to store specific snapshots</td>
</tr>
<tr>
<td>TRANSACTION_BACKOUT Procedures</td>
<td>Provides the mechanism to back out a transaction</td>
</tr>
</tbody>
</table>
68.7.1 DISABLE Procedure

This procedure disables the Flashback mode for the entire session.

Syntax

DBMS_FLASHBACK.DISABLE;

Examples

The following example queries the salary of an employee, Joe, on August 30, 2000:

EXECUTE dbms_flashback.enable_at_time('30-AUG-2000');
SELECT salary FROM emp where name = 'Joe'
EXECUTE dbms_flashback.disable;

68.7.2 ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure

This procedure takes an SCN as an input parameter and sets the session snapshot to the specified number.

In the Flashback mode, all queries return data consistent as of the specified wall-clock time or SCN. It enables Flashback for the entire session.

Syntax

DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER (query_scn IN NUMBER);

Parameters

Table 68-4   ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query_scn</td>
<td>The system change number (SCN), a version number for the database that is incremented on every transaction commit.</td>
</tr>
</tbody>
</table>

68.7.3 ENABLE_AT_TIME Procedure

This procedure enables Flashback for the entire session.

The snapshot time is set to the SCN that most closely matches the time specified in query_time. It enables Flashback for the entire session.

Syntax

DBMS_FLASHBACK.ENABLE_AT_TIME (query_time IN TIMESTAMP);
Parameters

Table 68-5 ENABLE_AT_TIME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| query_time | This is an input parameter of type TIMESTAMP. A time stamp can be specified in the following ways:  
  - Using the TIMESTAMP constructor  
    
    `EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME(TIMESTAMP '2001-01-09 12:31:00').`  
    
    Use the Globalization Support (NLS) format and supply a string. The format depends on the Globalization Support settings.  
  - Using the TO_TIMESTAMP function:  
    
    `EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME(TO_TIMESTAMP('12-02-2001 14:35:00', 'DD-MM-YYYY HH24:MI:SS'))`  
    
    You provide the format you want to use. This example shows the TO_TIMESTAMP function for February 12, 2001, 2:35 PM.  
  - If the time is omitted from query time, it defaults to the beginning of the day, that is, 12:00 A.M.  
  - Note that if the query time contains a time zone, the time zone information is truncated. |

68.7.4 GET_SYSTEM_CHANGE_NUMBER Function

This function returns the current SCN as an Oracle number datatype. You can obtain the current change number and store it for later use. This helps you retain specific snapshots.

Syntax

```
DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER
RETURN NUMBER;
```

68.7.5 TRANSACTION_BACKOUT Procedures

This procedure provides a mechanism to back out a set of transactions. The user can call these procedures with either transaction names or transaction identifiers (XIDS).

The procedure analyzes the transactional dependencies, perform DMLs and generates an extensive report on the operation performed by the subprogram. This procedure does not commit the DMLs performed as part of transaction back out. However it holds all the required locks on rows and tables in the right form, so that no other dependencies can enter the system. To make the changes permanent you must explicitly commit the transaction.

A report is generated in the system tables `DBA_FLASHBACK_TRANSACTION_STATE` and `DBA_FLASHBACK_TRANSACTION_REPORT`.

Syntax

```
DBMS_FLASHBACK.TRANSACTION_BACKOUT
numtxns NUMBER,
```
Parameters

Table 68-6 TRANSACTION_BACKOUT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numtxns</td>
<td>Number of transactions passed as input</td>
</tr>
<tr>
<td>xids</td>
<td>List of transaction IDs in the form of an array</td>
</tr>
<tr>
<td>txnnames</td>
<td>List of transaction names in the form of an array</td>
</tr>
<tr>
<td>options</td>
<td>Back out dependent transactions:</td>
</tr>
<tr>
<td></td>
<td>• NOCASCADE - No dependency is expected. If a dependency is found, this raises an error, with the first dependent transaction provided in the report.</td>
</tr>
<tr>
<td></td>
<td>• NOCASCADE_FORCE - The user forcibly backs out the given transactions without considering the dependent transactions. The RDBMS executes the UNDO SQL for the given transactions in reverse order of their commit times. If no constraints break, and the result is satisfactory, the user can either COMMIT the changes or else ROLL BACK.</td>
</tr>
<tr>
<td></td>
<td>• NONCONFLICT_ONLY - This option lets the user back out the changes to the nonconflicting rows of the given transactions. Note that a transaction dependency can happen due to a row conflict through either WAW or primary/unique key constraints. If the user chooses to back out only the nonconflicting rows, this does not cause any problem with database consistency, although transaction atomicity is lost. As this is a recovery operation, the user can correct the data.</td>
</tr>
<tr>
<td></td>
<td>• CASCADE - This completely removes the given transactions including their dependents in a post order fashion (reverse order of commit times).</td>
</tr>
<tr>
<td>timehint</td>
<td>Time hint on the start of the transaction</td>
</tr>
<tr>
<td>scnHint</td>
<td>SCN hint on the start of the transaction</td>
</tr>
</tbody>
</table>
Usage Notes

Note:

For information about restrictions in using TRANSACTION_BACKOUT, see "Using Flashback Transaction" in the Oracle Database Development Guide.

- If transaction name is used, a time hint must be provided. The time hint should be a time before the start of all the given transactions to back out.
- If the SCN hint is provided, it must be before the start of the earliest transaction in the specified input set, or this raises an error and terminates. If it is not provided and the transaction has committed within undo retention, the database system is able to determine the start time.
The **DBMS_FLASHBACK_ARCHIVE** package contains procedures for performing various flashback tasks.

These include:

- Disassociation and reassociation of a Flashback Archive enabled table from/with its underlying Flashback Data Archive (FDA)
- Tamper-proofing the tables of an application
- Importing of user history

⚠️ **Caution:**

Importing user-generated history can lead to inaccurate, or unreliable results. This procedure should only be used after consulting with Oracle Support.

- Enabling and disabling of session-level support for valid-time

**See Also:**

*Oracle Database Development Guide* for more information about "Using Flashback Data Archive (Oracle Temporal)"

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Constants**
- **Summary of DBMS_FLASHBACK_ARCHIVE Subprograms**
69.1 DBMS_FLASHBACK_ARCHIVE Overview

Flashback Data Archive (FDA) provides strict protection on the internal history tables that it creates and maintains for users.

⚠️ Caution:

The DBMS_FLASHBACK_ARCHIVE package does not support migration between databases. It allows you to move FDA history within the same database only, and not outside it. This package does not allow you to move FDA table data. It allows you to import customer history only and not FDA history.

The read-only semantics prohibit users, including a DBA, from doing updates, deletes, and inserts on the Flashback Data Archive internal history tables. Users are also prevented from issuing any DDL statements on these tables. This strict security enforcement helps meet the requirements of applications in regulatory / compliance environments. Flashback Data Archive supports most common DDL statements, including those that alter the table definition or incur data movement. However, some DDL statements are not supported on Flashback Archive enabled tables. Since most application schemas are modified during application software upgrades, the ability to perform DDL operations on tracked tables is critical.

To support schema evolution during application upgrades and other table maintenance tasks that require use of DDL statements not supported by Flashback Data Archive, the DBMS_FLASHBACK_ARCHIVE package provides a set of simple-to-use PL/SQL procedures:

- To disassociate a Flashback Archive enabled base table from the underlying FDA
- To reassociate a temporarily disassociated base table with its underlying FDA

After a user has disassociated the base table from its FDA, it’s possible to issue any DDL statements on the base table or the history tables in the FDA. Having finished with the schema changes, the user can then reassociate the base table with its FDA so that Flashback Data Archive protection is in operation and automatic tracking and archiving is resumed.

69.2 DBMS_FLASHBACK_ARCHIVE Security Model

Users need the FLASHBACK_ARCHIVE_ADMINISTER privilege to import user-generated history, to set context level, and to tamper-proof tables. After a table is disassociated, users can perform DDL and DML statements on the table if they have the necessary privileges. Enabling and disabling session-level Valid Time Temporal flashback needs no additional privileges.

69.3 DBMS_FLASHBACK_ARCHIVE Constants

The DBMS_FLASHBACK_ARCHIVE package uses the constants shown in the following table.
Table 69-1  DBMS_FLASHBACK_ARCHIVE Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODROP</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Do not drop temporary history table</td>
</tr>
<tr>
<td>NOCOMMIT</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Do not commit transaction</td>
</tr>
<tr>
<td>NODELETE</td>
<td>BINARY_INTEGER</td>
<td>4</td>
<td>Do not delete data in history table</td>
</tr>
</tbody>
</table>

69.4 Summary of DBMS_FLASHBACK_ARCHIVE Subprograms

This table lists the DBMS_FLASHBACK_ARCHIVE subprograms and briefly describes them.

Table 69-2  DBMS_FLASHBACK_ARCHIVE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TABLE_TO_APPLICATION</td>
<td>Takes an application name and adds a table to the application as a security table</td>
</tr>
<tr>
<td>CREATE_TEMP_HISTORY_TABLE</td>
<td>Creates a table called TEMP_HISTORY with the correct definition in schema</td>
</tr>
<tr>
<td>DISABLE_APPLICATION</td>
<td>Takes an application name and marks a table in it as a security table</td>
</tr>
<tr>
<td>DISABLE_ASOF_VALID_TIME</td>
<td>Disables session level valid-time flashback</td>
</tr>
<tr>
<td>DISASSOCIATE_FBA</td>
<td>Disassociates the given table from the flashback data archive</td>
</tr>
<tr>
<td>DROP_APPLICATION</td>
<td>Takes an application name and removes it from the list of applications</td>
</tr>
<tr>
<td>ENABLE_APPLICATION</td>
<td>Takes an application name and enables Flashback Data Archive on all the security tables for this application</td>
</tr>
<tr>
<td>ENABLE_AT_VALID_TIME</td>
<td>Enables session level valid time flashback</td>
</tr>
<tr>
<td>EXTEND_MAPPINGS</td>
<td>Extends time mappings to times in the past</td>
</tr>
<tr>
<td>GET_SYS_CONTEXT</td>
<td>Gets the context previously selected by the SET_CONTEXT_LEVEL Procedure</td>
</tr>
<tr>
<td>IMPORT_HISTORY</td>
<td>Imports history from a table called TEMP_HISTORY in the given schema.</td>
</tr>
<tr>
<td>LOCK_DOWN_APPLICATION</td>
<td>Takes an application name and makes all the security tables read-only. The group called SYSTEM cannot be locked</td>
</tr>
<tr>
<td>PURGE_CONTEXT</td>
<td>Purges the context to be saved selected by the SET_CONTEXT_LEVEL Procedure</td>
</tr>
<tr>
<td>REASSOCIATE_FBA</td>
<td>Reassociates the given table with the flashback data archive</td>
</tr>
<tr>
<td>REGISTER_APPLICATION</td>
<td>Takes an application name and optionally a Flashback Data Archive, and registers an application for database hardening</td>
</tr>
<tr>
<td>REMOVE_TABLE_FROM_APPLICATION</td>
<td>Takes an application name and marks a table in it as no longer being a security table</td>
</tr>
<tr>
<td>SET_CONTEXT_LEVEL</td>
<td>Defines how much of the user context is to be saved</td>
</tr>
</tbody>
</table>
69.4.1 ADD_TABLE_TO_APPLICATION Procedure

This procedure takes an application name and adds a table to the application as a security table. If the application is enabled for Flashback Data Archive, then this table will also be enabled for Flashback Data Archive.

**Syntax**

```
DBMS_FLASHBACK_ARCHIVE.ADD_TABLE_TO_APPLICATION (  
    application_name IN VARCHAR2,
    table_name     IN VARCHAR2,
    schema_name    IN VARCHAR2 := NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_name</td>
<td>Name of the application for which a table has been added as a security table</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table to add as a security table for the given application</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of the schema containing the desired table. If no schema name is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

69.4.2 CREATE_TEMP_HISTORY_TABLE Procedure

This procedure creates a table called TEMP_HISTORY with the correct definition in schema.

**Syntax**

```
DBMS_FLASHBACK_ARCHIVE.CREATE_TEMP_HISTORY_TABLE (  
    owner_name1   IN VARCHAR2,
    table_name1   IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name1</td>
<td>Schema of the Flashback Data Archive-enabled table</td>
</tr>
<tr>
<td>table_name1</td>
<td>Name of the Flashback Data Archive-enabled table</td>
</tr>
</tbody>
</table>
69.4.3 DISABLE_APPLICATION Procedure

This procedure takes an application name and disables Flashback Data Archive on all of its security tables.

Syntax

    DBMS_FLASHBACK_ARCHIVE.DISABLE_APPLICATION (
            application_name    IN VARCHAR2);

Parameters

Table 69-5  DISABLE_APPLICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_name</td>
<td>Name of the application whose security tables will be disabled for Flashback Data Archive</td>
</tr>
</tbody>
</table>

69.4.4 DISABLE_ASOF_VALID_TIME Procedure

This procedure disables session level valid-time flashback.

Syntax

    DBMS_FLASHBACK_ARCHIVE.DISABLE_ASOF_VALID_TIME;

69.4.5 DISASSOCIATE_FBA Procedure

This procedure disassociates the given table from the flashback data archive.

Syntax

    DBMS_FLASHBACK_ARCHIVE.DISASSOCIATE_FBA (
            owner_name     IN VARCHAR2,
            table_name     IN VARCHAR2);

Parameters

Table 69-6  DISASSOCIATE_FBA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>Schema of the Flashback Data Archive enabled base table</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the Flashback Data Archive enabled base table</td>
</tr>
</tbody>
</table>

Exceptions

Table 69-7  DISASSOCIATE_FBA Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-55602</td>
<td>User table is not enabled for Flashback Data Archive</td>
</tr>
</tbody>
</table>
### 69.4.6 DROP_APPLICATION Procedure

This procedure takes an application name and removes it from the list of applications. As part of this procedure, Flashback Data Archive will be disabled on all security-enabled tables and all history data will be lost. The group called `SYSTEM` cannot be dropped.

**Syntax**

```
DBMS_FLASHBACK_ARCHIVE.DROP_APPLICATION (application_name IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_name</td>
<td>Name of the application for which a table has been added as a security table</td>
</tr>
</tbody>
</table>

### 69.4.7 ENABLE_APPLICATION Procedure

This procedure takes an application name and enables Flashback Data Archive on all the security tables for this application. Once an application is enabled, every change to an FDA enabled table will be tracked.

**Syntax**

```
DBMS_FLASHBACK_ARCHIVE.ENABLE_APPLICATION (application_name IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_name</td>
<td>Name of the application for which to enable Flashback Data Archive on all its security tables</td>
</tr>
</tbody>
</table>
69.4.8 ENABLE_AT_VALID_TIME Procedure

This procedure enables session level valid time flashback.

Syntax

```sql
DBMS_FLASHBACK_ARCHIVE.ENABLE_AT_VALID_TIME (level IN VARCHAR2, query_time IN TIMESTAMP DEFAULT SYSTIMESTAMP);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td>· All - Sets the visibility of temporal data to the full table, which is the default temporal table visibility</td>
</tr>
<tr>
<td></td>
<td>· CURRENT - Sets the visibility of temporal data to currently valid data within the valid time period at the session level</td>
</tr>
<tr>
<td></td>
<td>· ASOF - Sets the visibility of temporal data to data valid as of the given time as defined by the timestamp</td>
</tr>
<tr>
<td>query_time</td>
<td>Used only if level is ASOF. Data which is valid at this query_time will only be shown.</td>
</tr>
</tbody>
</table>

69.4.9 EXTEND_MAPPINGS Procedure

This procedure extends time mappings to times in the past.

Syntax

```sql
DBMS_FLASHBACK_ARCHIVE.EXTEND_MAPPINGS;
```

69.4.10 GET_SYS_CONTEXT Function

This function gets the context previously selected by the SET_CONTEXT_LEVEL Procedure.

Syntax

```sql
DBMS_FLASHBACK_ARCHIVE.GET_SYS_CONTEXT (xid IN RAW, namespace IN VARCHAR2, parameter IN VARCHAR2) RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>Transaction identifier is an opaque handle to a transaction obtained from the versions query</td>
</tr>
</tbody>
</table>
Table 69-11  (Cont.) GET_SYS_CONTEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>Namespace</td>
</tr>
<tr>
<td>parameter</td>
<td>If undefined, the subprogram returns NULL</td>
</tr>
</tbody>
</table>

Related Topics

- SET_CONTEXT_LEVEL Procedure
  This procedure defines how much of the user context is to be saved.

### 69.4.11 IMPORT_HISTORY Procedure

This procedure is called after invoking the CREATE_TEMP_HISTORY_TABLE procedure, and after the TEMP_HISTORY table is populated with user-generated history data.

![Caution:]

Importing user-generated history can lead to inaccurate, or unreliable results. This procedure should only be used after consulting with Oracle Support.

> The DBMS_FLASHBACK_ARCHIVE package does not support migration between databases. It allows you to move FDA history within the same database only, and not outside it. This package does not allow you to move FDA table data. It allows you to import customer history only and not FDA history.

#### Syntax

```sql
DBMS_FLASHBACK_ARCHIVE.IMPORT_HISTORY (
  owner_name1         IN   VARCHAR2,
  table_name1         IN   VARCHAR2,
  temp_history_name   IN   VARCHAR2 DEFAULT 'TEMP_HISTORY',
  options             IN   BINARY_INTEGER DEFAULT 0);
```

#### Parameters

Table 69-12  IMPORT_HISTORY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name1</td>
<td>Schema of the Flashback Data Archive-enabled table</td>
</tr>
<tr>
<td>table_name1</td>
<td>Name of the Flashback Data Archive-enabled table</td>
</tr>
<tr>
<td>temp_history_name</td>
<td>Optional temporary history table from which we import history data</td>
</tr>
<tr>
<td>options</td>
<td>One (or a combination) of constants (NODROP, NOCOMMIT, and NODELETE) to specify if we want to drop, commit changes of, or truncate the temporary history table</td>
</tr>
</tbody>
</table>
Usage Notes

The database function `TIMESTAMP_TO_SCN` can be used to convert times to SCN when populating the temporary history table.

Related Topics

- **CREATE_TEMP_HISTORY_TABLE Procedure**
  This procedure creates a table called `TEMP_HISTORY` with the correct definition in schema.

### 69.4.12 LOCK_DOWN_APPLICATION Procedure

This procedure takes an application name and makes all the security tables read-only. The group called `SYSTEM` cannot be locked.

**Syntax**

```
DBMS_FLASHBACK_ARCHIVE.LOCK_DOWN_APPLICATION (
    application_name           IN   VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_name</td>
<td>Name of the application for which a table has been added as a security table</td>
</tr>
</tbody>
</table>

### 69.4.13 PURGE_CONTEXT Procedure

This procedure purges the context to be saved selected by the `SET_CONTEXT_LEVEL` Procedure.

**Syntax**

```
DBMS_FLASHBACK_ARCHIVE.PURGE_CONTEXT;
```

**Related Topics**

- **SET_CONTEXT_LEVEL Procedure**
  This procedure defines how much of the user context is to be saved.

### 69.4.14 REASSOCIATE_FBA Procedure

This procedure reassociates the given table with the flashback data archive.

**Syntax**

```
DBMS_FLASHBACK_ARCHIVE.REASSOCIATE_FBA (  
    owner_name      VARCHAR2,  
    table_name      VARCHAR2);
```
Parameters

Table 69-14  REASSOCIATE_FBA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>Schema of the Flashback Data Archive enabled base table</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the Flashback Data Archive enabled base table</td>
</tr>
</tbody>
</table>

Exceptions

Table 69-15  REASSOCIATE_FBA Procedure Exceptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-55602</td>
<td>User table is not enabled for Flashback Data Archive</td>
</tr>
<tr>
<td>ORA-55636</td>
<td>Table definition validation failed</td>
</tr>
</tbody>
</table>

Usage Notes

- The procedure will signal an error if the base table and the history table do not have identical data definitions. For example, when columns are added or table is split, the resulting base table and history table need to have the same schema.
- The FDA internal history table schema has some row versions metadata columns. The procedure will signal an error if any metadata column is dropped by users.

69.4.15 REGISTER_APPLICATION Procedure

This procedure takes an application name and optionally a Flashback Data Archive, and registers an application for database hardening.

When database hardening is enabled, then all the security tables for that application are enabled for Flashback Data Archive using the given Flashback Data Archive. If no Flashback Data Archive is specified, the default Flashback Data Archive is used.

See Also:

Using Flashback Data Archive in *Oracle Database Development Guide* regarding database hardening

Syntax

```sql
DBMS_FLASHBACK_ARCHIVE.REGISTER_APPLICATION (  
    application_name        IN   VARCHAR2,  
    flashback_archive_name IN VARCHAR2 := NULL);
```
### Parameters

#### Table 69-16 REGISTER_APPLICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_name</td>
<td>Name of the application which is being registered. The application SYSTEM is already registered when the package is created and is populated with list of tables needed for database hardening.</td>
</tr>
<tr>
<td>flashback_archive_name</td>
<td>Name of the Flashback Data Archive in which the historical data for the security tables for given application is stored. If no Flashback Data Archive is specified, the default Flashback Data Archive is used.</td>
</tr>
</tbody>
</table>

### 69.4.16 REMOVE_TABLE_FROM_APPLICATION Procedure

This procedure takes an application name and marks a table in it as no longer being a security table.

If the application is already enabled for Flashback Data Archive, Flashback Data Archive will be disabled for this table.

#### Syntax

```oracle
DBMS_FLASHBACK_ARCHIVE.REMOVE_TABLE_FROM_APPLICATION (  
  application_name           IN   VARCHAR2,  
  table_name                 IN   VARCHAR2,  
  schema_name                IN   VARCHAR2 := NULL);  
```

#### Parameters

#### Table 69-17 REMOVE_TABLE_FROM_APPLICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_name</td>
<td>Name of the application for which a table is being removed from the list of security tables</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table to mark as being no longer a security table for the given application</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of the schema containing the desired table. If no schema name is specified, the current schema is used.</td>
</tr>
</tbody>
</table>

### 69.4.17 SET_CONTEXT_LEVEL Procedure

This procedure defines how much of the user context is to be saved.

#### Syntax

```oracle
DBMS_FLASHBACK_ARCHIVE.SET_CONTEXT_LEVEL (  
  level          VARCHAR2);  
```
### Parameters

Table 69-18 SET_CONTEXT_LEVEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>Depending on how much of the user context needs to be saved:</td>
</tr>
<tr>
<td></td>
<td>• ALL - the entire SYS_CONTEXT</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL - the user ID, global user ID and the hostname</td>
</tr>
<tr>
<td></td>
<td>• NONE - nothing</td>
</tr>
</tbody>
</table>
The DBMS_FREQUENT_ITEMSET package enables frequent itemset counting. The two functions are identical except in the input cursor format difference.

This chapter contains the following topics:

- Summary of DBMS_FREQUENT_ITEMSET Subprograms

### 70.1 Summary of DBMS_FREQUENT_ITEMSET Subprograms

The DBMS_FREQUENT_ITEMSET package includes the `FI_HORIZONTAL` function and `FI_TRANSACTIONAL` function subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FI_HORIZONTAL</code> Function</td>
<td>Counts all frequent itemsets given a cursor for input data which is in 'HORIZONTAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded</td>
</tr>
<tr>
<td><code>FI_TRANSACTIONAL</code> Function</td>
<td>Counts all frequent itemsets given a cursor for input data which is in 'TRANSACTIONAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded</td>
</tr>
</tbody>
</table>

#### 70.1.1 FI_HORIZONTAL Function

The purpose of this table function is to count all frequent itemsets given a cursor for input data which is in 'HORIZONTAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded. The result will be a table of rows in form of itemset, support, length, total transactions counted.

In 'HORIZONTAL' row format, each row contains all of the item ids for a single transaction. Since all of the items come together, no transaction id is necessary.

The benefit of this table function is that if an application already has data in horizontal format, the database can skip the step of transforming rows that are in transactional format into horizontal format.

**Syntax**

```sql
DBMS_FREQUENT_ITEMSET.FI_HORIZONTAL(
    tranx_cursor IN SYSREFCURSOR,
    support_threshold IN NUMBER,
    itemset_length_min IN NUMBER,
    itemset_length_max IN NUMBER,
    including_items IN SYS_REFCURSOR DEFAULT NULL,
    excluding_items IN SYS_REFCURSOR DEFAULT NULL)
```
RETURN TABLE OF ROW (  
  itemset [Nested Table of Item Type DERIVED FROM tranx_cursor],  
  support NUMBER,  
  length NUMBER,  
  total_tranx NUMBER);  

Parameters  

Table 70-2  FI_HORIZONTAL Function Parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tranx_cursor</td>
<td>The cursor parameter that the user will supply when calling the function. There is no limits on the number of returning columns. Each column of cursor represents an item. All columns of the cursor must be of the same datatype. The item id must be number or character type (for example, VARCHAR2(n)).</td>
</tr>
<tr>
<td>support_threshold</td>
<td>A fraction number of total transaction count. An itemset is termed &quot;frequent&quot; if [the number of transactions it occurs in] divided by [the total number of transactions] exceed the fraction. The parameter must be a NUMBER.</td>
</tr>
<tr>
<td>itemset_length_min</td>
<td>The minimum length for interested frequent itemset. The parameter must be a NUMBER between 1 and 20, inclusive.</td>
</tr>
<tr>
<td>itemset_length_max</td>
<td>The maximum length for interested frequent itemset. This parameter must be a NUMBER between 1 and 20, inclusive, and must not be less than itemset_length_min.</td>
</tr>
<tr>
<td>including_items</td>
<td>A cursor from which a list of items can be fetched. At least one item from the list must appear in frequent itemsets that are returned. The default is NULL.</td>
</tr>
<tr>
<td>excluding_items</td>
<td>A cursor from which a list of items can be fetched. No item from the list can appear in frequent itemsets that are returned. The default is NULL.</td>
</tr>
</tbody>
</table>

Return Values  

Table 70-3  FI_HORIZONTAL Return Values  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>support</td>
<td>The number of transactions in which a frequent itemset occurs. This will be returned as a NUMBER.</td>
</tr>
<tr>
<td>itemset</td>
<td>A collection of items which is computed as frequent itemset. This will be returned as a nested table of item type which is the item column type of the input cursor.</td>
</tr>
<tr>
<td>length</td>
<td>Number of items in a frequent itemset. This will be returned as a NUMBER.</td>
</tr>
<tr>
<td>total_tranx</td>
<td>The total transaction count. This will be returned as a NUMBER.</td>
</tr>
</tbody>
</table>

Example  

Suppose you have a table horiz_table_in.
and the data in horiz_table_in looks as follows:

('apple', 'banana', NULL, NULL, NULL)
('apple', 'milk', 'banana', NULL, NULL)
('orange', NULL, NULL, NULL, NULL)

Suppose you want to find out what combinations of items is frequent with a given support threshold of 30%, requiring itemset containing at least one of ('apple','banana','orange'), but excluding any of ('milk') in any itemset. You use the following query:

```sql
CREATE TYPE fi_varchar_nt AS TABLE OF VARCHAR2(30);
SELECT CAST(itemset as FI_VARCHAR_NT)itemset, support, length, total_tranx
FROM table(DBMS_FREQUENT_ITEMSET.FI_HORIZONTAL(
    CURSOR(SELECT iid1, iid2, iid3, iid4, iid5
        FROM horiz_table_in),
    0.3,
    2,
    5,
    CURSOR(SELECT * FROM table(FI_VARCHAR_NT('apple','banana','orange'))),
    CURSOR(SELECT * FROM table(FI_VARCHAR_NT('milk')))));
```

### 70.1.2 FI_TRANSACTIONAL Function

This procedure counts all frequent itemsets given a cursor for input data which is in 'TRANSACTIONAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded. The result will be a table of rows in form of itemset, support, length, total number of transactions.

In 'TRANSACTIONAL' row format, each transaction is spread across multiple rows. All the rows of a given transaction have the same transaction id, and each row has a different item id. Combining all of the item ids which share a given transaction id results in a single transaction.

**Syntax**

```sql
DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL (
    tranx_cursor         IN    SYSREFCURSOR,
    support_threshold    IN    NUMBER,
    itemset_length_min   IN    NUMBER,
    itemset_length_max   IN    NUMBER,
    including_items      IN    SYS_REFCURSOR DEFAULT NULL,
    excluding_items      IN    SYS_REFCURSOR DEFAULT NULL)
RETURN TABLE OF ROW (
    itemset [Nested Table of Item Type DERIVED FROM tranx_cursor],
    support    NUMBER,
    length     NUMBER,
    total_tranx    NUMBER);
```
Parameters

Table 70-4  F1_TRANSACTIONAL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tranx_cursor</td>
<td>The cursor parameter that the user will supply when calling the function. It should return two columns in its returning row, the first column being the transaction id, the second column being the item id. The item id must be number or character type (for example, VARCHAR2(n)).</td>
</tr>
<tr>
<td>support_threshold</td>
<td>A fraction number of total transaction count. An itemset is termed &quot;frequent&quot; if [the number of transactions it occurs in] divided by [the total number of transactions] exceed the fraction. The parameter must be a NUMBER.</td>
</tr>
<tr>
<td>itemset_length_min</td>
<td>The minimum length for interested frequent itemset. The parameter must be a NUMBER between 1 and 20, inclusive.</td>
</tr>
<tr>
<td>itemset_length_max</td>
<td>The maximum length for interested frequent itemset. This parameter must be a NUMBER between 1 and 20, inclusive, and must not be less than itemset_length_min.</td>
</tr>
<tr>
<td>including_items</td>
<td>A cursor from which a list of items can be fetched. At least one item from the list must appear in frequent itemsets that will be returned. The default is NULL.</td>
</tr>
<tr>
<td>excluding_items</td>
<td>A cursor from which a list of items can be fetched. No item from the list can appear in frequent itemsets that will returned. The default is NULL.</td>
</tr>
</tbody>
</table>

Return Values

Table 70-5  F1_TRANSACTIONAL Return Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>support</td>
<td>The number of transactions in which a frequent itemset occurs. This will be returned as a NUMBER.</td>
</tr>
<tr>
<td>itemset</td>
<td>A collection of items which is computed as frequent itemset. This will be returned as a nested table of item type which is the item column type of the input cursor.</td>
</tr>
<tr>
<td>length</td>
<td>Number of items in a frequent itemset. This will be returned as a NUMBER.</td>
</tr>
<tr>
<td>total_tranx</td>
<td>The total transaction count. This will be returned as a NUMBER, and will be the same for all returned rows, similar to a reporting aggregate.</td>
</tr>
</tbody>
</table>

Usage Notes

Applications must predefine a nested table type of the input item type and cast the output itemset into this predefined nested table type before further processing, such as loading into a table.

Examples

Suppose that the input table tranx_table_in looks as follows:
and the user is trying to find itemsets that satisfy a support-threshold of 60% and have the itemset-length greater than 1 (namely, (apple, banana)).

The output of this function would contain the following output row:

itemset=('apple','banana'), support=2, length=2, total_tranx=3

You need to create a nested table of item type before you submit a query to perform the frequent itemset counting. In this example, since item is of VARCHAR2(30), you must create a nested table of VARCHAR2(30):

```
CREATE TYPE fi_varchar_nt AS TABLE OF VARCHAR2(30);
SELECT CAST(itemset as FI_VARCHAR_NT) itemset, support, length, total_tranx
FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
    cursor(SELECT tid, iid FROM tranx_table_in),
    0.6,
    2,
    5,
    NULL,
    NULL));
```

Here is another example to illustrate how to include certain items and exclude certain items in the counting.

```
SELECT CAST(itemset as FI_VARCHAR_NT) itemset, support, length, total_tranx
FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
    cursor(SELECT tid, iid FROM tranx_table_in),
    0.6,
    2,
    5,
    CURSOR(SELECT * FROM table(FI_VARCHAR_NT('apple','banana','orange'))),
    CURSOR(SELECT * FROM table(FI_VARCHAR_NT('milk')))));
```

Using the including/excluding items parameter, you are able to further optimize the execution by ignoring itemsets that are not expected by application.

You can also use transactional output through collection unnesting:

```
SELECT
    bt.setid, nt.*
FROM
    (SELECT cast(Itemset as FI_VARCHAR_NT) itemset, rownum setid
     FROM table(
        DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
            cursor(SELECT tid, iid FROM tranx_table_in),
            0.6, 2, 5,
            NULL, NULL))) bt,
    table(bt.itemset) nt;
```

If you want to use an insert statement to load frequent itemsets into a nested table, it is better to use the NESTED_TABLE_FAST_INSERT hint for performance:
CREATE TABLE fq_nt (coll VARCHAR2(NT)) NESTED TABLE coll STORE AS coll_nest;

INSERT /*+ NESTED_TABLE_FAST_INSERT */ INTO fq_nt
SE
collas VARCHAR2(NT))
FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
cursor(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
NULL, NULL));

Note that if you want to use the package inside a PL/SQL cursor, you must cast the
return type of the table function:

CREATE TYPE fi_res AS OBJECT (
itemset VARCHAR2(NT),
support NUMBER,
length NUMBER,
total_tranx NUMBER
);
/
CREATE TYPE fi_coll AS TABLE OF fi_res;
/

DECLARE
  cursor freqC is
    SELECT Itemset
    FROM table(
      CAST(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
        cursor(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
        NULL, NULL) AS fi_coll));
coll_nt VARCHAR2(NT);
num_rows int;
num_itms int;
BEGIN
  num_rows := 0;
  num_itms := 0;
  OPEN freqC;
  LOOP
    FETCH freqC INTO coll_nt;
    EXIT WHEN freqC%NOTFOUND;
    num_rows := num_rows + 1;
    num_itms := num_itms + coll_nt.count;
  END LOOP;
  CLOSE freqC;
  DBMS_OUTPUT.PUT_LINE('Totally ' || num_rows || ' rows ' || num_itms || ' items were produced.');
END;
/
71

DBMS_FS

The DBMS_FS package for performing operations on an Oracle file system (make, mount, unmount and destroy operations) in an Oracle database.

This chapter contains the following topics:

- DBMS_FS Overview
- DBMS_FS Security Model
- Summary of DBMS_FS Subprograms

71.1 DBMS_FS Overview

The DBMS_FS package contains Oracle file system (OFS) procedures that you can use to create, mount, unmount, and destroy an Oracle file system.

The DBMS_FS package enables applications to access database objects from a universal client such as an NFS server. This feature interfaces with Oracle SecureFiles to provide the file system access.

See Also:
Oracle Database SecureFiles and Large Objects Developer's Guide for a detailed description of managing an NFS server in Oracle Database

71.2 DBMS_FS Security Model

You must have the SYSDBA administrative privilege to use the DBMS_FS package.

The operations that you perform using the DBMS_FS package are equivalent to the file system operations that are performed in an operating system by the root user. Access to the individual file system that is created and mounted by this package is enforced using Access Control Lists (ACLs) and the permissions on the mounted directories to the operating system user.

71.3 Summary of DBMS_FS Subprograms

This table lists the DBMS_FS subprograms and briefly describes them.
### Table 71-1 DBMS_FS Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESTROY_ORACLE_FS Procedure</td>
<td>Destroys an Oracle file system, using the <code>fstype</code> and of name <code>fsname</code></td>
</tr>
<tr>
<td>MAKE_ORACLE_FS Procedure</td>
<td>Creates a file system of type specified by <code>fstype</code> and of name <code>fsname</code></td>
</tr>
<tr>
<td>MOUNT_ORACLE_FS Procedure</td>
<td>Mounts an Oracle file system on the specified mount point</td>
</tr>
<tr>
<td>UNMOUNT_ORACLE_FS Procedure</td>
<td>Unmounts an Oracle file system on the specified mount point</td>
</tr>
</tbody>
</table>

#### 71.3.1 DESTROY_ORACLE_FS Procedure

This procedure destroys an Oracle file system and then frees the resources that were associated with it.

**Syntax**

```sql
DBMS_FS.DESTROY_ORACLE_FS (    
    fstype IN VARCHAR2,  
    fsname IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fstype</td>
<td>File system type. Valid options are:</td>
</tr>
<tr>
<td></td>
<td>• DBFS (database file system)</td>
</tr>
<tr>
<td></td>
<td>• OFS (Oracle file system)</td>
</tr>
<tr>
<td>fsname</td>
<td>Name of the file system</td>
</tr>
</tbody>
</table>

**Usage Notes**

- You can find information about the currently mounted file systems by querying the `V$OFSMOUNT` dynamic view.
- For more information about the file system types, see the `fstype` description in `MAKE_ORACLE_FS Procedure`.
- Before you run the `DBMS_FS.DESTROY_ORACLE_FS` procedure, you must unmount the file system by using the `DBMS_FS.UNMOUNT_ORACLE_FS` procedure.
- After you run `DBMS_FS.DESTROY_ORACLE_FS`, Oracle Database destroys the file system and frees the associated resources.

**Example**

This example shows how to destroy an OFS file system:

```sql
BEGIN  
DBMS_FS.DESTROY_ORACLE_FS (    
    
```
71.3.2 MAKE_ORACLE_FS Procedure

This procedure creates a new file system of type ofs, on top of an existing Oracle tablespace or other database object.

Syntax

```
DBMS_FS.MAKE_ORACLE_FS (
    fstype       IN VARCHAR2,
    fsname       IN VARCHAR2,
    fsoptions    IN VARCHAR2);
```

Parameters

Table 71-3  DBMS_FS Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fstype</td>
<td>File system type. Enter ofs to create an Oracle file system.</td>
</tr>
<tr>
<td>fsname</td>
<td>Name of the file system. Enter a string no longer than 256 characters, using alpha numeric characters.</td>
</tr>
<tr>
<td>fsoptions</td>
<td>Specify an existing tablespace to use for the Oracle file system, using the following format: &quot;tablespace=tablespace_name&quot;</td>
</tr>
</tbody>
</table>

Usage Notes

- If you want to create a database file system (DBFS), then you must run the dbfs_create_filesystem.sql script, which in turn calls the dbfs_create_filesystem_advanced.sql script. By default, this script is in the $ORACLE_HOME/rdbms/admin directory. When you run this script, provide the name of an existing tablespace and a name for the file system that will be stored in the database. The size of the file system will be the same as the table size. For example, to create a file system in the dbfs_ts tablespace, in the file system dbfs_tab:

  ```
  @/$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem.sql dbfs_ts dbfs_tab
  ```

  After you run this script, you can use the other procedures in the DBMS_FS package to mount, unmount, and destroy the file system.

- Running the DBMS_FS.MAKE_ORACLE_FS procedure on the database instance is equivalent to running the mkfs command by root in an operating system.

- The tablespace that you specified in the fsoptions parameter must already exist before you execute the DBMS_FS.MAKE_ORACLE_FS procedure. To find existing tablespaces, query the DBA_TABLESPACES data dictionary view.

- The size of the file system is the same size as this tablespace.
Example

This example shows how to create an OFS file system:

BEGIN
  DBMS_FS.MAKE_ORACLE_FS (
    fstype   => 'ofs',
    fsname   => 'ofs_fs1',
    mount_options => 'TABLESPACE=ofs_fs1_tbspc');
END;
/

71.3.3 MOUNT_ORACLE_FS Procedure

This procedure mounts an Oracle file system on the specified mount point.

Syntax

DBMS_FS.MOUNT_ORACLE_FS (
  fstype IN VARCHAR2,
  fsname IN VARCHAR2,
  mount_point IN VARCHAR2,
  mount_options IN VARCHAR2);

Parameters

Table 71-4  MOUNT_ORACLE_FS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fstype</td>
<td>File system type. Valid options are:</td>
</tr>
<tr>
<td></td>
<td>· DBFS (database file system)</td>
</tr>
<tr>
<td></td>
<td>· OFS (Oracle file system)</td>
</tr>
<tr>
<td>fsname</td>
<td>Name of the file system. Enter a string no longer than 256 characters, using alpha numeric characters.</td>
</tr>
<tr>
<td>mount_point</td>
<td>Local directory where the file system should be mounted. This directory must already exist. Enter an absolute path. The maximum number of mount points that you can create is 64.</td>
</tr>
<tr>
<td>mount_options</td>
<td>Comma-separated mount options, listed in Table 71-5.</td>
</tr>
</tbody>
</table>

Usage Notes

Table 71-5  Supported Mount Options for the MOUNT_ORACLE_FS procedure

<table>
<thead>
<tr>
<th>Mount Option</th>
<th>Usage Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_permissions</td>
<td>Enables permission check and restrict access based on file mode. This option is useful with the allow_other mount option.</td>
</tr>
</tbody>
</table>
Table 71-5  (Cont.) Supported Mount Options for the MOUNT_ORACLE_FS procedure

<table>
<thead>
<tr>
<th>Mount Option</th>
<th>Usage Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow_other</td>
<td>Allows other users apart from the operating system user that did the mount can access the files. This will be used in conjunction with permission checks in determining the file access. This option requires setting the user_allow_other parameter in the /etc/fuse.conf configuration file on Linux.</td>
</tr>
<tr>
<td>max_read</td>
<td>Maximum size of the read operation. No maximum size is set by default.</td>
</tr>
<tr>
<td>max_write</td>
<td>Maximum write size in a single request. The default is 128K.</td>
</tr>
<tr>
<td>direct_io</td>
<td>Indicates to the operating system kernel not use file system cache.</td>
</tr>
<tr>
<td>nopersist</td>
<td>Does not store the mount options for use in next instance startup.</td>
</tr>
<tr>
<td>persist</td>
<td>Stores the mount entry persistently so that on subsequent instance startup it will be automatically mounted again.</td>
</tr>
<tr>
<td>ro</td>
<td>Mounts the file system in read-only mode. Files cannot be modified.</td>
</tr>
<tr>
<td>rw</td>
<td>Mounts the file system as read-write. This is the default.</td>
</tr>
<tr>
<td>nosuid</td>
<td>Specifies that the file system cannot contain set user id files.</td>
</tr>
<tr>
<td>suid</td>
<td>Specifies that the file system can contain set user id files. This is the default.</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure makes the file system visible in the local database instance.
- For more information about the file system types, see the fstype description in MAKE_ORACLE_FS Procedure.
- You can find information about currently mounted file systems by querying the V$OFSMOUNT dynamic view.
- Run the DBMS_FS.MOUNT_ORACLE_FS procedure on a file system that has already been created with DBMS_FS.MAKE_ORACLE_FS in the local computer node where the Oracle database instance is running. You cannot run this procedure on file systems that were created outside of Oracle Database.

Example

```
BEGIN
  DBMS_FS.MOUNT_ORACLE_FS (  
    fstype => 'dbfs',
    fsname => 'dbfs_fs1',
    mount_point => '/oracle/dbfs/testfs',
    mount_options => 'default_permissions, allow_other, persist');
END;
```
### 71.3.4 UNMOUNT_ORACLE_FS Procedure

This procedure unmounts an Oracle file system on the specified mount point.

**Syntax**

```sql
DBMS_FS.UNMOUNT_ORACLE_FS (    fsname IN VARCHAR2,    mount_point IN VARCHAR2,    unmount_options IN VARCHAR2);
```

**Table 71-6  UNMOUNT_ORACLE_FS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsname</td>
<td>Name of the file system</td>
</tr>
<tr>
<td>mount_point</td>
<td>Local directory where the file system had been mounted. Enter an absolute path.</td>
</tr>
<tr>
<td>unmount_options</td>
<td>Optionally, enter <code>force</code> to unmount the file system forcibly. This setting prevents new requests from being sent to the file system. All pending requests on the file system are either completed or cancelled. If you omit this setting, then attempts to unmount a busy file system cause an EBUSY error.</td>
</tr>
</tbody>
</table>

**Usage Notes**

- Before you unmount the file system, ensure that all applications that use this file system are shut down. Also ensure that no processes reference the mounting file system.
- You can find information about the currently mounted file systems by querying the `V$OSFSMOUNT` dynamic view.
- For more information about the file system types, see the `fstype` description in `MAKE_ORACLE_FS` Procedure.
- When an Oracle instance is shut down in normal immediate mode, then all the mounted file systems are automatically unmounted.
- If a file system is mounted from the operating system level with the `persist` option of the `mount` command, it is automatically mounted again when the database instance starts. If this file system is unmounted by executing `DBMS_FS.UNMOUNT_ORACLE_FS`, then it is not automatically mounted when the database instance starts.
- If you perform a `SHUTDOWN ABORT`, then the file system may still show as mounted but not accessible. In this case, you must unmount the system manually by calling `umount` command at the operating system level or `fusermount` procedure on Linux systems.
- You can export the local mount point of an Oracle file system to point to the remote system, and then NFS mount the file system from the remote system by using the operating system `mount` command. The `DBMS_FS.MOUNT_ORACLE_FS` procedure is similar to `mount` commands that are used for other local file systems.
• For better security, Oracle recommends that you use access control lists (ACLs) and Kerberos to control access to sensitive data.

• Do not attempt to unmount the file system from the operating system level. Doing so can leave the Oracle Database-created file system internal tables in an inconsistent state.

Example

BEGIN
    DBMS_FS.UNMOUNT_ORACLE_FS (    
        fsname          => 'dbfs_fs1',
        mount_point     => '/oracle/dbfs/testfs',
        mount_options   => 'force'    
    );
END;
The `DBMS_GOLDENGATE_ADM` package provides subprograms to configure and manage Oracle GoldenGate conflict detection and resolution.

This chapter contains the following topics:

- Using `DBMS_GOLDENGATE_ADM`
- Summary of `DBMS_GOLDENGATE_ADM` Subprograms

### 72.1 Using `DBMS_GOLDENGATE_ADM`

This section contains topics which relate to using the `DBMS_GOLDENGATE_ADM` package.

- `DBMS_GOLDENGATE_ADM` Overview
- `DBMS_GOLDENGATE_ADM` Security Model

### 72.1.1 `DBMS_GOLDENGATE_ADM` Overview

The `DBMS_GOLDENGATE_ADM` package provides interfaces to configure automatic conflict detection and resolution in an Oracle GoldenGate configuration that replicates tables between Oracle databases.

When more than one replica of a table allows changes to the table, a conflict can occur when a change is made to the same row in two different databases at nearly the same time. Oracle GoldenGate replicates changes using row logical change records (LCRs). It detects a conflict by comparing the old values in the row LCR with the current values of the corresponding table row identified by the key columns. If any column value does not match, then there is a conflict. After a conflict is detected, Oracle GoldenGate can resolve the conflict by overwriting values in the row with some values from the row LCR, ignoring the values in the row LCR, or computing a delta to update the row values.

XStream inbound servers and outbound servers can be used in an XStream configuration in a multitenant container database (CDB). A CDB is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs).

---

**Note:**

Using XStream requires purchasing a license for the Oracle GoldenGate product.
72.1.2 DBMS_GOLDENGATE_ADM Security Model

Security on this package can be controlled either by granting `EXECUTE` on this package to selected users or roles or by granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

An Oracle GoldenGate administrator must be configured at each Oracle database in the table's replication environment, and Oracle GoldenGate must be configured to replicate the table at each Oracle database. You can configure an Oracle GoldenGate administrator using the `GRANT_ADMIN_PRIVILEGE` procedure in the `DBMS_GOLDENGATE_ADM` package.

See Also:
The Oracle GoldenGate documentation for more information about Oracle GoldenGate replication and configuring an Oracle GoldenGate administrator

72.2 Summary of DBMS_GOLDENGATE_ADM Subprograms

**Table 72-1  DBMS_GOLDENGATE_ADM Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_AUTO_CDR Procedure</td>
<td>Configures Oracle GoldenGate automatic conflict detection and resolution for a table</td>
</tr>
<tr>
<td>ADD_AUTO_CDR_COLUMN_GROUP Procedure</td>
<td>Adds a column group and configures Oracle GoldenGate automatic conflict detection and resolution for the column group</td>
</tr>
<tr>
<td>ADD_AUTO_CDR_DELTA_RES Procedure</td>
<td>Configures Oracle GoldenGate automatic conflict detection and delta resolution for the column</td>
</tr>
<tr>
<td>ALTER_AUTO_CDR Procedure</td>
<td>Alters the Oracle GoldenGate automatic conflict detection and resolution for a table</td>
</tr>
<tr>
<td>ALTER_AUTO_CDR_COLUMN_GROUP Procedure</td>
<td>Alters a column group for Oracle GoldenGate automatic conflict detection and resolution</td>
</tr>
</tbody>
</table>
Table 72-1 (Cont.) DBMS_GOLDENGATE_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE_PROCREP_EXCLUSION_OBJ</td>
<td>Deletes a database object from the exclusion list for Oracle GoldenGate procedural replication</td>
</tr>
<tr>
<td>GG_PROCEDURE_REPLICATION_ON</td>
<td>Returns 1 if Oracle GoldenGate procedural replication is enabled and returns 0 if it is disabled</td>
</tr>
<tr>
<td>INSERT_PROCREP_EXCLUSION_OBJ</td>
<td>Inserts a database object into the exclusion list for Oracle GoldenGate procedural replication</td>
</tr>
<tr>
<td>PURGE_TOMBSTONES Procedure</td>
<td>Purges rows that were deleted before the specified timestamp from the tombstone table</td>
</tr>
<tr>
<td>REMOVE_AUTO_CDR Procedure</td>
<td>Removes Oracle GoldenGate automatic conflict detection and resolution for a table</td>
</tr>
<tr>
<td>REMOVE_AUTO_CDR_COLUMN_GROUP Procedure</td>
<td>Removes a column group that was configured for Oracle GoldenGate automatic conflict detection and resolution</td>
</tr>
<tr>
<td>REMOVE_AUTO_CDR_DELTA_RES Procedure</td>
<td>Removes Oracle GoldenGate automatic conflict detection and delta resolution for the column</td>
</tr>
</tbody>
</table>

Note:
All procedures commit unless specified otherwise.

72.2.1 ADD_AUTO_CDR Procedure

This procedure configures Oracle GoldenGate automatic conflict detection and resolution for a table.

The conflict detection and resolution configured by this procedure is based on the timestamp of the changes. The procedure adds one or more hidden columns of TIMESTAMP type to the table, and each hidden column is counted against the limit of 1,000 columns for each table.

The procedure automatically places the columns in the table into a default column group and into an unconditional supplemental log group, excluding nonscalar columns. To create column groups that include a subset of the columns in the table, use the ADD_AUTO_CDR_COLUMN_GROUP procedure in this package.

Syntax

```sql
DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR(
  schema_name             IN VARCHAR2,
  table_name              IN VARCHAR2,
  resolution_granularity  IN VARCHAR2 DEFAULT 'ROW',
  existing_data_timestamp IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  tombstone_deletes       IN BOOLEAN DEFAULT TRUE,
  fetchcols               IN BOOLEAN DEFAULT TRUE,
  record_conflicts        IN BOOLEAN DEFAULT FALSE,
  use_custom_handlers     IN BINARY_INTEGER DEFAULT 0);
```
Parameters

Table 72-2  ADD_AUTO_CDR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table’s schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>resolution_granularity</td>
<td>The default, adds one hidden TIMESTAMP column for the row and one hidden</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP column for each LOB column.</td>
</tr>
<tr>
<td></td>
<td>COLUMN adds one hidden TIMESTAMP column for each column in the table.</td>
</tr>
<tr>
<td>existing_data_timestamp</td>
<td>Timestamp to assign to existing rows.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the current system timestamp is used. If a time is specified,</td>
</tr>
<tr>
<td></td>
<td>and the operating system time zone is not a valid Oracle time zone, then</td>
</tr>
<tr>
<td></td>
<td>Oracle uses UTC as the default value.</td>
</tr>
<tr>
<td>tombstone_deletes</td>
<td>TRUE, the default, tracks deleted rows in a tombstone table.</td>
</tr>
<tr>
<td></td>
<td>Tracking deleted rows might be required to detect and resolve some conflicts,</td>
</tr>
<tr>
<td></td>
<td>but tracking deleted rows requires additional database resources.</td>
</tr>
<tr>
<td></td>
<td>FALSE does not track deleted rows in a tombstone table.</td>
</tr>
<tr>
<td>fetchcols</td>
<td>TRUE, the default, fetches the value of LOBs during conflict detection and</td>
</tr>
<tr>
<td></td>
<td>resolution. Fetching LOBs can be an expensive operation.</td>
</tr>
<tr>
<td></td>
<td>FALSE does not fetch the value of LOBs during conflict detection and</td>
</tr>
<tr>
<td></td>
<td>resolution.</td>
</tr>
<tr>
<td>record_conflicts</td>
<td>TRUE records the conflict in the DBA_APPLY_ERROR and DBA_APPLY_ERROR_MESSAGES</td>
</tr>
<tr>
<td></td>
<td>views.</td>
</tr>
<tr>
<td></td>
<td>FALSE, the default, does not record the conflict.</td>
</tr>
<tr>
<td>use_custom_handlers</td>
<td>0, the default, indicates that automatic conflict handlers are used.</td>
</tr>
<tr>
<td></td>
<td>1 indicates that automatic conflict handlers are not used and that a custom</td>
</tr>
<tr>
<td></td>
<td>error handler must be specified using the SET_DML_HANDLER procedure in the</td>
</tr>
<tr>
<td></td>
<td>DBMS_APPLY_ADM package.</td>
</tr>
</tbody>
</table>

72.2.2 ADD_AUTO_CDR_COLUMN_GROUP Procedure

This procedure adds a column group to a table that is configured for Oracle GoldenGate automatic conflict detection and resolution.

For a table that has been configured for timestamp conflict detection and resolution, this procedure adds a column group that includes a specified subset of columns in the table. Any columns in the table that are not part of a column group remain in the default column group for the table.

When you add a column group to a table, conflict detection and resolution is performed on the columns in the column group separately from the other columns in
the table. Column groups enable different databases to update different columns in the same row at nearly the same time without causing a conflict.

For example, a replicated table that contains employee information might have a salary column and a bonus column as well as other columns that identify the employee and a location column for the employees office number. Assume that one department in the company updates its database to change the employee’s salary while another department updates its database to change the employee’s location. If the salary and bonus columns are in a column group, these changes are applied to the replicated table in each database without requiring conflict resolution.

The procedure automatically places the columns in the column group into an unconditional supplemental log group, excluding nonscalar columns.

Before this procedure can be run on a table, the `DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR` procedure must be run in the table with `ROW` specified for the `resolution_granularity` parameter.

Syntax

```
DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_COLUMN_GROUP(
    schema_name             IN VARCHAR2,
    table_name              IN VARCHAR2,
    column_list             IN VARCHAR2,
    column_group_name       IN VARCHAR2 DEFAULT NULL,
    existing_data_timestamp IN TIMESTAMP WITH TIME ZONE DEFAULT NULL);
```

Parameters

**Table 72-3 ADD_AUTO_CDR_COLUMN_GROUP Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table’s schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>column_list</td>
<td>Group of columns for which the conflict detection and resolution is configured.</td>
</tr>
<tr>
<td></td>
<td>Specify the columns in a comma-separated list.</td>
</tr>
<tr>
<td></td>
<td>The same column cannot be in more than one column group.</td>
</tr>
<tr>
<td></td>
<td>Also, the same column cannot be in a column group and specified in a delta resolution.</td>
</tr>
<tr>
<td>column_group_name</td>
<td>The name of the column group.</td>
</tr>
<tr>
<td></td>
<td>If NULL, the column group name is system generated.</td>
</tr>
<tr>
<td>existing_data_timestamp</td>
<td>The time value for the added <code>TIMESTAMP</code> columns for existing table data.</td>
</tr>
</tbody>
</table>

72.2.3 ADD_AUTO_CDR_DELTA_RES Procedure

This procedure configures Oracle GoldenGate automatic conflict detection and delta resolution for the column.

The resolution method does not depend on a timestamp or an extra resolution column. With delta conflict resolution, the conflict is resolved by adding the difference between the new and old values in the LCR to the value in the table. For example, if a bank balance is updated at
two sites concurrently, then the converged value accounts for all debits and credits. This resolution method is generally used for financial data such as an account balance.

The procedure automatically places the column into an unconditional supplemental log group.

Before this procedure can be run on a table, the `DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR` procedure must be run in the table with `ROW` specified for the `resolution_granularity` parameter.

**Syntax**

```sql
DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_DELTA_RES(
    schema_name IN VARCHAR2,
    table_name  IN VARCHAR2,
    column_name IN VARCHAR2);
```

**Parameters**

Table 72-4   ADD_AUTO_CDR_DELTA_RES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table's schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column.</td>
</tr>
<tr>
<td></td>
<td>The specified column must be a <code>NUMBER</code> or <code>FLOAT</code> data type column.</td>
</tr>
<tr>
<td></td>
<td>The same column cannot be in a column group.</td>
</tr>
</tbody>
</table>

72.2.4 ALTER_AUTO_CDR Procedure

This procedure alters the Oracle GoldenGate automatic conflict detection and resolution for a table.

**Syntax**

```sql
DBMS_GOLDENGATE_ADM.ALTER_AUTO_CDR(
    schema_name             IN VARCHAR2,
    table_name              IN VARCHAR2,
    tombstone_deletes       IN BOOLEAN DEFAULT NULL,
    fetchcols               IN BOOLEAN DEFAULT NULL,
    record_conflicts        IN BOOLEAN DEFAULT NULL,
    use_custom_handlers     IN BINARY_INTEGER DEFAULT NULL);
```

**Parameters**

Table 72-5   ALTER_AUTO_CDR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table's schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
</tbody>
</table>
Table 72-5  (Cont.) ALTER_AUTO_CDR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tombstone_deletes</td>
<td>TRUE tracks deleted rows in a tombstone table. Tracking deleted rows might be required to detect and resolve some conflicts, but tracking deleted rows requires additional database resources. FALSE does not track deleted rows in a tombstone table. NULL retains the current setting for the parameter.</td>
</tr>
<tr>
<td>fetchcols</td>
<td>TRUE fetches the value of nonscalar columns during conflict detection and resolution. FALSE does not fetch the value of nonscalar columns during conflict detection and resolution. NULL retains the current setting for the parameter.</td>
</tr>
<tr>
<td>record_conflicts</td>
<td>TRUE records the conflict. FALSE does not record the conflict. NULL retains the current setting for the parameter.</td>
</tr>
<tr>
<td>use_custom_handlers</td>
<td>0 indicates that automatic conflict handlers are used. 1 indicates that automatic conflict handlers are not used and that a custom error handler must be specified using the SET_DML_HANDLER procedure in the DBMS_APPLY_ADM package. NULL, the default, retains the current setting for the parameter.</td>
</tr>
</tbody>
</table>

72.2.5 ALTER_AUTO_CDR_COLUMN_GROUP Procedure

This procedure alters a column group for Oracle GoldenGate automatic conflict detection and resolution.

Syntax

```sql
DBMS_GOLDENGATE_ADM.ALTER_AUTO_CDR_COLUMN_GROUP(
    schema_name        IN VARCHAR2,
    table_name         IN VARCHAR2,
    column_group_name  IN VARCHAR2,
    add_column_list    IN VARCHAR2,
    remove_column_list IN VARCHAR2);
```

Parameters

Table 72-6  ALTER_AUTO_CDR_COLUMN_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table’s schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>column_group_name</td>
<td>The name of the column group.</td>
</tr>
<tr>
<td>add_column_list</td>
<td>A comma-separated list of columns to add to the column group.</td>
</tr>
</tbody>
</table>
### 72.2.6 DELETE_PROCREP_EXCLUSION_OBJ Procedure

This procedure deletes a database object from the exclusion list for Oracle GoldenGate procedural replication.

When a database object is on the exclusion list for Oracle GoldenGate procedural replication, execution of subprogram in the package is not replicated if the subprogram operates on the excluded object. For example, if `hr.employees` is an excluded database object for the `DBMS_REDEFINITION` package, then an execution of the `DBMS_REDEFINITION.START_REDEF_TABLE` procedure on the `hr.employees` table is not replicated.

⚠️ **Caution:**

Run the `DELETE_PROCREP_EXCLUSION_OBJ` procedure only under the direction of Oracle Support.

#### Syntax

```sql
DBMS_GOLDENGATE_ADM.DELETE_PROCREP_EXCLUSION_OBJ(
    package_owner     IN VARCHAR2 DEFAULT NULL,
    package_name      IN VARCHAR2 DEFAULT NULL,
    object_owner      IN VARCHAR2 DEFAULT NULL,
    object_name       IN VARCHAR2 DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_owner</td>
<td>The owner of the package.</td>
</tr>
<tr>
<td>package_name</td>
<td>The name of the package.</td>
</tr>
<tr>
<td>object_owner</td>
<td>The owner of the object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object.</td>
</tr>
</tbody>
</table>
72.2.7 GG_PROCEDURE_REPLICATION_ON Function

This function returns 1 if Oracle GoldenGate procedural replication is enabled and returns 0 if it is disabled.

Syntax

DBMS_STREAMS.GG_PROCEDURE_REPLICATION_ON
RETURN NUMBER;

72.2.8 INSERT_PROCREP_EXCLUSION_OBJ Procedure

This procedure inserts a database object into the exclusion list for Oracle GoldenGate procedural replication.

When a database object is on the exclusion list for Oracle GoldenGate procedural replication, execution of subprogram in the package is not replicated if the subprogram operates on the excluded object. For example, if hr.employees is an excluded database object for the DBMS_REDEFINITION package, then an execution of the DBMS_REDEFINITION.START_REDEF_TABLE procedure on the hr.employees table is not replicated.

Caution:

Run the INSERT_PROCREP_EXCLUSION_OBJ procedure only under the direction of Oracle Support.

Syntax

DBMS_GOLDENGATE_ADM.INSERT_PROCREP_EXCLUSION_OBJ(
    package_owner     IN VARCHAR2 DEFAULT NULL,
    package_name      IN VARCHAR2 DEFAULT NULL,
    object_owner      IN VARCHAR2 DEFAULT NULL,
    object_name       IN VARCHAR2 DEFAULT NULL);

Parameters

Table 72-8 INSERT_PROCREP_EXCLUSION_OBJ Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_owner</td>
<td>The owner of the package.</td>
</tr>
<tr>
<td>package_name</td>
<td>The name of the package.</td>
</tr>
<tr>
<td>object_owner</td>
<td>The owner of the object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object.</td>
</tr>
</tbody>
</table>
72.2.9 PURGE_TOMBSTONES Procedure

This procedure purges rows that were deleted before the specified timestamp from the tombstone table.

Syntax

```
DBMS_GOLDENGATE_ADM.PURGE_TOMBSTONES(
    purge_timestamp IN TIMESTAMP WITH TIME ZONE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>purge_timestamp</td>
<td>The timestamp before which records are purged.</td>
</tr>
</tbody>
</table>

72.2.10 REMOVE_AUTO_CDR Procedure

This procedure removes Oracle GoldenGate automatic conflict detection and resolution for a table.

Syntax

```
DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR(
    schema_name             IN VARCHAR2,
    table_name              IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table’s schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
</tbody>
</table>

72.2.11 REMOVE_AUTO_CDR_COLUMN_GROUP Procedure

This procedure removes a column group that was configured for Oracle GoldenGate automatic conflict detection and resolution.

Syntax

```
DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR_COLUMN_GROUP(
    schema_name             IN VARCHAR2,
    table_name              IN VARCHAR2,
    column_group_name       IN VARCHAR2);
```
Parameters

Table 72-11  REMOVE_AUTO_CDR_COLUMN_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table’s schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>column_group_name</td>
<td>The name of the column group.</td>
</tr>
</tbody>
</table>

72.2.12 REMOVE_AUTO_CDR_DELTA_RES Procedure

This procedure removes Oracle GoldenGate automatic conflict detection and delta resolution for the column.

Syntax

```sql
DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR_DELTA_RES(
    schema_name IN VARCHAR2,
    table_name  IN VARCHAR2,
    column_name IN VARCHAR2);
```

Parameters

Table 72-12  REMOVE_AUTO_CDR_DELTA_RES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the table’s schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column.</td>
</tr>
</tbody>
</table>
The **DBMS_GOLDENGATE_AUTH** package provides subprograms for granting privileges to and revoking privileges from GoldenGate administrators.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_GOLDENGATE_AUTH Subprograms

### See Also:

- GRANT_ADMIN_PRIVILEGE Procedure in the DBMS_XSTREAM_AUTH package

## 73.1 DBMS_GOLDENGATE_AUTH Overview

This package provides subprograms for granting privileges to GoldenGate administrators and revoking privileges from GoldenGate administrators. A GoldenGate administrator manages an integrated GoldenGate and XStream Out configuration.

GoldenGate administrators can be used in a multitenant container database (CDB). A CDB is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs).

### See Also:

- GRANT_ADMIN_PRIVILEGE Procedure in the DBMS_XSTREAM_AUTH package
- Oracle Database XStream Guide
- Oracle Database Concepts for more information about CDBs and PDBs

## 73.2 DBMS_GOLDENGATE_AUTH Security Model

Security on this package can be controlled by granting **EXECUTE** on this package to selected users or roles, or by granting **EXECUTE_CATALOG_ROLE** to selected users or roles.

The user executing the subprograms in the **DBMS_GOLDENGATE_AUTH** package must have **SYSDBA** administrative privilege, and the user must exercise the privilege using **AS SYSDBA** at connect time.
If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, connect as an administrative user who can create users, grant privileges, and create tablespaces when using this package.

### 73.3 Summary of DBMS_GOLDENGATE_AUTH Subprograms

The DBMS_XSTREAM_AUTH package includes the `GRANT_ADMIN_PRIVILEGE` procedure and `REVOKE_ADMIN_PRIVILEGE` procedure subprograms.

<table>
<thead>
<tr>
<th>Table 73-1</th>
<th>DBMS_GOLDENGATE_AUTH Package Subprograms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subprogram</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><code>GRANT_ADMIN_PRIVILEGE Procedure</code></td>
<td>Either grants the privileges needed by a user to be a GoldenGate administrator directly, or generates a script that grants these privileges</td>
</tr>
<tr>
<td><code>REVOKE_ADMIN_PRIVILEGE Procedure</code></td>
<td>Either revokes GoldenGate administrator privileges from a user directly, or generates a script that revokes these privileges</td>
</tr>
</tbody>
</table>

**Note:**
All subprograms commit unless specified otherwise.

### 73.3.1 GRANT_ADMIN_PRIVILEGE Procedure

This procedure grants the privileges needed by a user to be a GoldenGate administrator.

**See Also:**
- `GRANT_ADMIN_PRIVILEGE Procedure` in the DBMS_XSTREAM_AUTH package

**Syntax**

```sql
DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee IN VARCHAR2,
    privilege_type IN VARCHAR2 DEFAULT '*',
    grant_select_privileges IN BOOLEAN DEFAULT TRUE,
    do_grants IN BOOLEAN DEFAULT TRUE,
    file_name IN VARCHAR2 DEFAULT NULL,
    directory_name IN VARCHAR2 DEFAULT NULL
)
```
grant_optional_privileges  IN  VARCHAR2  DEFAULT NULL,
container                  IN  VARCHAR2  DEFAULT 'CURRENT');

Parameters

Table 73-2  GRANT_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user to whom privileges are granted</td>
</tr>
<tr>
<td>privilege_type</td>
<td>Specify one of the following values:</td>
</tr>
</tbody>
</table>
|                            | · CAPTURE
|                            |     Specifying CAPTURE grants the minimum privileges required by the user    |
|                            |     to administer Oracle GoldenGate integrated extract.                     |
|                            | · APPLY
|                            |     Specifying APPLY grants the minimum privileges required by the user     |
|                            |     to administer Oracle GoldenGate integrated replicat.                    |
|                            | · *
|                            |     Specifying * grants the minimum privileges required by the user to      |
|                            |     administer Oracle GoldenGate integrated extract and Oracle              |
|                            |     GoldenGate integrated replicat.                                         |
| grant_select_privileges    | If TRUE, then the procedure grants a set of privileges, including SELECT     |
|                            | _CATALOG_ROLE, to the user. This setting is recommended for GoldenGate      |
|                            | administrators.                                                            |
|                            | If FALSE, then the procedure does not grant the set of privileges to the    |
|                            | user.                                                                     |
|                            | SELECT_CATALOG_ROLE enables the user to select from the data                |
|                            | dictionary.                                                               |
| do_grants                  | If TRUE, then the procedure grants the privileges to the specified grantee  |
|                            | directly, and adds the grantee to the DBA_GOLDENGATE_PRIVILEGES data       |
|                            | dictionary view. If the user already has an entry in this data dictionary   |
|                            | view, then the procedure does not make another entry, and no error is      |
|                            | raised. If TRUE and any of the grant statements fails, then the procedure  |
|                            | raises an error.                                                           |
|                            | If FALSE, then the procedure does not grant the privileges to the specified |
|                            | grantee directly, and does not add the grantee to the DBA_GOLDENGATE       |
|                            | PRIVILEGES data dictionary view. You specify FALSE when the procedure is    |
|                            | generating a file that you will run later. If you specify FALSE and either |
|                            | the file_name or directory_name parameter is NULL, then the procedure      |
|                            | raises an error.                                                           |
|                            | Note: It is recommended that do_grants be set to TRUE because many APIs    |
|                            | check for the presence of a user in the DBA_GOLDENGATE_PRIVILEGES view,    |
|                            | which will not be populated if do_grants is set to FALSE.                   |
| file_name                  | The name of the file generated by the procedure. The file contains all of   |
|                            | the statements that grant the privileges. If a file with the specified file |
|                            | name exists in the specified directory name, then the grant statements are  |
|                            | appended to the existing file. If NULL, then the procedure does not        |
|                            | generate a file.                                                           |
Table 73-2  (Cont.) GRANT_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory_name</td>
<td>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have the WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
| grant_optional_privilges| A comma-separated list of optional privileges to grant to the grantee. You can specify the following roles and privileges:  
  - XDBADMIN  
  - DV_XSTREAM_ADMIN  
  - DV_GOLDENGATE_ADMIN  
  - EXEMPT_ACCESS_POLICY  
  - EXEMPT_REDACTION_POLICY  |
| container               | If CURRENT, then grants privileges to the grantee only in the container where the procedure is invoked. CURRENT can be specified while connected to the root or to a PDB. If ALL, then grants privileges to the grantee in all containers in the CDB and all PDBs created after the procedure is invoked. To specify ALL, the procedure must be invoked in the root by a common user. If a container name, then grants privileges to the grantee only in the specified container. To specify root, use CDB$ROOT while connected to the root. To specify a PDB, the procedure must be invoked in the root. **Note:** This parameter only applies to CDBs. |

Usage Notes

The user who runs the procedure must be an administrative user who can grant privileges to other users.

Specifically, the procedure grants the following privileges to the specified user:

- **The RESTRICTED SESSION system privilege**
- **EXECUTE on the following packages:**
  - DBMS_APPLY_ADM
  - DBMS_AQ
  - DBMS_AQADM
  - DBMS_AQIN
  - DBMS_AQELM
  - DBMS_CAPTURE_ADM
  - DBMS_FLASHBACK
  - DBMS_LOCK
  - DBMS_PROPAGATION_ADM
Privileges to enqueue messages into and dequeue messages from any queue

Privileges to manage any queue

Privileges to create, alter, and execute any of the following types of objects in the user’s own schema and in other schemas:

- Evaluation contexts
- Rule sets
- Rules

In addition, the grantee can grant these privileges to other users.

- SELECT_CATALOG_ROLE
- SELECT or READ privilege on data dictionary views related to GoldenGate and Oracle Streams
- The ability to allow a remote GoldenGate administrator to perform administrative actions through a database link by connecting to the grantee

This ability is enabled by running the GRANT_REMOTE_ADMIN_ACCESS procedure in this package.

**Note:**

This procedure grants only the privileges necessary to configure and administer a GoldenGate environment. You can grant additional privileges to the grantee if necessary.

**See Also:**

GRANT_ADMIN_PRIVILEGE Procedure in the DBMS_XSTREAM_AUTH package
73.3.2 REVOKE_ADMIN_PRIVILEGE Procedure

This procedure revokes GoldenGate administrator privileges from a user.

Syntax

```sql
DBMS_GOLDENGATE_AUTH.REVOKE_ADMIN_PRIVILEGE(
    grantee                     IN  VARCHAR2,
    privilege_type              IN  VARCHAR2  DEFAULT '*',
    revoke_select_privileges    IN  BOOLEAN   DEFAULT FALSE,
    do_revokes                  IN  BOOLEAN   DEFAULT TRUE,
    file_name                   IN  VARCHAR2  DEFAULT NULL,
    directory_name              IN  VARCHAR2  DEFAULT NULL,
    revoke_optional_privileges  IN  VARCHAR2  DEFAULT NULL,
    container                   IN  VARCHAR2  DEFAULT 'CURRENT');
```

Parameters

Table 73-3  REVOKE_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user from whom privileges are revoked</td>
</tr>
<tr>
<td>privilege_type</td>
<td>Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td>-  CAPTURE</td>
</tr>
<tr>
<td></td>
<td>Specifying CAPTURE revokes the minimum privileges required by the user to</td>
</tr>
<tr>
<td></td>
<td>administer Oracle GoldenGate integrated extract.</td>
</tr>
<tr>
<td></td>
<td>-  APPLY</td>
</tr>
<tr>
<td></td>
<td>Specifying APPLY revokes the minimum privileges required by the user to</td>
</tr>
<tr>
<td></td>
<td>administer Oracle GoldenGate integrated replicat.</td>
</tr>
<tr>
<td></td>
<td>-  *</td>
</tr>
<tr>
<td></td>
<td>Specifying * revokes the minimum privileges required by the user to</td>
</tr>
<tr>
<td></td>
<td>administer Oracle GoldenGate integrated extract and</td>
</tr>
<tr>
<td></td>
<td>Oracle GoldenGate integrated replicat.</td>
</tr>
<tr>
<td>revoke_select_privileges</td>
<td>If TRUE, then the procedure revokes SELECT_CATALOG_ROLE from the user.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not revoke</td>
</tr>
<tr>
<td></td>
<td>SELECT_CATALOG_ROLE to the user.</td>
</tr>
<tr>
<td></td>
<td>SELECT_CATALOG_ROLE enables the user to select from the data</td>
</tr>
<tr>
<td></td>
<td>dictionary.</td>
</tr>
</tbody>
</table>

Table 73-3  (Cont.) REVOKE_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>do_revokes</td>
<td>If TRUE, then the procedure revokes the privileges from the specified user directly, and removes the user from the DBA_XSTREAM_ADMINISTRATOR data dictionary view. If the user does not have a record in this data dictionary view, then the procedure does not remove a record from the view, and no error is raised. If TRUE and any of the revoke statements fails, then the procedure raises an error. A revoke statement fails if the user is not granted the privilege that is being revoked. If FALSE, then the procedure does not revoke the privileges from the specified user directly, and does not remove the user from the DBA_XSTREAM_ADMINISTRATOR data dictionary view. You specify FALSE when the procedure is generating a file that you will run later. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure does not raise an error.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file generated by this procedure. The file contains all of the statements that revoke the privileges. If a file with the specified file name exists in the specified directory name, then the revoke statements are appended to the existing file. If NULL, then the procedure does not generate a file.</td>
</tr>
<tr>
<td>directory_name</td>
<td>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have the WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>revoke_optional_privileges</td>
<td>A comma-separated list of optional privileges to revoke from the grantee, such as the DV_XSTREAM_ADMIN and DV_GOLDENGATE_ADMIN privileges</td>
</tr>
<tr>
<td>container</td>
<td>If CURRENT, then revokes privileges from the grantee only in the container where the procedure is invoked. CURRENT can be specified while connected to the root or to a PDB. If ALL, then revokes privileges from the grantee in all containers in the CDB. To specify ALL, the procedure must be invoked in the root. If a container name, then revokes privileges from the grantee only in the specified container. To specify root, use CDB$ROOT while connected to the root. To specify a PDB, the procedure must be invoked in the root. Note: This parameter only applies to CDBs.</td>
</tr>
</tbody>
</table>

Usage Notes

The user who runs this procedure must be an administrative user who can revoke privileges from other users. Specifically, this procedure revokes the privileges granted by running the GRANT_ADMIN_PRIVILEGE procedure in this package.
See Also:

"GRANT_ADMIN_PRIVILEGE Procedure"
The DBMS_HADOOP package provides a PL/SQL procedure called CREATE_EXTDDL_FOR_HIVE(), that creates an Oracle external table for a given hive table.

Big Data SQL needs to be correctly set up for DBMS_HADOOP to work.

This chapter contains the following topics:

• DBMS_HADOOP Overview
• DBMS_HADOOP Security Model
• Summary of DBMS_HADOOP Subprograms

74.1 DBMS_HADOOP Overview

The DBMS_HADOOP package provides two procedures for creating an Oracle external table and for synchronizing the Oracle external table partitions.

These procedures are:

• CREATE_EXTDDL_FOR_HIVE() — creates an Oracle external table for a given hive table
• SYNCHRONIZE_PARTITIONS_FOR_HIVE() — helps to synchronize the Oracle external table partitions with those in the corresponding hive table

74.2 DBMS_HADOOP Security Model

Users must have ALTER privileges on the table for modifying external table parameters. In addition the ALTER privileges, users must have READ privileges for the directory object that contains the external data source and WRITE privileges for the directory objects containing bad, log, and discard files.

Similar privileges are applicable to the partitioned external tables.

74.3 Summary of DBMS_HADOOP Subprograms

DBMS_HADOOP includes the CREATE_EXTDDL_FOR_HIVE procedure and SYNC_PARTITIONS_FOR_HIVE procedure subprograms.

Table 74-1  DBMS_HADOOP Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_EXTDDL_FOR_HIVE Procedure</td>
<td>Given a hive table name, creates a text of DDL that can be executed to create an external table corresponding to the hive table</td>
</tr>
<tr>
<td>SYNC_PARTITIONS_FOR_HIVE Procedure</td>
<td>Synchronizes the existing partitioning definition of a hive table in Oracle catalog</td>
</tr>
</tbody>
</table>
74.3.1 CREATE_EXTDDL_FOR_HIVE Procedure

This procedure creates an Oracle external table for a given hive table.

Syntax

```
DBMS_HADOOP.CREATE_EXTDDL_FOR_HIVE (  
    cluster_id         IN  VARCHAR2,  
    db_name            IN  VARCHAR2 := NULL,  
    hive_table_name    IN  VARCHAR2,  
    hive_partition     IN  BOOLEAN,  
    table_name         IN  VARCHAR2 := NULL,  
    perform_DDL        IN  BOOLEAN := FALSE,  
    text_of_DDL        OUT CLOB);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster_id</td>
<td>Hadoop cluster ID</td>
</tr>
<tr>
<td>hive_db_name</td>
<td>Database where the hive table is located</td>
</tr>
<tr>
<td>hive_table_name</td>
<td>Name of the hive table</td>
</tr>
<tr>
<td>hive_partition</td>
<td>If this argument is TRUE and if the original hive table is partitioned, the corresponding Oracle external table will also be partitioned using the same partition key(s). If the original hive table is not partitioned, hive_partition=TRUE has no effect. If hive_partition=FALSE, the corresponding Oracle external table will not be partitioned even if the original hive table is partitioned.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the Oracle external table</td>
</tr>
<tr>
<td>perform_DDL</td>
<td>If this argument is TRUE, the external table will be automatically created. Otherwise, only the textual representation of the DDL statement will be generated and returned in text_of_DDL.</td>
</tr>
<tr>
<td>text_of_DDL</td>
<td>If the argument perform_DDL is FALSE, only the textual representation of the DDL statement will be generated and returned in text_of_DDL.</td>
</tr>
</tbody>
</table>

74.3.2 SYNCHRONIZE_PARTITIONS_FOR_HIVE Procedure

This procedure synchronizes the Oracle external table partitions with those in the corresponding hive table.

Syntax

```
DBMS_HADOOP.SYNCHRONIZE_PARTITIONS_FOR_HIVE (  
    table_name     IN  VARCHAR2,  
    table_owner    IN  VARCHAR2);  
```
## Parameters

### Table 74-3  SYNCHRONIZE_PARTITIONS_FOR_HIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Oracle external table</td>
</tr>
<tr>
<td>table_owner</td>
<td>Schema name</td>
</tr>
</tbody>
</table>
The DBMS_HANG_MANAGER package provides a method of changing some Hang Manager configuration parameters.

This chapter contains the following topics:

- DBMS_HANG_MANAGER Overview
- DBMS_HANG_MANAGER Security Model
- DBMS_HANG_MANAGER Constants
- DBMS_HANG_MANAGER Exceptions
- Summary of DBMS_HANG_MANAGER Subprograms

### 75.1 DBMS_HANG_MANAGER Overview

The DBMS_HANG_MANAGER package provides a method of changing some Hang Manager configuration parameters.

Hang Manager is always enabled on all databases, Automatic Storage Management (ASM) disk groups, and single instance systems (when not using Oracle RAC). Hang resolution is enabled on all instances by default except for single instance systems (when not using Oracle RAC) where it is disabled and cannot be enabled at this time.

**Note:**
The DBMS_HANG_MANAGER package is intended for use by database administrators (DBA) only. It is not intended for use by application developers.

### 75.2 DBMS_HANG_MANAGER Security Model

DBMS_HANG_MANAGER can only be run as the SYS user.

### 75.3 DBMS_HANG_MANAGER Constants

The table in this topic lists DBMS_HANG_MANAGER package constants.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOLUTION_SCOPE</td>
<td>VARCHAR2</td>
<td>resolution scope</td>
<td>Resolution scope parameter name</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SENSITIVITY</td>
<td>VARCHAR2</td>
<td>sensitivity</td>
<td>Sensitivity parameter name</td>
</tr>
<tr>
<td>BASE_FILE_SIZE_LIMIT</td>
<td>VARCHAR2</td>
<td>base file size limit</td>
<td>Base file size limit parameter name. The default value of this parameter is 100 MB.</td>
</tr>
<tr>
<td>BASE_FILE_SET_COUNT</td>
<td>VARCHAR2</td>
<td>base file set count</td>
<td>Base file set count parameter name. The default value is 5 trace files in the circular trace file set.</td>
</tr>
<tr>
<td>LWS_FILE_SIZE_LIMIT</td>
<td>VARCHAR2</td>
<td>long waiting session file size limit</td>
<td>Long waiting session file size limit parameter name. The default value of this parameter is 100 MB.</td>
</tr>
<tr>
<td>LWS_FILE_SET_COUNT</td>
<td>VARCHAR2</td>
<td>long waiting session set count</td>
<td>Long waiting session set count parameter name. The default value is 5 trace files in the circular trace file set.</td>
</tr>
<tr>
<td>RESOLUTION_SCOPE_PROCESS</td>
<td>VARCHAR2</td>
<td>PROCESS</td>
<td>Enables only session and/or process termination</td>
</tr>
<tr>
<td>RESOLUTION_SCOPE_INSTANCE</td>
<td>VARCHAR2</td>
<td>INSTANCE</td>
<td>Enables session, process, and instance termination. The latter is used in specific scenarios. The default value is INSTANCE.</td>
</tr>
<tr>
<td>SENSITIVITY_NORMAL</td>
<td>VARCHAR2</td>
<td>NORMAL</td>
<td>Sets various Hang Manager intervals to their default values. The default value is NORMAL.</td>
</tr>
<tr>
<td>SENSITIVITY_HIGH</td>
<td>VARCHAR2</td>
<td>HIGH</td>
<td>Sets the various Hang Manager intervals to half of their default values.</td>
</tr>
</tbody>
</table>
75.4 DBMS_HANG_MANAGER Exceptions

The table in this topic lists the exceptions raised by the DBMS_HANG_MANAGER package.

Table 75-2   DBMS_HANG_MANAGER Package Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRNUM_INPUT_ERROR</td>
<td>-32706</td>
<td>Invalid user input</td>
</tr>
<tr>
<td>ERRNUM_INTERNAL_ERROR</td>
<td>-32707</td>
<td>Database experienced an error setting a parameter</td>
</tr>
<tr>
<td>ERRNUM_UNSUPPORTED_ERROR</td>
<td>-32708</td>
<td>Operation is not supported on this instance type</td>
</tr>
</tbody>
</table>

75.5 Summary of DBMS_HANG_MANAGER Subprograms

The DBMS_HANG_MANAGER package includes the SET procedure subprogram.

Table 75-3   DBMS_HANG_MANAGER Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET Procedure</td>
<td>Sets the specified parameter to the specified value</td>
</tr>
</tbody>
</table>

75.5.1 SET Procedure

This procedure sets the specified parameters to specified values.

Syntax

```sql
DBMS_HANG_MANAGER.SET (  
    pname   IN  VARCHAR2,  
    pvalue  IN  VARCHAR2);  
```

Parameters

Table 75-4   SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>Contains the parameter name that needs to be modified</td>
</tr>
<tr>
<td>pvalue</td>
<td>Contains the new value for the parameter name specified in pname</td>
</tr>
</tbody>
</table>
The DBMS_HEAT_MAP package provides an interface to externalize heatmaps at various levels of storage including block, extent, segment, object and tablespace. A second set of subprograms externalize the heatmaps materialized by the background for top N tablespaces.

This chapter contains the following topics:

• Overview
• Security Model
• Summary of DBMS_HEAT_MAP Subprograms

See Also:

• Heat Map in Oracle Database VLDB and Partitioning Guide
• DBMS_ILM
• DBMS_ILM_ADMIN

76.1 DBMS_HEAT_MAP Overview

To implement your ILM strategy, you can use Heat Map in Oracle Database to track data access and modification. You can also use Automatic Data Optimization (ADO) to automate the compression and movement of data between different tiers of storage within the database.

The Heat Map tracks modification times at the block level, and multiple access statistics at the segment level. Objects in the SYSTEM and SYSAUX tablespaces are not tracked. DBMS_HEAT_MAP gives you access to the Heat Map statistics at various levels - block, extent, segment, object, and tablespace.

76.2 DBMS_HEAT_MAP Security Model

The execution privilege is granted to PUBLIC. Procedures in this package run under the caller security. The user must have ANALYZE privilege on the object.

76.3 Summary of DBMS_HEAT_MAP Subprograms

This table lists and briefly describes the DBMS_HEAT_MAP package subprograms.
Table 76-1  DBMS_HEAT_MAP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK_HEAT_MAP</td>
<td>Returns last modification time for each block in a table segment</td>
</tr>
<tr>
<td>EXTENT_HEAT_MAP</td>
<td>Returns the extent level Heat Map statistics for a table segment</td>
</tr>
<tr>
<td>OBJECT_HEAT_MAP</td>
<td>Returns the minimum, maximum and average access times for all the segments belonging to the object</td>
</tr>
<tr>
<td>SEGMENT_HEAT_MAP</td>
<td>Returns the heatmap attributes for the given segment</td>
</tr>
<tr>
<td>TABLESPACE_HEAT_MAP</td>
<td>Returns the minimum, maximum and average access times for all the segments in the tablespace</td>
</tr>
</tbody>
</table>

76.3.1 BLOCK_HEAT_MAP Function

This table function returns the last modification time for each block in a table segment. It returns no information for segment types that are not data.

Syntax

```sql
DBMS_HEAT_MAP.BLOCK_HEAT_MAP (
    owner             IN VARCHAR2,  
    segment_name      IN VARCHAR2,  
    partition_name    IN VARCHAR2 DEFAULT NULL,  
    sort_columnid     IN NUMBER DEFAULT NULL,  
    sort_order        IN VARCHAR2 DEFAULT NULL)
RETURN hm_bls_row PIPELINED;
```

Parameters

Table 76-2  BLOCK_HEAT_MAP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the segment</td>
</tr>
<tr>
<td>segment_name</td>
<td>Table name of a non-partitioned table or (sub)partition of partitioned table. Returns no rows when table name is specified for a partitioned table.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Defaults to NULL. For a partitioned table, specify the partition or subpartition segment name.</td>
</tr>
<tr>
<td>sort_columnid</td>
<td>ID of the column on which to sort the output. Valid values 1..9. Invalid values are ignored.</td>
</tr>
<tr>
<td>sort_order</td>
<td>Defaults to NULL. Possible values: ASC, DESC</td>
</tr>
</tbody>
</table>
Return Values

Table 76-3  BLOCK_HEAT_MAP Function Return Values (Output Parameters)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the segment</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name of the non-partitioned table</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Tablespace containing the segment</td>
</tr>
<tr>
<td>file_id</td>
<td>Absolute file number of the block in the segment</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Relative file number of the block in the segment</td>
</tr>
<tr>
<td>block_id</td>
<td>Block number of the block</td>
</tr>
<tr>
<td>write_time</td>
<td>Last modification time of the block</td>
</tr>
</tbody>
</table>

76.3.2 EXTENT_HEAT_MAP Function

This table function returns the extent level Heat Map statistics for a table segment. It returns no information for segment types that are not data. Aggregates at extent level, including minimum modification time and maximum modification time, are included.

Syntax

```sql
DBMS_HEAT_MAP.EXTENT_HEAT_MAP (  
    owner             IN VARCHAR2,
    segment_name      IN VARCHAR2,
    partition_name    IN VARCHAR2 DEFAULT NULL,
    RETURN hm_els_row PIPELINED;
)
```

Parameters

Table 76-4  EXTENT_HEAT_MAP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the segment</td>
</tr>
<tr>
<td>segment_name</td>
<td>Table name of a non-partitioned table or (sub)partition of partitioned table. Returns no rows when table name is specified for a partitioned table.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Defaults to NULL. For a partitioned table, specify the partition or subpartition segment name.</td>
</tr>
</tbody>
</table>

Return Values

Table 76-5  EXTENT_HEAT_MAP Function Return Values (Output Parameters)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the segment</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name of the non-partitioned table</td>
</tr>
</tbody>
</table>
Table 76-5  (Cont.) EXTENT_HEAT_MAP Function Return Values (Output Parameters)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Tablespace containing the segment</td>
</tr>
<tr>
<td>file_id</td>
<td>Absolute file number of the block in the segment</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Relative file number of the block in the segment</td>
</tr>
<tr>
<td>block_id</td>
<td>Block number of the block</td>
</tr>
<tr>
<td>blocks</td>
<td>Number of blocks in the extent</td>
</tr>
<tr>
<td>bytes</td>
<td>Number of bytes in the extent</td>
</tr>
<tr>
<td>min_writetime</td>
<td>Minimum of last modification time of the block</td>
</tr>
<tr>
<td>max_writetime</td>
<td>Maximum of last modification time of the block</td>
</tr>
<tr>
<td>avg_writetime</td>
<td>Average of last modification time of the block</td>
</tr>
</tbody>
</table>

76.3.3 OBJECT_HEAT_MAP Function

This table function returns the minimum, maximum and average access times for all the segments belonging to the object.

The object must be a table. The table function raises an error if called on object tables other than table.

Syntax

```sql
DBMS_HEAT_MAP.OBJECT_HEAT_MAP (
    object_owner      IN VARCHAR2,
    object_name       IN VARCHAR2)
RETURN hm_object_table PIPELINED;
```

Parameters

Table 76-6  OBJECT_HEAT_MAP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_owner</td>
<td>Tablespace containing the segment</td>
</tr>
<tr>
<td>object_name</td>
<td>Segment header relative file number</td>
</tr>
</tbody>
</table>

Return Values

Table 76-7  OBJECT_HEAT_MAP Function Return Values (Output Parameters)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_name</td>
<td>Name of the top level segment</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Name of the tablespace</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of segment as in DBA_SEGMENTS.SEGMENT_TYPE</td>
</tr>
</tbody>
</table>
Table 76-7  (Cont.) OBJECT_HEAT_MAP Function Return Values (Output Parameters)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_size</td>
<td>Segment size in bytes</td>
</tr>
<tr>
<td>min_writetime</td>
<td>Oldest write time for the segment</td>
</tr>
<tr>
<td>max_writetime</td>
<td>Latest write time for the segment</td>
</tr>
<tr>
<td>avg_writetime</td>
<td>Average write time for the segment</td>
</tr>
<tr>
<td>min_readtime</td>
<td>Oldest read time for the segment</td>
</tr>
<tr>
<td>max_readtime</td>
<td>Latest read time for the segment</td>
</tr>
<tr>
<td>avg_readtime</td>
<td>Average read time for the segment</td>
</tr>
<tr>
<td>min_lookuptime</td>
<td>Oldest index lookup time for the segment</td>
</tr>
<tr>
<td>max_lookuptime</td>
<td>Latest index lookup time for the segment</td>
</tr>
<tr>
<td>avg_lookuptime</td>
<td>Average index lookup time for the segment</td>
</tr>
<tr>
<td>min_ftstime</td>
<td>Oldest full table scan time for the segment</td>
</tr>
<tr>
<td>max_ftstime</td>
<td>Latest full table scan time for the segment</td>
</tr>
<tr>
<td>avg_ftstime</td>
<td>Average full table scan time for the segment</td>
</tr>
</tbody>
</table>

76.3.4 SEGMENT_HEAT_MAP Procedure

This procedure returns the heatmap attributes for the given segment.

Syntax

```sql
DBMS_HEAT_MAP.SEGMENT_HEAT_MAP (  
    tablespace_id          IN  NUMBER,  
    header_file            IN  NUMBER,  
    header_block           IN  NUMBER,  
    segment_objd           IN  NUMBER,  
    min_writetime          OUT DATE,  
    max_writetime          OUT DATE,  
    avg_writetime          OUT DATE,  
    min_readtime           OUT DATE,  
    max_readtime           OUT DATE,  
    avg_readtime           OUT DATE,  
    min_lookuptime         OUT DATE,  
    max_lookuptime         OUT DATE,  
    avg_lookuptime         OUT DATE,  
    min_ftstime            OUT DATE,  
    max_ftstime            OUT DATE,  
    avg_ftstime            OUT DATE);  
```

Parameters

Table 76-8  SEGMENT_HEAT_MAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_id</td>
<td>Tablespace containing the segment</td>
</tr>
</tbody>
</table>
### Table 76-8  (Cont.) SEGMENT_HEAT_MAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header_file</td>
<td>Segment header relative file number</td>
</tr>
<tr>
<td>header_block</td>
<td>Segment header block number</td>
</tr>
<tr>
<td>segment_objd</td>
<td>DATAOBJ of the segment</td>
</tr>
</tbody>
</table>

### Return Values

### Table 76-9  SEGMENT_HEAT_MAP Procedure Return Values (Output Parameters)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_writetime</td>
<td>Oldest write time for the segment</td>
</tr>
<tr>
<td>max_writetime</td>
<td>Latest write time for the segment</td>
</tr>
<tr>
<td>avg_writetime</td>
<td>Average write time for the segment</td>
</tr>
<tr>
<td>min_readtime</td>
<td>Oldest read time for the segment</td>
</tr>
<tr>
<td>max_readtime</td>
<td>Latest read time for the segment</td>
</tr>
<tr>
<td>avg_readtime</td>
<td>Average read time for the segment</td>
</tr>
<tr>
<td>min_LOOKuptime</td>
<td>Oldest index lookup time for the segment</td>
</tr>
<tr>
<td>max_LOOKuptime</td>
<td>Latest index lookup time for the segment</td>
</tr>
<tr>
<td>avg_LOOKuptime</td>
<td>Average index lookup time for the segment</td>
</tr>
<tr>
<td>min_ftstime</td>
<td>Oldest full table scan time for the segment</td>
</tr>
<tr>
<td>max_ftstime</td>
<td>Latest full table scan time for the segment</td>
</tr>
<tr>
<td>avg_ftstime</td>
<td>Average full table scan time for the segment</td>
</tr>
</tbody>
</table>

### 76.3.5 TABLESPACE_HEAT_MAP Function

This table function returns the minimum, maximum and average access times for all the segments in the tablespace.

#### Syntax

```sql
DBMS_HEAT_MAP.TABLESPACE_HEAT_MAP (  
    tablespace_name IN VARCHAR2)  
RETURN hm_tablespace_table PIPELINED;
```

#### Parameters

### Table 76-10  TABLESPACE_HEAT_MAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of the tablespace</td>
</tr>
</tbody>
</table>
## Return Values

Table 76-11  **TABLESPACE_HEAT_MAP Procedure Return Values (Output Parameters)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_count</td>
<td>Total number of segments in the tablespace</td>
</tr>
<tr>
<td>allocated_bytes</td>
<td>Space used by the segments in the tablespace</td>
</tr>
<tr>
<td>min_writetime</td>
<td>Oldest write time for the segment</td>
</tr>
<tr>
<td>max_writetime</td>
<td>Latest write time for the segment</td>
</tr>
<tr>
<td>avg_writetime</td>
<td>Average write time for the segment</td>
</tr>
<tr>
<td>min_readtime</td>
<td>Oldest read time for the segment</td>
</tr>
<tr>
<td>max_readtime</td>
<td>Latest read time for the segment</td>
</tr>
<tr>
<td>avg_writetime</td>
<td>Average write time for the segment</td>
</tr>
<tr>
<td>min_lookuptime</td>
<td>Oldest index lookup time for the segment</td>
</tr>
<tr>
<td>max_lookuptime</td>
<td>Latest index lookup time for the segment</td>
</tr>
<tr>
<td>avg_lookuptime</td>
<td>Average index lookup time for the segment</td>
</tr>
<tr>
<td>min_ftstime</td>
<td>Oldest full table scan time for the segment</td>
</tr>
<tr>
<td>max_ftstime</td>
<td>Latest full table scan time for the segment</td>
</tr>
<tr>
<td>avg_ftstime</td>
<td>Average full table scan time for the segment</td>
</tr>
</tbody>
</table>
DBMS_HIERARCHY

DBMS_HIERARCHY contains subprograms for validating the data in tables used by hierarchies and analytic views.

This chapter contains the following topics:

• DBMS_HIERARCHY Overview
• DBMS_HIERARCHY Security Model
• Summary of DBMS_HIERARCHY Subprograms

77.1 DBMS_HIERARCHY Overview

The DBMS_HIERARCHY package contains functions for validating that the contents of a database table are suitable for use by an analytic view or a hierarchy, a function for verifying the success of the validation, and a procedure for creating a table for logging validation operations.

Note:

Names specified by parameters of the DBMS_HIERARCHY subprograms are case-sensitive.

For information about using analytic views, see Oracle Database Data Warehousing Guide.

77.2 DBMS_HIERARCHY Security Model

Summarizes security considerations for the validation of analytic view and hierarchy objects.

All procedures in this package validate that the current user has the necessary privileges on the specified objects and return an error if those privileges are not found.

Note:

To ensure that the user has enough tablespace to log validation operations, do one of the following:

• GRANT UNLIMITED TABLESPACE TO username;
• ALTER USER username QUOTA size ON tablespace_name;

The following system privileges are required to use this package:
To validate objects in the user’s own schema:

- `CREATE TABLE privilege for CREATE_VALIDATE_LOG_TABLE` or to have `VALIDATE_ANALYTIC_VIEW` or `VALIDATE_HIERARCHY` automatically create a table
- `SELECT privilege` on the tables or views used by the analytic view or hierarchy
- `INSERT privilege` on the tables used by the attribute dimensions of the hierarchy or the fact table used by the analytic view

To validate objects in different schemas:

- `CREATE ANY TABLE privilege` for `CREATE_VALIDATE_LOG_TABLE` or to have the `VALIDATE_ANALYTIC_VIEW` or `VALIDATE_HIERARCHY` automatically create a table
- `INSERT ANY TABLE` privilege on the tables used by the attribute dimensions of the hierarchy or the fact table used by the analytic view

### 77.3 Summary of DBMS_HIERARCHY Subprograms

This table lists the `DBMS_HIERARCHY` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CREATE_VALIDATE_LOG_TABLE Procedure</strong></td>
<td>Creates a table that you can use for logging messages generated by the <code>VALIDATE_HIERARCHY</code> and <code>VALIDATE_ANALYTIC_VIEW</code> functions.</td>
</tr>
<tr>
<td><strong>VALIDATE_ANALYTIC_VIEW Function</strong></td>
<td>Validates that the data in a table is suitable for use by an analytic view.</td>
</tr>
<tr>
<td><strong>VALIDATE_CHECK_SUCCESS Function</strong></td>
<td>Indicates whether a prior call to <code>VALIDATE_HIERARCHY</code> or <code>VALIDATE_ANALYTIC_VIEW</code> was successful or produced validation errors.</td>
</tr>
<tr>
<td><strong>VALIDATE_HIERARCHY Function</strong></td>
<td>Validates that the data in a table is suitable for use by a hierarchy.</td>
</tr>
</tbody>
</table>

### 77.3.1 CREATE_VALIDATE_LOG_TABLE Procedure

This procedure creates a table that you can use for logging messages generated by the `VALIDATE_ANALYTIC_VIEW` or `VALIDATE_HIERARCHY` function, which validate data used by an analytic view or hierarchy.

The table that this procedure creates has the following structure.

<table>
<thead>
<tr>
<th>NAME</th>
<th>NULL?</th>
<th>DATATYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_NUMBER</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>ACTION_ORDER</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>OBJECT_OWNER</td>
<td>NOT NULL</td>
<td>VARCHAR2(128 BYTE)</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(128 BYTE)</td>
</tr>
<tr>
<td>ACTION</td>
<td>NOT NULL</td>
<td>VARCHAR2(10 BYTE)</td>
</tr>
<tr>
<td>TIME</td>
<td>NOT NULL</td>
<td>TIMESTAMP(6)</td>
</tr>
</tbody>
</table>
ERROR_NUMBER              NUMBER
ERROR_MESSAGE              VARCHAR2(4000)

Syntax

DBMS_HIERARCHY.CREATE_VALIDATE_LOG_TABLE (  
  table_name        IN  VARCHAR2,  
  owner_name        IN  VARCHAR2         DEFAULT NULL  
  IGNORE_IF_EXISTS  IN  PL/SQL BOOLEAN   DEFAULT FALSE);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table to create.</td>
</tr>
<tr>
<td>owner_name</td>
<td>The name of the schema in which to create the table. If owner_name is NULL, then the table is created in the current user’s schema.</td>
</tr>
<tr>
<td>IGNORE_IF_EXISTS</td>
<td>A Boolean that indicates whether to create the table if a table by the same name exists. If you specify a table, it must have the same structure as the table that this procedure creates.</td>
</tr>
</tbody>
</table>

Examples

Example 77-1    Creating a Validation Log Table

BEGIN  
  DBMS_HIERARCHY.CREATE_VALIDATE_LOG_TABLE (  
    'VAL_AV_HIERARCHY_LOG',  
    'AV_USER',  
    FALSE  
  );
END;
/

77.3.2 VALIDATE_ANALYTIC_VIEW Function

This function validates that the data in a table or view conforms to the logical constraints inherent in the definition of an analytic view.

Syntax

DBMS_HIERARCHY.VALIDATE_ANALYTIC_VIEW (  
  analytic_view_name         IN VARCHAR2   DEFAULT NULL,  
  analytic_view_owner_name   IN VARCHAR2   DEFAULT NULL,  
  log_table_name             IN VARCHAR2   DEFAULT NULL,  
  log_table_owner_name       IN VARCHAR2   DEFAULT NULL)  
  RETURN NUMBER;
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>analytic_view_name</td>
<td>The name of the analytic view to validate.</td>
</tr>
<tr>
<td>analytic_view_owner_name</td>
<td>The name of the owner of the schema that contains the analytic view.</td>
</tr>
<tr>
<td>log_table_name</td>
<td>The name of the validation log table in which to put the results of the validation operation.</td>
</tr>
<tr>
<td>log_table_owner_name</td>
<td>The name of the owner of the schema in which the validation log table exists or in which to create the table.</td>
</tr>
</tbody>
</table>

Returns

The number of the entry in the validation log table for the validation results.

Usage Notes

If the log_table_name parameter is NULL, then the VALIDATE_ANALYTIC_VIEW function creates a validation log table. The name of the table it creates is DBMS_HIERARCHY_LOG.

When the validation operation begins, a row is inserted into the log table with the action of START. When the operation completes, a row is inserted into the log table with the action of END. When an error is detected, a row is inserted into the log table with the action of ERROR, and the associated error_number and error_message columns are populated. All rows inserted into the validation log table include a log number and the time of the insert.

The VALIDATE_ANALYTIC_VIEW function verifies that the following conditions are true for each attribute dimension the analytic view is dimensioned by:

- The key values found in the fact table for the attribute dimension must exist in the star schema dimension table for that attribute dimension.
- The referenced attribute values for the attribute dimension must be unique across all rows of the star schema dimension table for that dimension.

Also, for every hierarchy in the analytic view, the function verifies that the following conditions are true:

- The primary key of a level determines a unique value for each attribute of the level.
- For each row of the table or view used by the attribute dimension of the hierarchy, the value for every level key column (including alternate keys) of a NOT NULL level is non-NULL.
- For each row of the table or view, either all level key columns and alternate key columns of a SKIP WHEN NULL level must be NULL or they must all be non-NULL. This verifies that the alternate level key is determined by the level key.
- For each group of rows that have the same alternate key column values for a level, the key column values must have the same column values. This verifies that the level key is determined by the alternate level key, which is required for an alternate key.
Examples

Example 77-2  Validating an Analytic View

DECLARE
    log_num NUMBER;
    obj_name VARCHAR2(8) := 'SALES_AV';
BEGIN
    log_num := DBMS_HIERARCHY.VALIDATE_ANALYTIC_VIEW(obj_name);
END;
/

77.3.3 VALIDATE_CHECK_SUCCESS Function

This function indicates whether a prior call to VALIDATE_HIERARCHY or VALIDATE_ANALYTIC_VIEW was successful or produced validation errors.

Syntax

DBMS_HIERARCHY.VALIDATE_CHECK_SUCCESS (  
    TOPOBJ_NAME IN VARCHAR2,  
    TOPOBJ_OWNER IN VARCHAR2,  
    LOG_NUMBER  IN VARCHAR2  
    LOG_TABLE_NAME IN VARCHAR2  
    LOG_TABLE_OWNER_NAME IN VARCHAR2  )  
RETURN VARCHAR2;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOPOBJ_NAME</td>
<td>The name of the hierarchy or analytic view.</td>
</tr>
<tr>
<td>TOPOBJ_OWNER</td>
<td>The owner of the hierarchy or analytic view.</td>
</tr>
<tr>
<td>LOG_NUMBER</td>
<td>The number of the log entry.</td>
</tr>
<tr>
<td>LOG_TABLE_NAME</td>
<td>The name of the log table.</td>
</tr>
<tr>
<td>LOG_TABLE_OWNER_NAME</td>
<td>The name of the schema in which the table exits.</td>
</tr>
</tbody>
</table>

Returns

A VARCHAR2 that is SUCCESS if no errors occurred or ERROR if errors did occur.

Examples

Example 77-3  Using VALIDATE_CHECK_SUCCESS

This example finds out whether the prior call to VALIDATE_ANALYTIC_VIEW encountered errors.

DECLARE
    log_num NUMBER;
    succ VARCHAR2(7);
    obj_name VARCHAR2(8) := 'SALES_AV';
BEGIN
log_num := dbms_hierarchy.validate_analytic_view(obj_name);
succ := dbms_hierarchy.validate_check_success(
    topobj_name => obj_name, log_number => log_num);
IF (succ != 'SUCCESS') THEN
    RAISE_APPLICATION_ERROR(
        num => -20000,
        msg => 'Validate failed!');
END IF;
END;
/

77.3.4 VALIDATE_HIERARCHY Function

This function validates that the data in a table or view conforms to the logical
constraints inherent in the definitions of an attribute dimension that uses the table or
view and a hierarchy that uses the attribute dimension.

Syntax

DBMS_HIERARCHY.VALIDATE_HIERARCHY (  
    hier_name                IN VARCHAR2,
    hier_owner_name          IN VARCHAR2 DEFAULT NULL,
    log_table_name           IN VARCHAR2 DEFAULT NULL,
    log_table_owner_name     IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hier_name</td>
<td>The name of the hierarchy to validate.</td>
</tr>
<tr>
<td>hier_owner_name</td>
<td>The name of the owner of the schema that contains the hierarchy.</td>
</tr>
<tr>
<td>log_table_name</td>
<td>The name of the validation log table in which to put the results of the validation operation.</td>
</tr>
<tr>
<td>log_table_owner_name</td>
<td>The name of the owner of the schema in which the validation log table exists or in which to create the table.</td>
</tr>
</tbody>
</table>

Returns

The number of the entry in the validation log table for the validation results.

Usage Notes

If the log_table_name parameter is NULL, then the VALIDATE_HIERARCHY function creates a validation log table. The name of the table it creates is DBMS_HIERARCHY_LOG.

When the validation operation begins, a row is inserted into the log table with the action of START. When the operation completes, a row is inserted into the log table with the action of END. When an error is detected, a row is inserted into the log table with the action of ERROR, and the associated error_number and error_message columns are populated. All rows inserted into the validation log table include a log number and the time of the insert.
The `VALIDATE_HIERARCHY` function verifies that the following conditions are true for the hierarchy:

- The primary key of a level determines a unique value for each attribute of the level.
- For each row of the table or view used by the attribute dimension of the hierarchy, the value for every level key column (including alternate keys) of a `NOT NULL` level is non-null.
- For each row of the table or view, either all level key columns and alternate key columns of a `SKIP WHEN NULL` level must be `NULL` or they must all be non-`NULL`. This verifies that the alternate level key is determined by the level key.
- For each group of rows that have the same alternate key column values for a level, the key column values must have the same column values. This verifies that the level key is determined by the alternate level key, which is required for an alternate key.

Examples

Example 77-4  Validating a Hierarchy and Specifying a Table Name

This example validates the `PRODUCT_HIER` hierarchy and specifies that the results be inserted in the table named `VAL_AV_HIERARCHY_LOG`. The owner of the hierarchy and of the schema that contains the table is `AV_USER`.

```sql
-- Create a log table.
BEGIN
    DBMS_HIERARCHY.CREATE_VALIDATE_LOG_TABLE (
        'VAL_AV_HIERARCHY_LOG',
        'AV_USER',
        FALSE
    );
END;
/

-- Validate the hierarchy.
DECLARE
    log_num NUMBER;
    obj_name VARCHAR2(12) := 'PRODUCT_HIER';
    table_name  VARCHAR2(28) := 'VAL_AV_HIERARCHY_LOG';
BEGIN
    log_num := DBMS_HIERARCHY.VALIDATE_HIERARCHY(obj_name, 'AV_USER', table_name);
END;
/

Query the log table.

```

```
SELECT LOG_NUMBER, ACTION, OBJECT_NAME, ERROR_NUMBER, ERROR_MESSAGE
FROM AV_USER.VAL_AV_HIERARCHY_LOG
WHERE OBJECT_NAME = 'PRODUCT_HIER';

<table>
<thead>
<tr>
<th>LOG_NUMBER</th>
<th>ACTION</th>
<th>OBJECT_NAME</th>
<th>ERROR_NUMBER</th>
<th>ERROR_MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>START</td>
<td>PRODUCT_HIER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>END</td>
<td>PRODUCT_HIER</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

Chapter 77

Summary of DBMS_HIERARCHY Subprograms

77-7
Example 77-5  Validating a Hierarchy Without Specifying a Table Name

This example shows that if you do not specify a validation log table, then the
VALIDATE_HIERARCHY function creates one named DBMS_HIERARCHY_LOG.

```
DECLARE
    log_num NUMBER;
    obj_name VARCHAR2(12) := 'PRODUCT_HIER';
BEGIN
    log_num := DBMS_HIERARCHY.VALIDATE_HIERARCHY(obj_name);
END;
```

Query the log table.

```
SELECT LOG_NUMBER, ACTION, OBJECT_NAME, ERROR_NUMBER, ERROR_MESSAGE
FROM DBMS_HIERARCHY_LOG
WHERE OBJECT_NAME = 'PRODUCT_HIER';
```

<table>
<thead>
<tr>
<th>LOG_NUMBER</th>
<th>ACTION</th>
<th>OBJECT_NAME</th>
<th>ERROR_NUMBER</th>
<th>ERROR_MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>START</td>
<td>PRODUCT_HIER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>END</td>
<td>PRODUCT_HIER</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DBMS_HM

This package contains constants and procedure declarations for health check management. Health Monitor provides facilities to run a check store and retrieve the reports through DBMS_HM package.

This chapter contains the following topics:

• Security Model
• Summary of DBMS_HM Subprograms

See Also:

Oracle Database Administrator's Guide for more information about "Health Monitor"

78.1 DBMS_HM Security Model

Users must have EXECUTE privilege to run the procedures of DBMS_HM package.

78.2 Summary of DBMS_HM Subprograms

This table lists the DBMS_CONNECTION_POOL subprograms and briefly describes them.

Table 78-1 DBMS_HM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_RUN_REPORT Function</td>
<td>Returns the report for the specified checker run</td>
</tr>
<tr>
<td>RUN_CHECK Procedure</td>
<td>Runs the specified checker with the given arguments</td>
</tr>
</tbody>
</table>

78.2.1 GET_RUN_REPORT Function

This function returns the report for the specified checker run.

Syntax

```sql
DBMS_HM.GET_RUN_REPORT ( run_name IN VARCHAR2, type IN VARCHAR2 := 'TEXT', level IN VARCHAR2 := 'BASIC',)
RETURN CLOB;
```
78.2.2 RUN_CHECK Procedure

This procedure runs the specified checker with the specified arguments.

You can specify a name for the run, the inputs needed and maximum time-out for the run. The run report will be maintained persistently in the database.

Syntax

DBMS_HM.RUN_CHECK (  
    check_name    IN  VARCHAR2,  
    run_name      IN  VARCHAR2 := NULL,  
    timeout       IN  NUMBER := NULL,  
    input_params  IN  VARCHAR2 := NULL);  

Parameters

Table 78-3  RUN_CHECK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_name</td>
<td>Name of the check to be invoked. Check names and their parameters can be accessed from the V$HM_CHECK and V$HM_CHECK_PARAM views. Users can run all checks which are not internal in nature: SELECT name FROM V$HM_CHECK WHERE INTERNAL_CHECK = 'N' retrieves the list of checks that can be run manually by users.</td>
</tr>
<tr>
<td>run_name</td>
<td>Name with which external users can uniquely identify this check's run. If NULL value is passed, then HM creates a unique name and associates with this check's run.</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum amount of time (in units of seconds), this checker run is allowed to run. HM will interrupt the run, if it the specified time elapses for the run. If NULL value is passed, HM doesn't impose any timeout limits on the run.</td>
</tr>
</tbody>
</table>
### Table 78-3  (Cont.) RUN_CHECK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_params</td>
<td>Input string: which consists of name, value pairs delimited by a special character <code>;</code>. Example (‘Data Block Integrity Check’ invocation may take following type of input parameters. 'BLC_DF_NUM=1;BLC_BL_NUM=23456'）</td>
<td></td>
</tr>
</tbody>
</table>

Input parameters `BLC_DF_NUM` and `BLC.BL_NUM` have values '1' and '23456' respectively.

Every check will have well defined set of inputs associated with it. These Input parameters, their types, default values and descriptions can be obtained using `V$HM_CHECK_PARAM` view.

Example: The following query gets the list of parameters, their default values and descriptions for a ‘Data Block Integrity Check’

```sql
SELECT a.* FROM v$hm_check_param a, v$hm_check b
WHERE a.check_id = b.id
AND b.name = 'Data Block Integrity Check';
```
79

DBMS_HPROF

The DBMS_HPROF package provides an interface for profiling the execution of PL/SQL applications. It provides services for collecting the hierarchical profiler data, analyzing the raw profiler output and profiling information generation.

This chapter contains the following topic:

• Summary of DBMS_HPROF Subprograms

See Also:

Oracle Database Development Guide for more information about the "PL/SQL Hierarchical Profiler"

79.1 Summary of DBMS_HPROF Subprograms

This table lists and briefly describes the DBMS_HPROF package subprograms.

Table 79-1 DBMS_HPROF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZE Function</td>
<td>Analyzes the raw profiler output and produces hierarchical profiler information in database tables.</td>
</tr>
<tr>
<td>CREATE_TABLES Procedure</td>
<td>Creates the hierarchical profiler database tables and data structures in the user's session.</td>
</tr>
<tr>
<td>START_PROFILING Procedure</td>
<td>Starts hierarchical profiler data collection in the user's session.</td>
</tr>
<tr>
<td>STOP_PROFILING Procedure</td>
<td>Stops profiler data collection in the user's session.</td>
</tr>
</tbody>
</table>

79.1.1 ANALYZE Function

This function analyzes the raw profiler output and produces hierarchical profiler information in database tables or generates out-of-the-box HTML reports.

Syntax

DBMS_HPROF.ANALYZE (
    trace_id IN NUMBER,
    summary_mode IN BOOLEAN DEFAULT FALSE,
    trace IN VARCHAR2 DEFAULT NULL,
    skip IN PLS_INTEGER DEFAULT 0,
    collect IN PLS_INTEGER DEFAULT NULL,
    run_comment IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
DBMS_HPROF.ANALYZE (  
  trace_id         IN NUMBER,  
  report_clob      OUT CLOB,  
  trace            IN VARCHAR2 DEFAULT NULL,  
  skip             IN PLS_INTEGER DEFAULT 0,  
  collect          IN PLS_INTEGER DEFAULT NULL);  

Parameters  

Table 79-2   ANALYZE Function Parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace_id</td>
<td>The trace_id of the raw profiler data entry in the raw profiler data table (dbmshp_trace_data).</td>
</tr>
<tr>
<td>summary_mode</td>
<td>By default (that is, when summary_mode is FALSE), a detailed analysis is done. When summary_mode is TRUE, only top-level summary information is generated into the database table.</td>
</tr>
<tr>
<td>report_clob</td>
<td>The analyzed HTML report.</td>
</tr>
<tr>
<td>trace</td>
<td>Analyzes only the subtrees rooted at the specified trace entry. By default (when trace is NULL), the analysis/reporting is generated for the entire run. The trace entry must be specified in a special quoted qualified format. For example, &quot;HR.&quot;.&quot;PKG&quot;.&quot;FOO&quot; or &quot;&quot;.&quot;.&quot;__plsql_vm&quot;. If multiple overloads exist for the specified name, all of them will be analyzed.</td>
</tr>
<tr>
<td>skip</td>
<td>Used only when trace is specified. Analyze only the subtrees rooted at the specified trace, but ignore the first skip invocations to trace. The default value for skip is 0.</td>
</tr>
<tr>
<td>collect</td>
<td>Used only when trace is specified. Analyze collect number of invocations of traces (starting from skip+1'th invocation). By default, only 1 invocation is collected.</td>
</tr>
<tr>
<td>run_comment</td>
<td>User-provided comment for this run.</td>
</tr>
</tbody>
</table>

Return Values  

A unique run identifier for this run of the analyzer. This can then be used to look up the results corresponding to this run from the hierarchical profiler tables.  

Usage Notes  

- Use the DBMS_HPROF.CREATE_TABLES subprogram to create the hierarchical profiler database tables and other data structures required for persistently storing the results of analyzing the raw profiler data.  
- Calling the DBMS_HPROF.CREATE_TABLES with default value (FALSE) will raise error if table already exists.  
- Use DBMS_HPROF.CREATE_TABLES(TRUE) to drop any previously created hierarchical profiler tables.
• Use the `DBMS_HPROF.CREATE_TABLES` to drop any previously created hierarchical profiler tables. By default, `force_it` is `FALSE`; therefore, to drop any previously created hierarchical profiler tables you must set the value of `force_it` to `TRUE`.

• If `trace_id` entry is `NULL`, error is raised.

• If `trace_id` entry in the raw profiler data table does not exist, error is raised.

• If raw data of the `trace_id` entry in the raw profiler data table is `NULL` or is zero size, error is raised.

Examples

The following snippet installs the hierarchical profiler tables in HR schema.

```
connect HR/<password>;
```

The following example analyzes and generates HTML CLOB report from a raw profiler data table.

```
DECLARE
    reportclob clob;
    trace_id number;
BEGIN
    -- create raw profiler data and analysis tables
    -- force_it =>TRUE will dropped the tables if table exists
    DBMS_HPROF.CREATE_TABLES(force_it =>TRUE);

    -- Start profiling
    -- Write raw profiler data in raw profiler data table
    trace_id := DBMS_HPROF.START_PROFILING;

    -- Run procedure to be profiled
    test;

    -- Stop profiling
    DBMS_HPROF.STOP_PROFILING;

    -- analyzes trace_id entry in raw profiler data table and produce
    -- analyzed HTML report in reportclob
    DBMS_HPROF.ANALYZE(trace_id, reportclob);
END;
/
```

79.1.2 CREATE_TABLES Procedure

Creates the hierarchical profiler database tables and data structures in the user's session.

Syntax

```
DBMS_HPROF.CREATE_TABLES (force_it IN BOOLEAN DEFAULT FALSE);
```
Parameters

Table 79-3  CREATE_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force_it</td>
<td>If FALSE and DBMS_HPROF tables are present, then an HPROF error is raised. If TRUE, then the procedure creates tables. If the tables already exist, then they are dropped and new tables are created.</td>
</tr>
</tbody>
</table>

Note:

Users need not use the dbmshptab.sql script located in the rdbms/admin directory to create the hierarchical profiler database tables and data structures anymore.

The dbmshptab.sql script is deprecated starting in Oracle Database 18c.

79.1.3 START_PROFILING Procedure

This procedure starts hierarchical profiler data collection in the user's session.

Syntax

```
DBMS_HPROF.START_PROFILING(
    max_depth    IN PLS_INTEGER DEFAULT NULL,
    sqlmonitor   IN BOOLEAN DEFAULT TRUE,
    run_comment  IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Parameters

Table 79-4  START_PROFILING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_depth</td>
<td>By default (that is, when max_depth value is NULL) profile information is gathered for all functions irrespective of their call depth. When a non-NULL value is specified for max_depth, the profiler collects data only for functions up to a call depth level of max_depth.</td>
</tr>
<tr>
<td>sqlmonitor</td>
<td>Generates a real-time monitoring report for a profiler run when the profiler run ends. The default value is TRUE.</td>
</tr>
<tr>
<td>run_comment</td>
<td>User provided comment for the profiler data collection run.</td>
</tr>
</tbody>
</table>
Return Values

Unique run identifier for this profiler run. This can then be used to look up the results corresponding to this run from the hierarchical profiler tables.

Usage Notes

- Even though the profiler does not individually track functions at depth greater than max_depth, the time spent in such functions is charged to the ancestor function at depth max_depth.
- Raw profiler data is generated in the raw profiler data table with an unique trace_id.
- The unique trace_id is used to manage the raw profiler output stored in the raw profiler data table.

79.1.4 STOP_PROFILING Procedure

This procedure stops profiler data collection in the user's session. This subprogram also has the side effect of flushing data collected so far in the session, and it signals the end of a run. When the STOP_PROFILING procedure returns CLOB, it contains the Real-Time Monitoring report for the profiler run.

Syntax

DBMS_HPROF.STOP_PROFILING;

DBMS_HPROF.STOP_PROFILING
  RETURN CLOB;

Examples

Profiling with raw profiler data table

DECLARE
  analyze_runid number;
  trace_id number;
BEGIN
  -- create raw profiler data and analysis tables
  -- call create_tables with force_it =>FALSE (default) when
  -- raw profiler data and analysis tables do not exist already
  DBMS_HPROF.CREATE_TABLES;
  -- Start profiling
  -- Write raw profiler data in raw profiler data table
  trace_id := DBMS_HPROF.START_PROFILING;
  -- Run the procedure to be profiled
  test;
  -- Stop profiling
  DBMS_HPROF.STOP_PROFILING;
  -- analyzes trace_id entry in raw profiler data table and writes
  -- hierarchical profiler information in hprof's analysis tables
  analyze_runid := DBMS_HPROF.ANALYZE(trace_id);
END;
/
The DBMS_HS_PARALLEL PL/SQL package enables parallel processing for heterogeneous targets access. This package is designed to improve performance when retrieving data from a large foreign table.

This chapter discusses the following topics:

- Using DBMS_HS_PARALLEL
- Summary of DBMS_HS_PARALLEL Subprograms

80.1 Using the DBMS_HS_PARALLEL Subprogram

DBMS_HS_PARALLEL is compiled with the authorization ID of CURRENT_USER, which uses invoker's rights. In other words, all procedures in this package are executed with the privileges of the calling user.

80.2 Summary of DBMS_HS_PARALLEL Subprograms

This table lists the DBMS_HS_PARALLEL subprograms in alphabetic order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_OR_REPLACE_VIEW</td>
<td>Creates (or replaces) a read-only view to be referenced for retrieving the data from a remote table in parallel.</td>
</tr>
<tr>
<td>CREATE_TABLE_TEMPLATE</td>
<td>Writes out a CREATE TABLE template based on information gathered from the remote table. You can use the information to add any optimal Oracle CREATE TABLE clauses.</td>
</tr>
<tr>
<td>DROP_VIEW</td>
<td>Drops the view and internal objects created by the CREATE_OR_REPLACE_VIEW procedure. If the view has not already been created by the CREATE_OR_REPLACE_VIEW procedure, an error message is returned.</td>
</tr>
<tr>
<td>LOAD_TABLE</td>
<td>Loads the data from a remote table to a local Oracle table in parallel. If the local Oracle table does not already exist, it is created automatically.</td>
</tr>
</tbody>
</table>

80.2.1 CREATE_OR_REPLACE_VIEW Procedure

This procedure creates (or replaces) a read-only view to be referenced for retrieving the data from a remote table in parallel.

Syntax

CREATE_OR_REPLACE_VIEW (remote_table, database_link, oracle_view, parallel_degree)
### Parameters

**Table 80-2  CREATE_OR_REPLACE_VIEW Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_table</td>
<td>The name of the remote database table. It is specified as <code>[remote_schema_name.]remote_table_name</code>.</td>
</tr>
<tr>
<td>database_link</td>
<td>The remote database link name. The call can only be applied to a heterogeneous services database link.</td>
</tr>
<tr>
<td>oracle_view</td>
<td>The name of the Oracle view. It is specified as <code>[schema_name.]oracle_view_name</code>. The default schema name is the current user. If the oracle_view parameter is not specified, the remote table name will be used as the view name.</td>
</tr>
<tr>
<td>parallel_degree</td>
<td>The number of parallel processes for the operation is computed based on the range-partition number if applicable, or the number of CPUs. The range of values is 2 to 16.</td>
</tr>
</tbody>
</table>

### Usage Notes

- The specified Oracle view is created and future reference of this view utilizes internal database objects for parallel retrieval of remote non-Oracle table data. If the Oracle view already exists, the following Oracle error message is raised:

  ORA-00955: name is already used by an existing object

- This view is created as a read-only view. If you attempt to insert and update the view, the following Oracle error message is raised:

  ORA-01733: virtual column not allowed here

- If the remote table or the database link does not exist, one of the following Oracle error messages is raised:

  ORA-00942: table or view does not exist or
  ORA-02019: connection description for remote database not found

- **You need the CREATE VIEW, CREATE TABLE, CREATE TYPE, CREATE PACKAGE, and CREATE FUNCTION privileges to execute the CREATE_OR_REPLACE_VIEW procedure.**

- If you encounter either of the following Oracle error messages, increase the PROCESSES and SESSIONS parameter in the Oracle initialization parameter file:

  ORA-12801: error signaled in parallel query server P003 or
  ORA-00018: maximum number of session exceeded

- **Because the CREATE_OR_REPLACE_VIEW procedure creates some internal objects, use the DROP_VIEW procedure to drop the view and the internal objects. The SQL DROP VIEW statement only drops the view and not the internal objects.**
80.2.2 CREATE_TABLE_TEMPLATE Procedure

This procedure writes out a `CREATE TABLE` template based on information gathered from the remote table. You can use the information to add any optimal Oracle `CREATE TABLE` clauses.

Syntax

```sql
CREATE_TABLE_TEMPLATE (remote_table, database_link, oracle_table, create_table_template_string)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_table</td>
<td>The name of the remote database table. It is specified as <code>[remote_schema_name.]remote_table_name</code>.</td>
</tr>
<tr>
<td>database_link</td>
<td>The remote database link name. The call can only be applied to a heterogeneous services database link.</td>
</tr>
<tr>
<td>oracle_table</td>
<td>The name of the local Oracle table the data will be loaded into. It is specified as <code>[schema_name.]oracle_table_name</code>. The default schema name is the current user. If the oracle_table parameter is not specified, the remote table name will be used as the local Oracle name.</td>
</tr>
<tr>
<td>create_table_template_string</td>
<td>Contains the Oracle CREATE TABLE SQL template when the procedure is returned.</td>
</tr>
</tbody>
</table>

80.2.3 DROP_VIEW Procedure

This procedure drops the view and internal objects created by the `CREATE_OR_REPLACE_VIEW` procedure. If the view has not already been created by the `CREATE_OR_REPLACE_VIEW` procedure, an error message is returned.

Syntax

```
DROP_VIEW (oracle_view)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle_view</td>
<td>The name of the Oracle view created by the <code>CREATE_OR_REPLACE_VIEW</code> procedure. If the view has not been created by the <code>CREATE_OR_REPLACE_VIEW</code> procedure, an error is returned.</td>
</tr>
</tbody>
</table>
80.2.4 LOAD_TABLE Procedure

This procedure loads the data from a remote table to a local Oracle table in parallel. If the local Oracle table does not already exist, it is created automatically.

Syntax

LOAD_TABLE (remote_table, database_link, oracle_table, truncate, parallel_degree, row_count)

Parameters

Table 80-5  LOAD_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_table</td>
<td>The name of the remote database table. It is specified as [remote_schema_name.]remote_table_name</td>
</tr>
<tr>
<td>database_link</td>
<td>The remote database link name. The call can only be applied to a heterogeneous services database link.</td>
</tr>
<tr>
<td>oracle_table</td>
<td>The name of the local Oracle table the data will be loaded into. It is specified as [schema_name.]oracle_table_name. The default schema name is the current user. If the oracle_table parameter is not specified, the remote table name will be used as the local Oracle name.</td>
</tr>
<tr>
<td>truncate</td>
<td>Determines whether the Oracle table is truncated before the data is loaded. The value is either TRUE or FALSE. The default value is TRUE which means the Oracle table is truncated first. When set to FALSE, the Oracle table will not be truncated before the data is loaded.</td>
</tr>
<tr>
<td>parallel_degree</td>
<td>The number of parallel processes for the operation is computed based on the range-partition number if applicable, or the number of CPUs. The range of values is 2 to 16.</td>
</tr>
<tr>
<td>row_count</td>
<td>Contains the number of rows just added with the load table operation.</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure only loads the remote table data into Oracle local table. It does not create a key, index, constraints or any other dependencies such as triggers. It is recommended that you create these dependencies after the table data is loaded as performance will improve greatly. You will need to decide whether to create the dependencies before or after the data is loaded based on your knowledge of the remote table data and dependencies.
- If the local table does not exist, the LOAD_TABLE procedure creates a simple (non-partitioned) local table based on the exact column matching of the remote table after which the data is inserted into the local table.
• If the remote table or the database link does not exist, an error message is returned.

• If the local table is incompatible with the remote table, an error message is returned.

• You need the `CREATE TABLE`, `CREATE TYPE`, `CREATE PACKAGE`, and `CREATE FUNCTION` privileges to execute the `LOAD_TABLE` procedure.

• If you encounter either of the following Oracle error messages, increase the `PROCESSES` and `SESSIONS` parameter in Oracle initialization parameter file:
  ORA-12801: error signaled in parallel query server P003
  ORA-00018: maximum number of session exceeded

• One of the following is required for parallel processing:
  – The remote table is range partitioned.
  – Histogram information for a numeric column is available.
  – There is a numeric index or primary key.

• To drop the local table, use the `DROP TABLE` SQL statement.
The DBMS_HS_PASSTHROUGH PL/SQL package allows you to send a statement directly to a non-Oracle system without being interpreted by the Oracle server. This can be useful if the non-Oracle system allows operations in statements for which there is no equivalent in Oracle.

This chapter discusses the following topics:

- DBMS_HS_PASSTHROUGH Overview
- DBMS_HS_PASSTHROUGH Operational Notes
- Summary of DBMS_HS_PASSTHROUGH Subprograms

81.1 DBMS_HS_PASSTHROUGH Overview

You can execute passthrough SQL statements directly at the non-Oracle system using the PL/SQL package DBMS_HS_PASSTHROUGH. Any statement executed with this package is executed in the same transaction as standard SQL statements.

81.2 DBMS_HS_PASSTHROUGH Operational Notes

The DBMS_HS_PASSTHROUGH package is a virtual package. It conceptually resides at the non-Oracle system. In reality, however, calls to this package are intercepted by Heterogeneous Services and mapped to one or more Heterogeneous Services calls. The driver, in turn, maps these Heterogeneous Services calls to the API of the non-Oracle system. The client application should invoke the procedures in the package through a database link in exactly the same way as it would invoke a non-Oracle system stored procedure. The special processing done by Heterogeneous Services is transparent to the user.

81.3 Summary of DBMS_HS_PASSTHROUGH Subprograms

This table lists the DBMS_HS_PASSTHROUGH subprograms and briefly describes them.

See Also:
Oracle Database Heterogeneous Connectivity User's Guide for more information about this package
Table 81-1  DBMS_HS_PASSTHROUGH Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIND_INOUT_VARIABLE Procedure</td>
<td>Binds IN OUT bind variables</td>
</tr>
<tr>
<td>BIND_INOUT_VARIABLE_RAW Procedure</td>
<td>Binds IN OUT bind variables of datatype RAW</td>
</tr>
<tr>
<td>BIND_OUT_VARIABLE Procedure</td>
<td>Binds an OUT variable with a PL/SQL program variable</td>
</tr>
<tr>
<td>BIND_OUT_VARIABLE_RAW Procedure</td>
<td>Binds an OUT variable of datatype RAW with a PL/SQL program variable</td>
</tr>
<tr>
<td>BIND_VARIABLE Procedure</td>
<td>Binds an IN variable positionally with a PL/SQL program variable</td>
</tr>
<tr>
<td>BIND_VARIABLE_RAW Procedure</td>
<td>Binds IN variables of type RAW</td>
</tr>
<tr>
<td>CLOSE_CURSOR Procedure</td>
<td>Closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system</td>
</tr>
<tr>
<td>EXECUTE_IMMEDIATE Procedure</td>
<td>Runs a (non-SELECT) SQL statement immediately, without bind variables</td>
</tr>
<tr>
<td>EXECUTE_NON_QUERY Function</td>
<td>Runs a (non-SELECT) SQL statement</td>
</tr>
<tr>
<td>FETCH_ROW Function</td>
<td>Fetches columns from a query</td>
</tr>
<tr>
<td>GET_VALUE Procedure</td>
<td>Retrieves column value from SELECT statement, or retrieves OUT bind parameters</td>
</tr>
<tr>
<td>GET_VALUE_RAW Procedure</td>
<td>Similar to GET_VALUE, but for datatype RAW</td>
</tr>
<tr>
<td>OPEN_CURSOR Function</td>
<td>Opens a cursor for running a passthrough SQL statement at the non-Oracle system</td>
</tr>
<tr>
<td>PARSE Procedure</td>
<td>Parses SQL statement at non-Oracle system</td>
</tr>
</tbody>
</table>

81.3.1 BIND_INOUT_VARIABLE Procedure

This procedure binds IN OUT bind variables.

Syntax

```sql
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (  
c      IN  BINARY_INTEGER NOT NULL,  
p      IN  BINARY_INTEGER NULL,  
v      OUT <dty>,  
n      IN  VARCHAR2);
```

<dty> is either DATE, NUMBER, or VARCHAR2.

See Also:

For binding OUT variables of datatype RAW, see BIND_OUT_VARIABLE_RAW Procedure.
Pragmas
Purity level defined : WNDS, RNDS

Parameters

Table 81-2   BIND_INOUT_VARIABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>p</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
</tbody>
</table>
| v         | This value is used for two purposes:  
               - To provide the IN value before the SQL statement is run.  
               - To determine the size of the out value.  
| n         | (Optional) Name of the bind variable.  
               For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required. |

Exceptions

Table 81-3   BIND_INOUT_VARIABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.2 BIND_INOUT_VARIABLE_RAW Procedure

This procedure binds IN OUT bind variables of datatype RAW.

Syntax

```sql
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW (  
    c IN   BINARY_INTEGER NOT NULL,  
    p IN   BINARY_INTEGER NOT NULL,  
    v IN OUT RAW,  
    n IN   VARCHAR2);
```

Pragmas
Purity level defined : WNDS, RNDS
### Parameters

#### Table 81-4  BIND_INOUT_VARIABLE_RAW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>c</strong></td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td><strong>p</strong></td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
</tbody>
</table>
| **v** | This value is used for two purposes:  
- To provide the IN value before the SQL statement is run.  
- To determine the size of the out value. |
| **n** | (Optional) Name the bind variable.  
For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable `ename` is 1, the name is `ename`.  
This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required. |

#### Exceptions

#### Table 81-5  BIND_INOUT_VARIABLE_RAW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

### 81.3.3 BIND_OUT_VARIABLE Procedure

This procedure binds an **out** variable with a PL/SQL program variable.

#### Syntax

```sql
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (  
c IN BINARY_INTEGER NOT NULL,  
p IN BINARY_INTEGER NULL,  
v OUT <dty>,  
n IN VARCHAR2);  
```

<dty> is either DATE, NUMBER, or VARCHAR2.
See Also:

For binding OUT variables of datatype RAW, see BIND_OUT_VARIABLE_RAW Procedure.

Pragmas

Purity level defined: WNDS, RNDS

Parameters

Table 81-6  BIND_OUT_VARIABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>p</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>v</td>
<td>Variable in which the OUT bind variable stores its value. The package remembers only the &quot;size&quot; of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE.</td>
</tr>
<tr>
<td>n</td>
<td>(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

Exceptions

Table 81-7  BIND_OUT_VARIABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.4 BIND_OUT_VARIABLE_RAW Procedure

This procedure binds an OUT variable of datatype RAW with a PL/SQL program variable.

Syntax

```sql
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW (  
  c IN BINARY_INTEGER NOT NULL,  
  p IN BINARY_INTEGER NOT NULL,  
```
v OUT RAW,
n IN VARCHAR2);

**Pragmas**

Purity level defined: WNDS, RNDS

**Parameters**

**Table 81-8  BIND_OUT_VARIABLE_RAW Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>p</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>v</td>
<td>Variable in which the OUT bind variable stores its value. The package remembers only the &quot;size&quot; of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE_RAW.</td>
</tr>
<tr>
<td>n</td>
<td>(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable ename is 1, the name is ename. This parameter can be used if the non-Oracle system supports &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

**Exceptions**

**Table 81-9  BIND_OUT_VARIABLE_RAW Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

**81.3.5 BIND_VARIABLE Procedure**

This procedure binds an IN variable positionally with a PL/SQL program variable.

**Syntax**

```sql
DBMS_HS_PASSTHROUGH.BIND_VARIABLE (  
    c IN BINARY_INTEGER NOT NULL,  
    p IN BINARY_INTEGER NOT NULL,  
    v IN <dty>,  
    n IN VARCHAR2);
```
is either DATE, NUMBER, or VARCHAR2.

See Also:
To bind RAW variables use BIND_VARIABLE_RAW Procedure.

Pragmas
Purity level defined: WNDS, RNDS

Parameters

Table 81-10  BIND_VARIABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>p</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>v</td>
<td>Value that must be passed to the bind variable name.</td>
</tr>
<tr>
<td>n</td>
<td>(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable ename is 1, the name is ename. This parameter can be used if the non-Oracle system supports &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

Exceptions

Table 81-11  BIND_VARIABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.6 BIND_VARIABLE_RAW Procedure

This procedure binds IN variables of type RAW.

Syntax

```sql
DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW (  
  c IN BINARY_INTEGER NOT NULL,  
  p IN BINARY_INTEGER NOT NULL,  
  v IN RAW,  
  n IN VARCHAR2);
```
Pragmas

Purity level defined: WNDS, RNDS

Parameters

Table 81-12  BIND_VARIABLE_RAW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>p</td>
<td>Position of the bind variable in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>v</td>
<td>Value that must be passed to the bind variable.</td>
</tr>
<tr>
<td>n</td>
<td>(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable ename is 1, the name is ename. This parameter can be used if the non-Oracle system supports &quot;named binds&quot; instead of positional binds. Passing the position is still required.</td>
</tr>
</tbody>
</table>

Exceptions

Table 81-13  BIND_VARIABLE_RAW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.7 CLOSE_CURSOR Procedure

This function closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system. If the cursor was not open, then the operation is a “no operation”.

Syntax

```
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (  
    c IN BINARY_INTEGER NOT NULL);  
```

Pragmas

Purity level defined: WNDS, RNDS
Parameters

Table 81-14  CLOSE_CURSOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor to be released.</td>
</tr>
</tbody>
</table>

Exceptions

Table 81-15  CLOSE_CURSOR Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.8 EXECUTE_IMMEDIATE Procedure

This function runs a SQL statement immediately. Any valid SQL command except SELECT can be run immediately.

The statement must not contain any bind variables. The statement is passed in as a VARCHAR2 in the argument. Internally the SQL statement is run using the PASSTHROUGH SQL protocol sequence of OPEN_CURSOR, PARSE, EXECUTE NON_QUERY, CLOSE_CURSOR.

Syntax

```sql
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE (
    s IN VARCHAR2 NOT NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 81-16  EXECUTE_IMMEDIATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>VARCHAR2 variable with the statement to be executed immediately.</td>
</tr>
</tbody>
</table>

Return Values

The number of rows affected by the execution of the SQL statement.

Exceptions

Table 81-17  EXECUTE_IMMEDIATE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28551</td>
<td>SQL statement is invalid.</td>
</tr>
<tr>
<td>ORA-28554</td>
<td>Max open cursors.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>
81.3.9 EXECUTE_NON_QUERY Function

This function runs a SQL statement. The SQL statement cannot be a SELECT statement. A cursor has to be open and the SQL statement has to be parsed before the SQL statement can be run.

Syntax

```sql
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY (
    c IN BINARY_INTEGER NOT NULL
) RETURN BINARY_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
</tbody>
</table>

Return Values

The number of rows affected by the SQL statement in the non-Oracle system

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.10 FETCH_ROW Function

This function fetches rows from a result set.

The result set is defined with a SQL SELECT statement. When there are no more rows to be fetched, the exception NO_DATA_FOUND is raised. Before the rows can be fetched, a cursor has to be opened, and the SQL statement has to be parsed.

Syntax

```sql
DBMS_HS_PASSTHROUGH.FETCH_ROW (
    c IN BINARY_INTEGER NOT NULL,
    f IN BOOLEAN
) RETURN BINARY_INTEGER;
```
Pragmas

Purity level defined : WNDS

Parameters

Table 81-20  FETCH_ROW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>first</td>
<td>(Optional) Reexecutes SELECT statement. Possible values: - TRUE: reexecute SELECT statement. - FALSE: fetch the next row, or if run for the first time, then execute and fetch rows (default).</td>
</tr>
</tbody>
</table>

Return Values

The returns the number of rows fetched. The function returns "0" if the last row was already fetched.

Exceptions

Table 81-21  FETCH_ROW Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.11 GET_VALUE Procedure

This procedure has two purposes: it retrieves the select list items of SELECT statements after a row has been fetched, and it retrieves the OUT bind values, after the SQL statement has been run.

Syntax

```sql
DBMS_HS_PASSTHROUGH.GET_VALUE (
  c      IN  BINARY_INTEGER NOT NULL,
  p    IN  BINARY_INTEGER NOT NULL,
  v    OUT <dty>);
```

<dty> is either DATE, NUMBER, or VARCHAR2.
See Also:
For retrieving values of datatype RAW, see GET_VALUE_RAW Procedure.

Pragmas

Purity level defined : WNDS

Parameters

Table 81-22 GET_VALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>p</td>
<td>Position of the bind variable or select list item in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>v</td>
<td>Variable in which the OUT bind variable or select list item stores its value.</td>
</tr>
</tbody>
</table>

Exceptions

Table 81-23 GET_VALUE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-1403</td>
<td>Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (that is, FETCH_ROW returned &quot;0&quot;).</td>
</tr>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>

81.3.12 GET_VALUE_RAW Procedure

This procedure is similar to GET_VALUE, but for datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (  
  c IN BINARY_INTEGER NOT NULL,  
  p IN BINARY_INTEGER NOT NULL,  
  v OUT RAW);
```

Pragmas

Purity level defined : WNDS
Parameters

Table 81-24  GET_VALUE_RAW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.</td>
</tr>
<tr>
<td>p</td>
<td>Position of the bind variable or select list item in the SQL statement: Starts at 1.</td>
</tr>
<tr>
<td>v</td>
<td>Variable in which the <strong>OUT</strong> bind variable or select list item stores its value.</td>
</tr>
</tbody>
</table>

Exceptions

Table 81-25  GET_VALUE_RAW Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-1403</td>
<td>Returns <strong>NO_DATA_FOUND</strong> exception when running the <strong>GET_VALUE</strong> after the last row was fetched (that is, <strong>FETCH_ROW</strong> returned &quot;0&quot;).</td>
</tr>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28552</td>
<td>Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)</td>
</tr>
<tr>
<td>ORA-28553</td>
<td>The position of the bind variable is out of range.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A <strong>NULL</strong> value was passed for a <strong>NOT NULL</strong> parameter.</td>
</tr>
</tbody>
</table>

81.3.13 OPEN_CURSOR Function

This function opens a cursor for running a passthrough SQL statement at the non-Oracle system. This function must be called for any type of SQL statement.

The function returns a cursor, which must be used in subsequent calls. This call allocates memory. To deallocate the associated memory, call the procedure **CLOSE_CURSOR**.

Syntax

```
DBMS_HS_PASSTHROUGH.OPEN_CURSOR
   RETURN BINARY_INTEGER;
```

Pragmas

Purity level defined: WNDS, RNDS

Return Values

The cursor to be used on subsequent procedure and function calls.
81.3.14 PARSE Procedure

This procedure parses an SQL statement at a non-Oracle system.

Syntax

```sql
DBMS_HS_PASSTHROUGH.PARSE (
    c       IN  BINARY_INTEGER NOT NULL,
    stmt    IN  VARCHAR2 NOT NULL);
```

Pragmas

- Purity level defined: WNDS, RNDS

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor associated with the passthrough SQL statement. Cursor must be opened using function OPEN_CURSOR.</td>
</tr>
<tr>
<td>stmt</td>
<td>Statement to be parsed.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-28550</td>
<td>The cursor passed is invalid.</td>
</tr>
<tr>
<td>ORA-28551</td>
<td>SQL statement is illegal.</td>
</tr>
<tr>
<td>ORA-28555</td>
<td>A NULL value was passed for a NOT NULL parameter.</td>
</tr>
</tbody>
</table>
DBMS_ILM

The DBMS_ILM package provides an interface for implementing Information Lifecycle Management (ILM) strategies using Automatic Data Optimization (ADO) policies.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Exceptions
- Summary of DBMS_ILM Subprograms

See Also:

- Oracle Database VLDB and Partitioning Guide for information about managing Automatic Data Optimization (ADO) with this package
- DBMS_ILM_ADMIN
- DBMS_HEAT_MAP

82.1 DBMS_ILM Overview

To implement your ILM strategy, you can use Heat Map in Oracle Database to track data access and modification. You can also use Automatic Data Optimization (ADO) to automate the compression and movement of data between different tiers of storage within the database. The DBMS_ILM package supports immediate evaluation or execution of ADO related tasks.

The package supports the following two ways for scheduling ADO actions.

- A database user schedules immediate ADO policy execution on a set of objects.
- A database user views the results of evaluation of ADO policies on a set of objects. The user then adds or deletes objects to this set and reviews the results of ADO policy evaluation again. The user repeats this step to determine the set of objects for ADO execution. The user can then schedule ADO actions for immediate execution on this set of objects.

The following procedures support the two usage modes. Before describing the procedures, we introduce the notion of an ADO task as an entity that helps to track a particular evaluation or (an evaluation and execution) of ADO policies. A particular ADO task could be in one of the following states.

- Inactive
- Active
82.2 DBMS_ILM Security Model

The **DBMS_ILM** package runs under invoker's rights.

82.3 DBMS_ILM Constants

The table in this topic lists the constants used by the **DBMS_ILM** package.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILM_ALL_POLICIES</td>
<td>'ALL POLICIES'</td>
<td>VARCHAR2(20)</td>
<td>Selects all ADO policies on an object</td>
</tr>
<tr>
<td>ILM_EXECUTION_OF_FLINE</td>
<td>1</td>
<td>NUMBER</td>
<td>Specifies that the object may be offline while ADO action is performed</td>
</tr>
<tr>
<td>ILM_EXECUTION_ON_LINE</td>
<td>2</td>
<td>NUMBER</td>
<td>Specifies that the object should be online while ADO action is performed</td>
</tr>
<tr>
<td>SCOPE_DATABASE</td>
<td>1</td>
<td>NUMBER</td>
<td>Selects all ADO policies in the database</td>
</tr>
<tr>
<td>SCOPE_SCHEMA</td>
<td>2</td>
<td>NUMBER</td>
<td>Selects all ADO policies in the current schema</td>
</tr>
<tr>
<td>SCHEDULE_IMMEDIATE</td>
<td>1</td>
<td>NUMBER</td>
<td>Schedules ADO task for immediate execution</td>
</tr>
<tr>
<td>ARCHIVE_STATE_ACTIVE</td>
<td>'0'</td>
<td>VARCHAR2(1)</td>
<td>Represents the value of the ORA_ARCHIVE_STATE column of a row-archival enabled table that would make the row active</td>
</tr>
<tr>
<td>ARCHIVE_STATE_ARCHIVED</td>
<td>'1'</td>
<td>VARCHAR2(1)</td>
<td>Represents the value of the ORA_ARCHIVE_STATE column of a row-archival enabled table that would make the row inactive</td>
</tr>
</tbody>
</table>

82.4 DBMS_ILM Exceptions

The table in this topic lists the exceptions raised by the **DBMS_ILM** package.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGUMENT_VALUE</td>
<td>38327</td>
<td>Invalid argument value</td>
</tr>
<tr>
<td>INVALID_ILM_DICTIONARY</td>
<td>38328</td>
<td>Inconsistent dictionary state</td>
</tr>
<tr>
<td>INTERNAL_ILM_ERROR</td>
<td>38329</td>
<td>Internal error</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGES</td>
<td>38330</td>
<td>Insufficient privileges</td>
</tr>
</tbody>
</table>
82.5 Summary of DBMS_ILM Subprograms

This table lists and describes the DBMS_ILM package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TO_ILM Procedure</td>
<td>Adds the object specified through the argument to a particular ADO task and evaluates the ADO policies on this object.</td>
</tr>
<tr>
<td>ARCHIVESTATENAME Function</td>
<td>Returns the value of the ORA_ARCHIVE_STATE column of a row-archival enabled table.</td>
</tr>
<tr>
<td>EXECUTE_ILM Procedure</td>
<td>Executes an ADO task.</td>
</tr>
<tr>
<td>EXECUTE_ILM_TASK Procedure</td>
<td>Executes an ADO task that has been evaluated previously.</td>
</tr>
<tr>
<td>PREVIEW_ILM Procedure</td>
<td>Evaluates all ADO policies in the scope specified by means of an argument.</td>
</tr>
<tr>
<td>REMOVE_FROM_ILM Procedure</td>
<td>Removes the object specified through the argument from a particular ADO task.</td>
</tr>
<tr>
<td>STOP_ILM Procedure</td>
<td>Stops ADO-related jobs created for a particular ADO task.</td>
</tr>
</tbody>
</table>

82.5.1 ADD_TO_ILM Procedure

This procedure adds the object specified through the argument to a particular ADO task and evaluates the ADO policies on this object.

The procedure can only be executed on an ADO task in an inactive state. The results of the ADO policy evaluation on this object can be viewed using the appropriate views depending on role and access (USER_ILMTASKS or DBA_ILMTASKS, USER_ILMEVALUATIONDETAILS or DBA_ILMEVALUATIONDETAILS, USER_ILMRESULTS or DBA_ILMRESULTS).

Syntax

```sql
DBMS_ILM.ADD_TO_ILM (  
  task_id           IN    NUMBER,  
  owner             IN    VARCHAR2,  
  object_name       IN    VARCHAR2,  
  subobject_name    IN    VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>Identifies a particular ADO task</td>
</tr>
<tr>
<td>owner</td>
<td>Owner of the object</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object</td>
</tr>
<tr>
<td>subobject_name</td>
<td>Name of the subobject (partition name in the case of partitioned tables)</td>
</tr>
</tbody>
</table>
82.5.2 ARCHIVESTATENAME Function

This function returns the value of the ORA_ARCHIVE_STATE column of a row-archival enabled table.

Syntax

DBMS_ILM.ARCHIVESTATENAME (value IN VARCHAR2)
RETURN VARCHAR2;

Parameters

Table 82-5  ARCHIVESTATENAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value for which the archive state name is to be returned</td>
</tr>
</tbody>
</table>

Usage Notes

Returns ARCHIVE_STATE_ACTIVE for 0, ARCHIVE_STATE_ARCHIVED for others

See Also:

"Using In-Database Archiving" in Oracle Database VLDB and Partitioning Guide

82.5.3 EXECUTE_ILM Procedure

This procedure executes an ADO task.

There are two overloads to this procedure. The first overload executes an ADO task for a set of objects without having evaluated them previously. The second overload executes ADO policies for a specific object.

Syntax

DBMS_ILM.EXECUTE_ILM (task_id OUT NUMBER, ilm_scope IN NUMBER DEFAULT SCOPE_SCHEMA, execution_mode IN NUMBER DEFAULT ILM_EXECUTION_ONLINE);

DBMS_ILM.EXECUTE_ILM (owner IN VARCHAR2, object_name IN VARCHAR2, task_id OUT NUMBER, subobject_name IN VARCHAR2 DEFAULT NULL, policy_name IN VARCHAR2 DEFAULT ILM_ALL_POLICIES, execution_mode IN NUMBER DEFAULT ILM_EXECUTION_ONLINE);
Parameters

Table 82-6  EXECUTE_ILM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>Identifies a particular ADO task</td>
</tr>
<tr>
<td>ilm_scope</td>
<td>Determines the set of objects considered for ADO execution. The default is to consider only the objects in the schema.</td>
</tr>
<tr>
<td>execution_mode</td>
<td>Whether the ADO task be executed online (ILM_EXECUTION_ONLINE) or offline (ILM_EXECUTION_OFFLINE)</td>
</tr>
<tr>
<td>owner</td>
<td>Owner of the object</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object</td>
</tr>
<tr>
<td>subobject_name</td>
<td>Name of the subobject (partition name in the case of partitioned tables)</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of the ADO policy to be evaluated on the object. The package constant ILM_ALL_POLICIES should be used if all ADO policies on an object should be evaluated.</td>
</tr>
</tbody>
</table>

Usage Notes

- The EXECUTE_ILM procedure can be used by users who want more control of when ADO is performed, and who do not want to wait until the next maintenance window.
- The procedure executes like a DDL in that it auto commits before and after the ADO task and related jobs are created.

82.5.4 EXECUTE_ILM_TASK Procedure

This procedure executes an ADO task that has been evaluated previously and moves it to an active state.

Syntax

```sql
DBMS_ILM.EXECUTE_ILM_TASK (    task_id IN NUMBER,    execution_mode IN NUMBER DEFAULT ILM_EXECUTION_ONLINE),
    execution_schedule IN NUMBER DEFAULT SCHEDULE_IMMEDIATE);
```

Parameters

Table 82-7  EXECUTE_ILM_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>Identifies a particular ADO task</td>
</tr>
<tr>
<td>execution_mode</td>
<td>Whether the ADO task be executed online (ILM_EXECUTION_ONLINE) or offline (ILM_EXECUTION_OFFLINE)</td>
</tr>
<tr>
<td>execution_schedule</td>
<td>Identifies when the ADO task should be executed. Currently, the only choice available is immediate scheduling of ADO jobs</td>
</tr>
</tbody>
</table>
82.5.5 PREVIEW_ILM Procedure

This procedure evaluates the ADO policies on the objects specified using the ILM_SCOPE argument.

It returns a number as task_id which identifies a particular ADO task. This can be used to view the results of the policy evaluation in the appropriate views depending on role and access (USER_ILMTASKS or DBA_ILMTASKS, USER_ILMEVALUATIONDETAILS or DBA_ILMEVALUATIONDETAILS, USER_ILMRESULTS or DBA_ILMRESULTS).

The PREVIEW_ILM procedure leaves the ADO task in an inactive state. Once you have previewed the results, you can add or delete objects to this task.

Syntax

```sql
DBMS_ILM.PREVIEW_ILM (  
    task_id           OUT    NUMBER,  
    ilm_scope          IN     NUMBER DEFAULT SCOPE_SCHEMA);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>Identifies a particular ADO task</td>
</tr>
<tr>
<td>ilm_scope</td>
<td>Identifies the scope of execution. Should be either SCOPE_DATABASE or SCOPE_SCHEMA as described in Constants</td>
</tr>
</tbody>
</table>

82.5.6 REMOVE_FROM_ILM Procedure

This procedure removes the object specified through the argument from a particular ADO task.

The procedure can only be executed on an ADO task in an inactive state.

Syntax

```sql
DBMS_ILM.REMOVE_FROM_ILM (  
    task_id           IN    NUMBER,  
    owner             IN    VARCHAR2,  
    object_name       IN    VARCHAR2,  
    subobject_name    IN    VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>Identifies a particular ADO task</td>
</tr>
<tr>
<td>owner</td>
<td>Owner of the object</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object</td>
</tr>
</tbody>
</table>
### 82.5.7 STOP_ILM Procedure

This procedure terminates ILM ADO jobs associated to a particular task Id or job name.

#### Syntax

```sql
DBMS_ILM.STOP_ILM (
  task_id               IN         NUMBER,
  p_drop_running_jobs   IN         BOOLEAN  DEFAULT FALSE,
  p_jobname             IN         VARCHAR2 DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>Number that uniquely identifies a particular ADO task</td>
</tr>
<tr>
<td>p_drop_running_jobs</td>
<td>Determines whether running jobs are dropped</td>
</tr>
<tr>
<td>p_jobname</td>
<td>Name of job to be terminated</td>
</tr>
</tbody>
</table>

---

**Table 82-9  (Cont.) REMOVE_FROM_ILM Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subobject_name</td>
<td>Name of the subobject (partition name in the case of partitioned tables)</td>
</tr>
</tbody>
</table>
The **DBMS_ILM_ADMIN** package provides an interface to customize Automatic Data Optimization (ADO) policy execution. In combination with partitioning and compression, ADO policies can be used to help implement an Information Lifecycle Management (ILM) strategy.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Summary of DBMS_ILM_ADMIN Subprograms

### See Also:

- *Oracle Database VLDB and Partitioning Guide* for information about managing Automatic Data Optimization (ADO) with this package
- DBMS_ILM
- DBMS_HEAT_MAP

#### 83.1 DBMS_ILM_ADMIN Overview

To implement your ILM strategy, you can use Heat Map in Oracle Database to track data access and modification. You can also use Automatic Data Optimization (ADO) to automate the compression and movement of data between different tiers of storage within the database.

#### 83.2 DBMS_ILM_ADMIN Security Model

This package runs under definer's rights. The user requires DBA privileges.

#### 83.3 DBMS_ILM_ADMIN Constants

The table in this topic describes constants used by the **DBMS_ILM_ADMIN** package.

The value column refers to the numeric or character value that the constants resolve to.
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS_JOBLIMIT</td>
<td>12</td>
<td>NUMBER</td>
<td>Specifies the absolute number of concurrent ILM ADO jobs.</td>
</tr>
<tr>
<td>EXECUTION_INTERVAL</td>
<td>1</td>
<td>NUMBER</td>
<td>Determines the frequency with which ADO background evaluation occurs. Specified in minutes.</td>
</tr>
<tr>
<td>RETENTION_TIME</td>
<td>2</td>
<td>NUMBER</td>
<td>Controls the amount of time ADO history should be maintained. Specified in days.</td>
</tr>
<tr>
<td>EXECUTION_MODE</td>
<td>4</td>
<td>NUMBER</td>
<td>Controls whether ADO execution is online, offline. The value for this parameter should either be DBMS_ILM_ADMIN.ILM_EXECUTION_OFFLINE or DBMS_ILM_ADMIN.ILM_EXECUTION_ONLINE.</td>
</tr>
<tr>
<td>JOBLIMIT</td>
<td>5</td>
<td>NUMBER</td>
<td>Controls the upper limit on number of ILM ADO jobs at any time. The maximum number of concurrent ADO jobs is JOBLIMIT *(number of instances) * (number of CPUs per instance).</td>
</tr>
<tr>
<td>JOB_SIZELIMIT</td>
<td>13</td>
<td>NUMBER</td>
<td>Specifies the size (in megabytes) of the data that is processed by a single ILM ADO row level compression job.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>7</td>
<td>NUMBER</td>
<td>Provides a way to turn background ADO off or on.</td>
</tr>
<tr>
<td>TBS_PERCENT_USED</td>
<td>8</td>
<td>NUMBER</td>
<td>Decides when a tablespace is considered full. Specified as a percentage of tablespace quota.</td>
</tr>
<tr>
<td>TBS_PERCENT_FREE</td>
<td>9</td>
<td>NUMBER</td>
<td>Decides the targeted tablespace storage through ADO actions as a percentage of tablespace quota.</td>
</tr>
<tr>
<td>DEG_PARALLEL</td>
<td>10</td>
<td>NUMBER</td>
<td>Decides the degree of parallelism to be used for ADO jobs</td>
</tr>
<tr>
<td>POLICY_TIME</td>
<td>11</td>
<td>NUMBER</td>
<td>Decides if ADO policies are treated as though they are specified in seconds rather than days. Can take value ILM_POLICY_IN_SECONDS (treat policy time in seconds) or ILM_POLICY_IN_DAYS (treat policy time in days - default).</td>
</tr>
<tr>
<td>HEAT_MAP_SEG_READ</td>
<td>1</td>
<td>NUMBER</td>
<td>Segment read done</td>
</tr>
<tr>
<td>HEAT_MAP_SEG_WRITE</td>
<td>2</td>
<td>NUMBER</td>
<td>Segment write done</td>
</tr>
<tr>
<td>HEAT_MAP_SEG_SCAN</td>
<td>4</td>
<td>NUMBER</td>
<td>Full table scan done</td>
</tr>
<tr>
<td>HEAT_MAP_SEG_LOOKUP</td>
<td>8</td>
<td>NUMBER</td>
<td>Index scan done</td>
</tr>
</tbody>
</table>

The DBMS_ILM_ADMIN package uses the constants as parameter values shown in Table 83-2.
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILM_EXECUTION_OFFLINE</td>
<td>1</td>
<td>NUMBER</td>
<td>Specifies that the object may be offline while ADO action is performed.</td>
</tr>
<tr>
<td>ILM_EXECUTION_ONLINE</td>
<td>2</td>
<td>NUMBER</td>
<td>Specifies that the object should be online while ADO action is performed.</td>
</tr>
<tr>
<td>ILM_ENABLED</td>
<td>1</td>
<td>NUMBER</td>
<td>Indicates automatic ADO policy evaluation and execution is enabled.</td>
</tr>
<tr>
<td>ILM_DISABLED</td>
<td>2</td>
<td>NUMBER</td>
<td>Indicates automatic ADO policy evaluation and execution is disabled.</td>
</tr>
<tr>
<td>ILM_POLICY_IN_DAYS</td>
<td>0</td>
<td>NUMBER</td>
<td>Indicates policy is specified in days. This is the default.</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ILM_POLICY_IN SECONDS</td>
<td>1</td>
<td>NUMBER</td>
<td>Indicates policy unit is changed from days to seconds. This could be used to test ADO policy evaluation quickly instead of waiting for the policy duration.</td>
</tr>
</tbody>
</table>

**Note:**
- Setting ILM_POLICY_IN_SECONDS does not compress the blocks within the specified seconds.
- Setting ILM_POLICY_IN_SECONDS is for test ADO and should not be set in the production environment.
83.4 Summary of DBMS_ILM_ADMIN Subprograms

This table lists and briefly describes the DBMS_ILM_ADMIN package subprograms.

Table 83-3  DBMS_ILM_ADMIN Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR_HEAT_MAP_ALL Procedure</td>
<td>Deletes all rows except the dummy row</td>
</tr>
<tr>
<td>CLEAR_HEAT_MAP_TABLE Procedure</td>
<td>Clears all or some statistics for the heat map table, deleting rows for a</td>
</tr>
<tr>
<td></td>
<td>given table or segment which match a given pattern, or all such rows</td>
</tr>
<tr>
<td>CUSTOMIZE_ILM Procedure</td>
<td>Customizes environment for ILM execution by specifying the values for</td>
</tr>
<tr>
<td></td>
<td>ILM execution related parameters</td>
</tr>
<tr>
<td>DISABLE_ILM Procedure</td>
<td>Turns off all background ILM scheduling</td>
</tr>
<tr>
<td>ENABLE_ILM Procedure</td>
<td>Turns on all background ILM scheduling</td>
</tr>
<tr>
<td>SET_HEAT_MAP_ALL Procedure</td>
<td>Updates or inserts heat map rows for all tables</td>
</tr>
<tr>
<td>SET_HEAT_MAP_START Procedure</td>
<td>Sets the start date for collecting heat map data</td>
</tr>
<tr>
<td>SET_HEAT_MAP_TABLE Procedure</td>
<td>Updates or inserts a row for the specified table or segment</td>
</tr>
</tbody>
</table>

83.4.1 CLEAR_HEAT_MAP_ALL Procedure

This procedure deletes all rows in HEAT_MAP_STAT$ except the dummy row.

Syntax

DBMS_ILM_ADMIN.CLEAR_HEAT_MAP_ALL;

83.4.2 CLEAR_HEAT_MAP_TABLE Procedure

This procedure clears all or some statistics for the heat map table, deleting rows for a given table or segment which match a given pattern, or all such rows.

Syntax

DBMS_ILM_ADMIN.CLEAR_HEAT_MAP_TABLE (  
    owner IN VARCHAR2,  
    tablename IN VARCHAR2,  
    partition IN VARCHAR2 default '',  
    access_date IN DATE DEFAULT NULL,  
    segment_access_summary IN NUMBER DEFAULT NULL);

Parameters

Table 83-4  CLEAR_HEAT_MAP_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Table owner</td>
</tr>
</tbody>
</table>
Table 83-4  (Cont.) CLEAR_HEAT_MAP_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Table name</td>
</tr>
<tr>
<td>partition</td>
<td>Name of the subobject, defaults to NULL</td>
</tr>
<tr>
<td>access_date</td>
<td>Date for the entry in HEAT_MAP_STAT$ to be removed</td>
</tr>
<tr>
<td>segment_access_summary</td>
<td>Summary of segment access constants indicating access operations performed on the segment</td>
</tr>
</tbody>
</table>

83.4.3 CUSTOMIZE_ILM Procedure

This procedure customizes environment for ILM execution by specifying the values for ILM execution related parameters. These values take effect for the next background scheduling.

Syntax

```sql
DBMS_ILM_ADMIN.CUSTOMIZE_ILM  (
    parameter            IN       NUMBER,
    value                IN       NUMBER);
```

Parameters

Table 83-5  CUSTOMIZE_ILM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>One of the parameter constants defined in DBMS_ILM_ADMIN package</td>
</tr>
<tr>
<td>value</td>
<td>Value of parameter</td>
</tr>
</tbody>
</table>

83.4.4 DISABLE_ILM Procedure

This procedure turns off all background ILM scheduling.

Syntax

```sql
DBMS_ILM_ADMIN.DISABLE_ILM;
```

83.4.5 ENABLE_ILM Procedure

This procedure turns on all background ILM scheduling.

Syntax

```sql
DBMS_ILM_ADMIN.ENABLE_ILM;
```
83.4.6 SET_HEAT_MAP_ALL Procedure

This procedure sets an HTTP request header. The request header is sent to the Web server as soon as it is set.

Syntax

```sql
DBMS_IJM_ADMIN.SET_HEAT_MAP_ALL (
    access_date    IN DATE,
    segment_access_summary IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_date</td>
<td>Date for the entry in HEAT_MAP_STATS to be added</td>
</tr>
<tr>
<td>segment_access_summary</td>
<td>Summary of segment access constants indicating access operations performed on the segment</td>
</tr>
</tbody>
</table>

83.4.7 SET_HEAT_MAP_START Procedure

This procedure sets the start date for collecting heat map data.

Syntax

```sql
DBMS_IJM_ADMIN.SET_HEAT_MAP_START (
    start_date  IN   DATE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>Indicates the new date from which all statistics are valid</td>
</tr>
</tbody>
</table>

83.4.8 SET_HEAT_MAP_TABLE Procedure

This procedure updates or inserts a row for the specified table or segment.

Syntax

```sql
DBMS_IJM_ADMIN.SET_HEAT_MAP_TABLE (
    owner                  IN VARCHAR2,
    tablename              IN VARCHAR2,
    partition              IN VARCHAR2 DEFAULT '',
    access_date            IN DATE DEFAULT NULL,
    segment_access_summary IN NUMBER DEFAULT NULL);
```
### Parameters

Table 83-8  *SET_HEAT_MAP_TABLE Procedure Parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Table owner</td>
</tr>
<tr>
<td>tablename</td>
<td>Table name</td>
</tr>
<tr>
<td>partition</td>
<td>Name of the subobject, defaults to NULL</td>
</tr>
<tr>
<td>access_date</td>
<td>Date for the entry in HEAT_MAP_STAT$ to be added</td>
</tr>
<tr>
<td>segment_access_summary</td>
<td>Summary of segment access constants indicating access operations performed on the segment</td>
</tr>
</tbody>
</table>
The **DBMS_INMEMORY** package provides an interface for In-Memory Column Store (IM column store) functionality.

This chapter contains the following topics:

- DBMS_INMEMORY Overview
- DBMS_INMEMORY Security Model
- Summary of DBMS_INMEMORY Subprograms

### 84.1 DBMS_INMEMORY Overview

This package contains procedures for populating and repopulating the IM column store, and for dropping IM expressions from a specified table.

**IM Population and Repopulation**

In-Memory population (population) occurs when the database reads existing row-format data from disk, transforms it into columnar format, and then stores it in the IM column store. Only objects with the **INMEMORY** attribute are eligible for population.

Population, which transforms existing data on disk into columnar format, is different from repopulation, which loads new data into the IM column store. Repopulation occurs automatically after their columnar data undergo significant DML activity.

When an object has the **INMEMORY** attribute and a priority other than **NONE**, the database gradually populates the object in the IM column store according to an internal priority queue. Objects with priority of **NONE** are populated only when they undergo a full scan. The **DBMS_INMEMORY.POPULATE** procedure forces immediate population of an object. Similarly, **DBMS_INMEMORY.REPOPULATE** procedure forces immediate repopulation of an object.

**See Also:**

*Oracle Database In-Memory Guide* to learn more about IM population

**IM Expressions**

IM expressions populate frequently evaluated query expressions in the IM column store for subsequent reuse. An IM expression is materialized as a hidden virtual column, prefixed with the string **SYS_IME**, and is accessed in the same way as a non-virtual column.

When you use **DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS**, the database adds the 20 hottest expressions to their respective tables as **SYS_IME** columns and applies the default **INMEMORY** column compression clause. If any **SYS_IME** columns that were added during a previous invocation are no longer in the latest expression list, then the database changes their attribute to **NO INMEMORY**.
The maximum number of SYS_IME columns for a table, regardless of whether the attribute is INMEMORY or NO INMEMORY, is 50. After the 50 expression limit is reached for a table, the database will not add new SYS_IME columns. To make space for new expressions, you must manually drop SYS_IME columns with the DBMS_INMEMORY.IME_DROP_EXPRESSIONS or DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS procedures.

See Also:

Oracle Database In-Memory Guide to learn more about IM expressions

84.2 DBMS_INMEMORY Security Model

The DBMS_INMEMORY package subprograms execute with invoker’s rights.

The POPULATE and REPOPULATE procedures require the invoking user to have SELECT privileges on the specified object. For IME_DROP_EXPRESSIONS, the invoking user must have ALTER TABLE privileges on the specified table.

84.3 Summary of DBMS_INMEMORY Subprograms

This table lists and briefly describes the DBMS_INMEMORY package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IME_DROP_EXPRESSIONS Procedure</td>
<td>Drops a specified set of SYS_IME virtual columns from a table</td>
</tr>
<tr>
<td>POPULATE Procedure</td>
<td>Forces population of the specified table</td>
</tr>
<tr>
<td>REPOPULATE Procedure</td>
<td>Forces repopulation of the specified table</td>
</tr>
<tr>
<td>SEGMENT_DEALLOCATE_VERSIONS Procedure</td>
<td>Deallocates non-current IMCUs in the IM column store</td>
</tr>
</tbody>
</table>

84.3.1 IME_DROP_EXPRESSIONS Procedure

This procedure drops a specified set of SYS_IME virtual columns from a table.

Syntax

```sql
DBMS_INMEMORY.IME_DROP_EXPRESSIONS(
    schema_name  IN  VARCHAR2,
    table_name   IN  VARCHAR2,
    column_name  IN  VARCHAR2 DEFAULT NULL);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that contains the In-Memory table</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the In-Memory table that contains the SYS_IME columns</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the SYS_IME column. By default this value is null, which specifies all SYS_IME columns in this table.</td>
</tr>
</tbody>
</table>

Usage Notes

Typical reasons for dropping SYS_IME columns are space and performance. The maximum number of SYS_IME columns for a table, regardless of whether the attribute is INMEMORY or NO INMEMORY, is 50. After the 50-expression limit is reached for a table, the database will not add new SYS_IME columns. To make space for new expressions, you must manually drop SYS_IME columns with the DBMS_INMEMORY.IME_DROP_EXPRESSIONS or DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS procedures.

To drop a specified SYS_IME column or all SYS_IME columns in the requested table, use DBMS_INMEMORY.IME_DROP_EXPRESSIONS. To populate these segments again, either invoke the DBMS_INMEMORY.POPULATE procedure, or perform a full table scan.

84.3.2 POPULATE Procedure

This procedure forces population of the specified table, partition, or subpartition into the IM column store.

Syntax

DBMS_INMEMORY.POPULATE(
    schema_name   IN    VARCHAR2,
    table_name    IN    VARCHAR2,
    subobject_name IN    VARCHAR2 DEFAULT NULL);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of schema</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of table</td>
</tr>
<tr>
<td>subobject_name</td>
<td>Partition or subpartition</td>
</tr>
</tbody>
</table>
84.3.3 REPOPULATE Procedure

This procedure forces repopulation of a table, partition, or subpartition that is currently populated in the IM column store.

**Syntax**

```
DBMS_INMEMORY.REPOPULATE(
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2,
    subobject_name IN VARCHAR2 DEFAULT NULL,
    force IN BOOLEAN DEFAULT FALSE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema that owns the object.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table requiring repopulation.</td>
</tr>
<tr>
<td>subobject_name</td>
<td>Name of the partition or subpartition. If null, then repopulate the entire table.</td>
</tr>
<tr>
<td>force</td>
<td>Whether to repopulate all IMCUs in the segment, just as in initial population.</td>
</tr>
</tbody>
</table>

The following values are possible for the `force` parameter:

- **FALSE** — The database repopulates only IMCUs containing modified rows. This is the default.
- **TRUE** — The database drops the segment, and then rebuilds it. The database increments the statistics and performs all other tasks related to initial population.

For example, IMCU 1 contains rows 1 to 500,000, and IMCU 2 contains rows 500,001 to 1,000,000. A statement modifies row 600,000. When `force` is **FALSE**, the database only repopulates IMCU 2. When `force` is **TRUE**, the database repopulates both IMCUs.

Consider further that the `INMEMORY_VIRTUAL_COLUMNS` initialization parameter is set to **ENABLE**, and an application creates a new virtual column. When `force` is **FALSE**, the database only repopulates IMCU 2 with the new column. When `force` is **TRUE**, the database repopulates both IMCUs with the new column.

84.3.4 SEGMENT_DEALLOCATE_VERSIONS Procedure

This procedure deallocates non-current IMCUs in the IM column store.

**Syntax**

```
DBMS_INMEMORY.SEGMENT_DEALLOCATE_VERSIONS(
    SCHEMA_NAME IN VARCHAR2,
    TABLE_NAME IN VARCHAR2,
```
PARTITION_NAME   IN  VARCHAR2 DEFAULT NULL,
SPCPRESSURE      IN  BOOLEAN DEFAULT FALSE);

Parameters

Table 84-5  SEGMENT_DEALLOCATE_VERSIONS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema that owns the object.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table requiring repopulation.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition or subpartition. If null, then repopulate the entire table.</td>
</tr>
<tr>
<td>spcpressure</td>
<td>Whether to force deallocation of non-current IMCUs (TRUE), or wait for the database to deallocate them automatically. By default, the database deallocates non-current IMCUs every two minutes.</td>
</tr>
</tbody>
</table>

Usage Notes

During repopulation, the IM column store maintains both the current IMCU and non-current IMCU. This mechanism, which is called double buffering, ensures that queries do not decrease performance because an IMCU is unavailable during repopulation. After repopulation completes, the IM column store retains the non-current IMCU for a short time (2 minutes by default) to optimize queries with older SCNs. Typically, the default behavior is sufficient. However, you can force deallocation of non-current IMCUs by using the SEGMENT_DEALLOCATE_VERSIONS procedure.

Example

The following program forces deallocation of non-current IMCUs for the products table:

```
BEGIN
  DBMS_INMEMORY.SEGMENT_DEALLOCATE_VERSIONS(
    schema_name => 'SH',
    table_name  => 'PRODUCTS',
    spcpressure => TRUE );
END;
```

See Also:

Oracle Database In-Memory Guide to learn more about double buffering in the IM column store
DBMS_INMEMORY_ADMIN provides interfaces for managing an In-Memory FastStart (IM FastStart) area and In-Memory Expressions (IM expressions).

This chapter contains the following topics:

- DBMS_INMEMORY_ADMIN Overview
- DBMS_INMEMORY_ADMIN Security Model
- DBMS_INMEMORY_ADMIN Operational Notes
- Summary of DBMS_INMEMORY_ADMIN Subprograms

See Also:
Oracle Database In-Memory Guide to learn more about Oracle Database In-Memory features

85.1 DBMS_INMEMORY_ADMIN Overview

This package provides interfaces for managing In-Memory Expressions (IM expressions) and the In-Memory FastStart (IM FastStart) area.

IM Expressions

Analytic queries often contain complex expressions or calculations that consume significant CPU and memory during execution. Use IME_CAPTURE_EXPRESSIONS to identify these frequently used ("hot") expressions and IME_POPULATE_EXPRESSIONS to populate them in the IM column store. By using IM expressions, the database avoids repeated computations and improves performance.

The database represents IM expressions as system-generated virtual columns. The name of an IM virtual column begins with `SYS_IME`. You can also use DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS and DBMS_INMEMORY.IME_DROP_EXPRESSIONS to remove existing SYS_IME columns.

The DBA_IM_EXPRESSIONS view shows the SYS_IME columns that have the INMEMORY attribute. After using the IME_CAPTURE_EXPRESSIONS procedure, you can query this view to see the hot expressions added to different tables in the database.

See Also:
Oracle Database In-Memory Guide to learn more about IM expressions
IM FastStart Area

The IM FastStart area stores data that optimizes the population of the IM column store when the database restarts. Because the database reads columnar data directly from persistent storage without needing to compress or format it, population is faster when a database instance restarts.

When you enable IM FastStart for the IM column store, you must specify an ASSM tablespace for the IM FastStart area. The tablespace stores the data in a SecureFiles LOB named SYSDBIMFS_LOGSEG$. The SYSAUX tablespace stores the metadata. When data is populated or repopulated in the IM column store, the database automatically writes the data to the IM FastStart area. You cannot manually force a write. If you specify an object as NO INMEMORY, then the database removes it from the IM FastStart area.

When the IM FastStart area is under space pressure, the database automatically drops the oldest 15% of segments and continues saving columnar data. If space is unavailable, then the database stops writing to the IM FastStart area.

See Also:
Oracle Database In-Memory Guide to learn more about IM expressions

Automatic In-Memory

Automatic In-Memory uses access tracking and column statistics to manage objects in the IM column store. If the IM column store is full, and if other more frequently accessed segments would benefit from population in the IM column store, then the IM column store evicts inactive segments. If the IM column store is configured to hold all INMEMORY segments, however, then Automatic In-Memory takes no action.

By default, Automatic In-Memory checks usage statistics for the past 31 days. You can change the current setting by supplying the AIM_STATWINDOW_DAYS parameter to DBMS_INMEMORY_ADMIN.AIM_SET_PARAMETER.

See Also:
Oracle Database In-Memory Guide to learn more about Automatic In-Memory

85.2 DBMS_INMEMORY_ADMIN Security Model

This package requires administrator privileges. Package subprograms execute with invoker's rights.
85.3 DBMS_INMEMORY_ADMIN Operational Notes

It is possible for a DBMS_INMEMORY_ADMIN FastStart operation to fail or be interrupted.

In a failure or interruption scenario, the following rules determine which subprograms you can use:

- If FASTSTART_ENABLE does not succeed, then the only permitted operation is re-executing FASTSTART_ENABLE.
- If FASTSTART_MIGRATE_STORAGE does not succeed, then the only permitted operation is re-executing FASTSTART_MIGRATE_STORAGE.
- If FASTSTART_DISABLE does not succeed, then all DBMS_INMEMORY_ADMIN operations are permitted.

85.4 Summary of DBMS_INMEMORY_ADMIN Subprograms

This table lists the DBMS_INMEMORY_ADMIN subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIM_GET_PARAMETER Procedure</td>
<td>This procedure obtains the current values for parameters that control Automatic In-Memory.</td>
</tr>
<tr>
<td>AIM_SET_PARAMETER Procedure</td>
<td>The procedure customizes the execution environment of Automatic In-Memory.</td>
</tr>
<tr>
<td>FASTSTART_DISABLE Procedure</td>
<td>This procedure disables the In-Memory FastStart (IM FastStart) feature.</td>
</tr>
<tr>
<td>FASTSTART_ENABLE Procedure</td>
<td>This procedure enables IM FastStart and assigns a tablespace.</td>
</tr>
<tr>
<td>FASTSTART_MIGRATE_STORAGE Procedure</td>
<td>This procedure moves all IM FastStart data and metadata from the existing tablespace to the specified new tablespace.</td>
</tr>
<tr>
<td>GET_FASTSTART_TABLESPACE Function</td>
<td>This function returns the name of the tablespace that is currently designated for IM FastStart.</td>
</tr>
<tr>
<td>IME_CAPTURE_EXPRESSIONS Procedure</td>
<td>This procedure captures the 20 most frequently accessed (“hottest”) expressions in the database in the specified time interval.</td>
</tr>
<tr>
<td>IME_CLOSE_CAPTURE_WINDOW Procedure</td>
<td>This procedure signals the end of the current expression capture window.</td>
</tr>
<tr>
<td>IME_DROP_ALL_EXPRESSIONS Procedure</td>
<td>This procedure drops all SYS_IME expression virtual columns in the database.</td>
</tr>
<tr>
<td>IME_GET_CAPTURE_STATE Procedure</td>
<td>This procedure returns the current capture state of the expression capture window and the timestamp of the most recent modification.</td>
</tr>
<tr>
<td>IME_OPEN_CAPTURE_WINDOW Procedure</td>
<td>This procedure signals the beginning of an expression capture window.</td>
</tr>
</tbody>
</table>
Table 85-1  (Cont.) DBMS_INMEMORY_ADMIN Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IME_POPULATE_EXPRESSIONS Procedure</td>
<td>This procedure forces the population of expressions captured in the latest invocation of DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS.</td>
</tr>
</tbody>
</table>

85.4.1 AIM_GET_PARAMETER Procedure

This procedure obtains the current values for parameters that control Automatic In-Memory.

Syntax

```plsql
DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER(
    parameter   IN    NUMBER,
    value       OUT   NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>Specifies a predefined constant that controls Automatic In-Memory. The only valid constant is AIM_STATWINDOW_DAYS, which specifies the number of days in the sliding statistics window. Automatic In-Memory uses this duration to filter statistics for INMEMORY objects as part of its algorithms. For example, if the duration is set to 7 days, then Automatic In-Memory considers only statistics of the past 7 days for its algorithms. The default is 31.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies the value assigned to AIM_STATWINDOW_DAYS.</td>
</tr>
</tbody>
</table>

Example 85-1  Getting the Number of Days in the Statistics Window

The following code prints the number of days in the statistics window to the screen:

```plsql
VARIABLE b_statwin NUMBER
BEGIN
    DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER(DBMS_INMEMORY_ADMIN.AIM_STATWINDOW_DAYS, :b_statwin);
    END;
/

PRINT b_statwin
```
Sample output appears below:

```
B_STATWIN
--------
14
```

See Also:
Oracle Database In-Memory Guide to learn how to use AIM_GET_PARAMETER

85.4.2 AIM_SET_PARAMETER Procedure

The procedure customizes the execution environment of Automatic In-Memory.

Syntax

```
DBMS_INMEMORY_ADMIN.AIM_SET_PARAMETER(
    parameter   IN    NUMBER,
    value       IN    NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>Specifies a predefined constant that controls Automatic In-Memory. The only valid constant is AIM_STATWINDOW_DAYS, which specifies the number of days in the sliding statistics window. The default is 31.</td>
</tr>
<tr>
<td>value</td>
<td>Assigns the value assigned to AIM_STATWINDOW_DAYS.</td>
</tr>
</tbody>
</table>

Example 85-2  Setting the Number of Days in the Statistics Window

The following example gets the current number of days in the window, sets it to 14, and then prints the value to the screen:

```
VARIABLE b_statwin NUMBER
BEGIN
    DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER(DBMS_INMEMORY_ADMIN.AIM_STATWINDOW_DAYS, :b_statwin);
END;
/

PRINT b_statwin
BEGIN
    DBMS_INMEMORY_ADMIN.AIM_SET_PARAMETER(DBMS_INMEMORY_ADMIN.AIM_STATWINDOW_DAYS, 14);
END;
```
85.4.3 FASTSTART_DISABLE Procedure

This procedure disables the In-Memory FastStart (IM FastStart) feature.

Syntax

DBMS_INMEMORY_ADMIN.FASTSTART_DISABLE();

Security Model

Administrator privileges are required to execute this procedure.

Usage Notes

When you execute the procedure, the database executes the following actions:

1. Waits until all IM FastStart operations complete
2. Disables the IM FastStart feature, and performs the following operations:
   • Cleans the IM FastStart area
   • Deletes IM FastStart metadata stored in the SYSAUX tablespace
   • Releases the IM FastStart tablespace (but does not delete it)

This procedure does not interrupt or affect any concurrent IM column store operations.
Examples

The following PL/SQL program disables the IM FastStart feature:

EXEC DBMS_INMEMORY_ADMIN.FASTSTART_DISABLE;

The following query shows that the LOB for the IM FastStart tablespace has been deleted (sample output included):

```
COL OWNER FORMAT a5
COL SEGMENT_NAME FORMAT a20
SELECT   l.OWNER, l.SEGMENT_NAME, SUM(s.BYTES)/1024/1024 MB
FROM     DBA_LOBS l, DBA_SEGMENTS s
WHERE    l.SEGMENT_NAME = s.SEGMENT_NAME
AND      l.TABLESPACE_NAME = 'FS_TBS'
GROUP BY l.OWNER, l.SEGMENT_NAME;
```

no rows selected

85.4.4 FASTSTART_ENABLE Procedure

This procedure enables In-Memory FastStart (IM FastStart), and designates a tablespace for the IM FastStart (FastStart) area.

Syntax

```
DBMS_INMEMORY_ADMIN.FASTSTART_ENABLE(
    tbs_name    IN    VARCHAR2,
    nologging   IN    BOOLEAN DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbs_name</td>
<td>The name of the ASSM tablespace for the FastStart area.</td>
</tr>
<tr>
<td>nologging</td>
<td>The logging mode of the LOB created for the FastStart area.</td>
</tr>
</tbody>
</table>

If the nologging parameter is set to FALSE, then the database creates the FastStart LOB with the LOGGING option. If set to TRUE (default), then the database creates the LOB with the NOLOGGING option.

Security Model

Administrator privileges are required to execute this procedure.

Usage Notes

To enable IM FastStart, the ASSM tablespace specified in FASTSTART_ENABLE must exist, and the SYSAUX tablespace must be online. Only one FastStart tablespace can exist for every PDB or non-CDB. The specified tablespace must have enough space to store data for the IM column store, and it must not contain any other data before it is designated for the FastStart
area. Oracle recommends sizing the tablespace at least twice of the size of the
INMEMORY_SIZE initialization parameter.

The database does not create the FastStart area on disk until the IM column store is
populated. After population, the data periodically saves the columnar data (but not
metadata such as the transaction journal) to the FastStart area, which is represented
on disk as the SYSDBIMFS_LOBSEG$ segment. The database stores the FastStart
metadata in the SYSAUX tablespace. In an Oracle Real Application Clusters (Oracle
RAC) environment, IM FastStart data is shared across all nodes.

Note:

IM FastStart is not supported in a standby database instance.

Whereas the initial loading of IMCUs into memory is expensive and CPU-bound, an IM
FastStart tablespace requires intermittent I/O. The database periodically writes
columnar data to the IM FastStart area. If a database instance must restart, then
Oracle Database reads the columnar data directly from the IM FastStart area rather
than reconstructing the IMCUs from scratch. No compression or formatting of the
columnar data is required.

Examples

This example creates fs_tbs as an ASSM tablespace, and then uses
FASTSTART_ENABLE to specify this tablespace as the IM FastStart area:

```
CREATE TABLESPACE fs_tbs
   DATAFILE 'fs_tbs.dbf' SIZE 500M
   EXTENT MANAGEMENT LOCAL
   SEGMENT SPACE MANAGEMENT AUTO;
EXEC DBMS_INMEMORY_ADMIN.FASTSTART_ENABLE('fs_tbs');
```

The following query shows that the IM FastStart LOB was created (sample output
included):

```
COL OWNER FORMAT a5
COL SEGMENT_NAME FORMAT a20
SELECT  l.OWNER, l.SEGMENT_NAME, SUM(s.BYTES)/1024/1024 MB
FROM     DBA_LOBS l, DBA_SEGMENTS s
WHERE    l.SEGMENT_NAME = s.SEGMENT_NAME
AND      l.TABLESPACE_NAME = 'FS_TBS'
GROUP BY l.OWNER, l.SEGMENT_NAME;

OWNER SEGMENT_NAME          MB
------- --------------- ----- 
SYS  SYSDBIMFS_LOBSEG$ .125`
85.4.5 FASTSTART_MIGRATE_STORAGE Procedure

This procedure moves the In-Memory FastStart (IM FastStart) data and catalogs from the current tablespace to a new tablespace.

Syntax

```sql
DBMS_INMEMORY_ADMIN.FASTSTART_MIGRATE_STORAGE(
    tbs_name    IN    VARCHAR2);
```

Parameters

Table 85-3  FASTSTART_MIGRATE_STORAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbs_name</td>
<td>The name of the new ASSM tablespace for the IM FastStart area.</td>
</tr>
</tbody>
</table>

Security Model

DBA privileges are required to execute this procedure.

Usage Notes

When you execute the procedure, the database executes the following actions:

1. Waits until all IM FastStart operations complete
2. Disables the IM FastStart feature
3. Copies IM FastStart data and metadata to the new tablespace, leaving the old tablespace intact
4. Re-enables IM FastStart the feature

Examples

The following program obtains the name of the IM FastStart tablespace, if one exists, and prints the result (sample output included):

```sql
VARIABLE b_fstbs VARCHAR2(20)
BEGIN
    :b_fstbs := DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE;
END;
/
PRINT b_fstbs

B_FSTBS
-------------------------
FS_TBS
```
The following statements create a new tablespace named `fs_tbs2`, and then migrate the IM FastStart area to this tablespace:

```sql
CREATE TABLESPACE fs_tbs2
  DATAFILE 'fs_tbs2.dbf' SIZE 500M
  EXTENT MANAGEMENT LOCAL
  SEGMENT SPACE MANAGEMENT AUTO;
EXEC DBMS_INMEMORY_ADMIN.FASTSTART_MIGRATE_STORAGE('fs_tbs2');
```

The following program prints the name of the current IM FastStart tablespace (sample output included):

```sql
BEGIN
  :b_fstbs := DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE;
END;
/
PRINT b_fstbs
```

**85.4.6 GET_FASTSTART_TABLESPACE Function**

This function returns the tablespace assigned to In-Memory FastStart (IM FastStart). If the feature is disabled, then the function returns NOT ENABLED.

**Syntax**

```
DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE();
```

**Security Model**

DBA privileges are required to execute this function.

**Examples**

This program obtains the name of the IM FastStart tablespace, if one exists, and prints the result:

```sql
VARIABLE b_fstbs VARCHAR2(20)
BEGIN
  :b_fstbs := DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE;
END;
/
PRINT b_fstbs
```

```
B_FSTBS
------------------------
FS_TBS2
```

```
NOT ENABLED
```
85.4.7 IME_CAPTURE_EXPRESSIONS Procedure

This procedure captures the 20 most frequently accessed ("hottest") expressions in the database in the specified time interval.

Syntax

```
DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS(
    snapshot    IN    VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshot</td>
<td>Specifies a snapshot that defines the time interval in which expression statistics are considered. You can specify any of the following values:</td>
</tr>
<tr>
<td></td>
<td>• CUMULATIVE</td>
</tr>
<tr>
<td></td>
<td>The database considers all expression statistics since the creation of the database.</td>
</tr>
<tr>
<td></td>
<td>• CURRENT</td>
</tr>
<tr>
<td></td>
<td>The database considers only expression statistics from the past 24 hours.</td>
</tr>
<tr>
<td></td>
<td>• WINDOW</td>
</tr>
<tr>
<td></td>
<td>The database considers statistics for expressions tracked in the most recent expression capture window. The database adds hidden virtual columns for expressions tracked in the most recent window. If the capture window is currently open, then the database considers all expressions tracked in the current window up until this point, and then materializes the hottest expressions. To list the expressions that have been tracked in the current window, query <code>DBA_EXPRESSION_STATISTICS with SNAPSHOT='WINDOW'</code>.</td>
</tr>
</tbody>
</table>

Usage Notes

When you invoke this procedure, the database queries the Expression Statistics Store (ESS), and considers only expressions on tables that are at least partially populated in the IM column store. The database adds the 20 hottest expressions to their respective tables as hidden virtual columns, prefixed with the string `SYS_IME`, and applies the default `INMEMORY` column compression clause. If any `SYS_IME` columns added during a previous invocation are no longer in the latest top 20 list, then the database marks them as `NO INMEMORY`.

**Note:**

Executing the `IME_CAPTURE_EXPRESSIONS` procedure on a standby database has no effect.

The maximum number of `SYS_IME` columns for a table, regardless of whether the attribute is `INMEMORY` or `NO INMEMORY`, is 50. After the limit is reached for a table, the database will not
add new SYS_IME columns. To make space for new expressions, you must manually drop SYS_IME columns with the IME_DROP_ALL_EXPRESSIONS Procedure or IME_DROP_EXPRESSIONS Procedure.

The 50-expression limit for each table, which includes both INMEMORY and NO INMEMORY expressions, is different from the 20-expression limit for the database, which includes only INMEMORY expressions. For example, if 20 tables are populated in the IM column store, then each table might each have 1 SYS_IME column with the INMEMORY attribute, and 49 SYS_IME columns with the NO INMEMORY attribute.

IM expressions and virtual columns are stored in In-Memory structured called In-Memory Expression Units (IMEUs). Every IMEU is linked to a parent In-Memory Compression Unit (IMCU) from which it inherits compression characteristics.

ESS information is stored in the data dictionary and exposed in the DBA_EXPRESSION_STATISTICS view. This view shows the metadata that the optimizer has collected in the ESS. IM expressions are exposed as system-generated virtual columns, prefixed by the string SYS_IME, in the DBA_IM_EXPRESSIONS view.

Example 85-3 Capturing Expressions in a User-Defined Window

This example demonstrates use of the WINDOW capture mode. Your goal is to open and close an expression capture window, and then capture all expressions that the database tracked during this window. You perform the following steps:

1. Open an expression capture window, generate expressions, and then close the window:

   EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();
   -- Generate expressions for the database to track
   EXEC DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW();

2. Query DBA_EXPRESSION_STATISTICS (sample output included):

   COL OWNER FORMAT A6
   COL TABLE_NAME FORMAT A9
   COL COUNT FORMAT 99999
   COL CREATED FORMAT A10
   COL EXPRESSION_TEXT FORMAT A29

   SELECT OWNER, TABLE_NAME, EVALUATION_COUNT AS COUNT,
       CREATED, EXPRESSION_TEXT
   FROM DBA_EXPRESSION_STATISTICS
   WHERE SNAPSHOT = 'WINDOW'
   AND OWNER = 'SH';

<table>
<thead>
<tr>
<th>OWNER</th>
<th>TABLE_NAME</th>
<th>COUNT</th>
<th>CREATED</th>
<th>EXPRESSION_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>SALES</td>
<td>4702</td>
<td>09-OCT-17</td>
<td>&quot;QUANTITY_SOLD&quot;</td>
</tr>
<tr>
<td>SH</td>
<td>SALES</td>
<td>4702</td>
<td>09-OCT-17</td>
<td>&quot;QUANTITY_SOLD&quot;*&quot;AMOUNT_SOLD&quot;</td>
</tr>
<tr>
<td>SH</td>
<td>SALES</td>
<td>4702</td>
<td>09-OCT-17</td>
<td>&quot;PROD_ID&quot;</td>
</tr>
<tr>
<td>SH</td>
<td>SALES</td>
<td>4702</td>
<td>09-OCT-17</td>
<td>&quot;CUST_ID&quot;</td>
</tr>
<tr>
<td>SH</td>
<td>SALES</td>
<td>4702</td>
<td>09-OCT-17</td>
<td>&quot;CHANNEL_ID&quot;</td>
</tr>
<tr>
<td>SH</td>
<td>SALES</td>
<td>4702</td>
<td>09-OCT-17</td>
<td>&quot;AMOUNT_SOLD&quot;</td>
</tr>
</tbody>
</table>
The preceding query shows both the columns tracked in the ESS and the expressions captured during the window for queries in the sh schema. During the most recent window, the database captured one expression: QUANTITY_SOLD*AMOUNT_SOLD.

3. Use IME_CAPTURE_EXPRESSIONS to make the database consider all expressions in the current window for materialization:

```sql
EXEC DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS('WINDOW');
```

4. Query DBA_IM_EXPRESSIONS (sample output included):

```sql
COL OWNER FORMAT a6
COL TABLE_NAME FORMAT a9
COL COLUMN_NAME FORMAT a25
SET LONG 50
SET LINESIZE 150

SELECT OWNER, TABLE_NAME, COLUMN_NAME, SQL_EXPRESSION
FROM   DBA_IM_EXPRESSIONS;

OWNER  TABLE_NAME COLUMN_NAME               SQL_EXPRESSION
------ --------- ------------------------- -----------------------------
SH     SALES     SYS_IME000100000025201B   "QUANTITY_SOLD"**"AMOUNT_SOLD"
```

The preceding output shows all virtual columns that were added to the table and marked INMEMORY as part of the latest IME_CAPTURE_EXPRESSIONS invocation. The database gradually populates the captured expressions into the IM column store when it repopulates different IMCUs of the table.

5. Execute the following procedure to explicitly force a population of all captured IM expressions:

```sql
EXEC DBMS_INMEMORY_ADMIN.IME_POPULATE_EXPRESSIONS();
```

Note that you can populate IM expressions from a specific table by executing the DBMS_INMEMORY.REPOPULATE procedure with the force parameter set to TRUE.

**Example 85-4  Capturing Expressions for the Past Day**

The following program captures expressions tracked during the last 24 hours:

```sql
EXEC DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS('CURRENT');
```

### 85.4.8 IME_CLOSE_CAPTURE_WINDOW Procedure

This procedure signals the end of the current expression capture window.

**Syntax**

```sql
DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW();
```
Usage Notes

On invocation of this procedure, the optimizer saves all gathered statistics to disk, and essentially freezes the expressions tracked in the window. The database preserves the statistics captured in this window until a new expression capture window is opened, at which point the database purges the statistics captured in the previous window.

Example 85-5  Example

This example opens an expression capture window, and then issues IME_CAPTURE_EXPRESSIONS('WINDOW') so that the database considers all expressions in the current window for materialization. Finally, the example closes the window.

EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();
-- Generate expressions so that the database can track them
EXEC DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW();
EXEC DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS('WINDOW');

85.4.9 IME_DROP_ALL_EXPRESSIONS Procedure

This procedure drops all SYS_IME expression virtual columns in the database.

Syntax

DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS();

Usage Notes

The IME_DROP_ALL_EXPRESSIONS procedure drops all SYS_IME columns from all tables, regardless of whether they have the INMEMORY attribute. In effect, the procedure acts as a database-wide reset button.

Using IME_DROP_ALL_EXPRESSIONS triggers a drop of all IMEUs and IMCUs for segments that have SYS_IME columns. For example, if 50 populated tables have one SYS_IME column each, then IME_DROP_ALL_EXPRESSIONS removes all 50 tables from the IM column store. To populate these segments again, you must use the DBMS_INMEMORY.POPULATE procedure or perform a full table scan.

85.4.10 IME_GET_CAPTURE_STATE Procedure

This procedure returns the current capture state of the expression capture window and the timestamp of the most recent modification.

Syntax

DBMS_INMEMORY_ADMIN.IME_GET_CAPTURE_STATE(
    p_capture_state  OUT  VARCHAR2,
    p_last_modified  OUT  TIMESTAMP);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_capture_state</td>
<td>Describes the current state of the expression capture window.</td>
</tr>
<tr>
<td></td>
<td>The following states are possible:</td>
</tr>
<tr>
<td></td>
<td>• OPEN — Indicates that the window is open.</td>
</tr>
<tr>
<td></td>
<td>• CLOSED — Indicates that the window is closed.</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT — Indicates that the window has not been used. It is equivalent</td>
</tr>
<tr>
<td></td>
<td>to the CLOSED state.</td>
</tr>
<tr>
<td>p_last_modified</td>
<td>Indicates the timestamp of the most recent action.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is useful for avoiding conflicting calls for `IME_OPEN_CAPTURE_WINDOW Procedure` and `IME_CLOSE_CAPTURE_WINDOW Procedure`. For example, if the current expression capture window state is OPEN, then you cannot open another window, and if the window state is CLOSED, then you cannot close a window.

Example 85-6 Determining the State of an Expression Capture Window

This example opens an expression capture window, and then determines its capture state.

```sql
EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();

VARIABLE b_state VARCHAR2(25)
VARIABLE b_time  VARCHAR2(10)
EXECUTE DBMS_INMEMORY_ADMIN.IME_GET_CAPTURE_STATE(:b_state, :b_time)
PRINT b_state b_time
```

The following sample output indicates that an expression capture window is currently open:

```
B_STATE
---------------------------
OPEN

B_TIME
---------------------------
09-OCT-17
```

85.4.11 `IME_OPEN_CAPTURE_WINDOW Procedure`

This procedure signals the beginning of an expression capture window.

Syntax

```sql
DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();
```

Security Model

Administrator privileges are required to execute this procedure.
Usage Notes

On invocation of this procedure, the optimizer begins a new window snapshot and starts tracking expressions that occur within this window. An expression capture window is global across all instances in an Oracle RAC database.

Conflicting actions are not permitted. For example, in an Oracle RAC database, opening expression capture window on instance 1 at time t0 and attempting to open another expression capture window on instance 2 at time t1 before closing the first window is a conflicting action. To obtain the current capture state and reduce the potential for conflicting procedure invocations, use the IME_GET_CAPTURE_STATE Procedure.

Example

This following program opens an expression capture window:

EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();

85.4.12 IME_POPULATE_EXPRESSIONS Procedure

This procedure forces the population of expressions captured in the latest invocation of DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS.

Syntax

DBMS_INMEMORY_ADMIN.IME_POPULATE_EXPRESSIONS();

Usage Notes

If you do not invoke this procedure, then the database gradually repopulates SYS_IME columns when their parent IMCUs are repopulated. If a table is not repopulated, then any new SYS_IME columns captured by the IME_CAPTURE_EXPRESSIONS procedure are not populated. IME_POPULATE_EXPRESSIONS solves this problem by forcing population.

Internally, the procedure invokes DBMS_INMEMORY.REPOPULATE for all tables that have SYS_IME columns with the INMEMORY attribute. To populate SYS_IME columns in a specified subset of tables, use DBMS_INMEMORY.REPOPULATE instead of IME_POPULATE_EXPRESSIONS.
The **DBMS_IOT** package creates a table into which references to the chained rows for an index-organized table can be placed using the **ANALYZE** command. **DBMS_IOT** can also create an exception table into which references to the rows of an index-organized table that violate a constraint can be placed during the **enable_constraint** operation.

**DBMS_IOT** is not loaded during database installation. To install **DBMS_IOT**, run `dbmsiotc.sql`, available in the **ADMIN** directory.

This chapter contains the following topics:

- Summary of DBMS_IOT Subprograms

---

**Note:**

With the introduction of logical-rowids for IOTs with Oracle Database Release 8.1, you no longer need to use the procedures contained in this package which is retained for backward compatibility only. It is however required for servers running with Oracle Database Release 8.0.

---

### 86.1 Summary of DBMS_IOT Subprograms

This table lists and briefly describes the **DBMS_IOT** subprograms.

**Table 86-1  DBMS_IOT Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BUILD_CHAIN_ROWS_TABLE</strong> Procedure</td>
<td>Creates a table into which references to the chained rows for an index-organized table can be placed using the <strong>ANALYZE</strong> command</td>
</tr>
<tr>
<td><strong>BUILD_EXCEPTIONS_TABLE</strong> Procedure</td>
<td>Creates an exception table into which rows of an index-organized table that violate a constraint can be placed</td>
</tr>
</tbody>
</table>

---

### 86.1.1 BUILD_CHAIN_ROWS_TABLE Procedure

This procedure creates a table into which references to the chained rows for an index-organized table can be placed using the **ANALYZE** command.

**Syntax**

```sql
DBMS_IOT.BUILD_CHAIN_ROWS_TABLE (
    owner               IN VARCHAR2,
    iot_name            IN VARCHAR2,
    chainrow_table_name IN VARCHAR2 default 'IOT_CHAINED_ROWS');
```
Parameters

Table 86-2  BUILD_CHAIN_ROWS_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the index-organized table.</td>
</tr>
<tr>
<td>iot_name</td>
<td>Index-organized table name.</td>
</tr>
<tr>
<td>chainrow_table_name</td>
<td>Intended name for the chained-rows table.</td>
</tr>
</tbody>
</table>

Usage Notes

You should create a separate chained-rows table for each index-organized table to accommodate its primary key.

Examples

CREATE TABLE l(a char(16), b char(16), c char(16), d char(240),
PRIMARY KEY(a,b,c)) ORGANIZATION INDEX pctthreshold 10 overflow;
EXECUTE DBMS_IOT.BUILD_CHAIN_ROWS_TABLE('SYS','L','LC');

A chained-row table is created with the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>CLUSTER_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>PARTITION_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>SUBPARTITION_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>HEAD_ROWID</td>
<td></td>
<td>ROWID</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>CHAR(16)</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>CHAR(16)</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>CHAR(16)</td>
</tr>
</tbody>
</table>

86.1.2 BUILD_EXCEPTIONS_TABLE Procedure

This procedure creates an exception table.

Rows of an index-organized table that violate a constraint can be placed into this table during the execution of the following SQL statements:

• ALTER TABLE ... ENABLE CONSTRAINT ... EXCEPTIONS INTO
• ALTER TABLE ... ADD CONSTRAINT ... EXCEPTIONS INTO

Syntax

DBMS_IOT.BUILD_EXCEPTIONS_TABLE (  
  owner IN VARCHAR2,
  iot_name IN VARCHAR2,
  exceptions_table_name IN VARCHAR2 default 'IOT_EXCEPTIONS');
Parameters

Table 86-3  BUILD_EXCEPTIONS_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the index-organized table.</td>
</tr>
<tr>
<td>iot_name</td>
<td>Index-organized table name.</td>
</tr>
<tr>
<td>exceptions_table_name</td>
<td>Intended name for exception-table.</td>
</tr>
</tbody>
</table>

Usage Notes

You should create a separate exception table for each index-organized table to accommodate its primary key.

Examples

EXECUTE DBMS_IOT.BUILD_EXCEPTIONS_TABLE('SYS','L','LE');

An exception table for the preceding index-organized table with the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW_ID</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>OWNER</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>CHAR(16)</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>CHAR(16)</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>CHAR(16)</td>
</tr>
</tbody>
</table>
The DBMS_JAVA package provides a PL/SQL interface for accessing database functionality from Java.

For a complete description of this package within the context of DBMS_JAVA, see DBMS_JAVA in the Oracle Database Java Developer's Guide.
The **DBMS_JOB** package schedules and manages jobs in the job queue.

---

**Note:**

The **DBMS_JOB** package has been superseded by the **DBMS_SCHEDULER** package, and support for **DBMS_JOB** might be removed in future releases of Oracle Database. In particular, if you are administering jobs to manage system load, you are encouraged to disable **DBMS_JOB** by revoking the package execution privilege for users.

For more information, see **DBMS_SCHEDULER** and "Moving from **DBMS_JOB** to **DBMS_SCHEDULER**" in the *Oracle Database Administrator's Guide*.

---

This chapter contains the following topics:

- Security Model
- Operational Notes
- Summary of DBMS_JOB Subprograms

### 88.1 DBMS_JOB Security Model

No specific system privileges are required to use **DBMS_JOB**. No system privileges are available to manage **DBMS_JOB**. Jobs cannot be altered or deleted other than jobs owned by the user. This is true for all users including those users granted DBA privileges.

You can execute procedures that are owned by the user or for which the user is explicitly granted **EXECUTE**. However, procedures for which the user is granted the execute privilege through roles cannot be executed.

Note that, once a job is started and running, there is no easy way to stop the job.

---

### 88.2 DBMS_JOB Operational Notes

These notes describe stopping a job, and working with Oracle Real Application Clusters.

**Stopping a Job**

Note that, once a job is started and running, there is no easy way to stop the job.

**Working with Oracle Real Application Clusters**

**DBMS_JOB** supports multi-instance execution of jobs. By default jobs can be executed on any instance, but only one single instance will execute the job. In addition, you can force instance binding by binding the job to a particular instance. You implement instance binding by specifying an instance number to the instance affinity parameter. Note, however, that in
Oracle Database 10g Release 1 (10.1) instance binding is not recommended. Service affinity is preferred. This concept is implemented in the `DBMS_SCHEDULER` package.

The following procedures can be used to create, alter or run jobs with instance affinity. Note that not specifying affinity means any instance can run the job.

- `DBMS_JOB.SUBMIT`
- `DBMS_JOB.INSTANCE`
- `DBMS_JOB.CHANGE`
- `DBMS_JOB.RUN`

### DBMS_JOB.SUBMIT

To submit a job to the job queue, use the following syntax:

```sql
DBMS_JOB.SUBMIT(
    job       OUT    BINARY_INTEGER,
    what      IN     VARCHAR2,
    next_date IN     DATE DEFAULT SYSDATE,
    interval  IN     VARCHAR2 DEFAULT 'NULL',
    no_parse  IN     BOOLEAN DEFAULT FALSE,
    instance  IN     BINARY_INTEGER DEFAULT ANY_INSTANCE,
    force     IN     BOOLEAN DEFAULT FALSE);
```

Use the parameters `instance` and `force` to control job and instance affinity. The default value of `instance` is 0 (zero) to indicate that any instance can execute the job. To run the job on a certain instance, specify the `instance` value. Oracle displays error ORA-23319 if the `instance` value is a negative number or NULL.

The `force` parameter defaults to false. If `force` is TRUE, any positive integer is acceptable as the job instance. If `force` is FALSE, the specified instance must be running, or Oracle displays error number ORA-23428.

### DBMS_JOB.INSTANCE

To assign a particular instance to execute a job, use the following syntax:

```sql
DBMS_JOB.INSTANCE(  JOB IN BINARY_INTEGER,
    instance                IN BINARY_INTEGER,
    force                   IN BOOLEAN DEFAULT FALSE);
```

The `FORCE` parameter in this example defaults to FALSE. If the instance value is 0 (zero), job affinity is altered and any available instance can execute the job despite the value of `force`. If the `INSTANCE` value is positive and the `FORCE` parameter is FALSE, job affinity is altered only if the specified instance is running, or Oracle displays error ORA-23428.

If the `force` parameter is TRUE, any positive integer is acceptable as the job instance and the job affinity is altered. Oracle displays error ORA-23319 if the `instance` value is negative or NULL.

### DBMS_JOB.CHANGE

To alter user-definable parameters associated with a job, use the following syntax:

```sql
DBMS_JOB.CHANGE(  JOB IN BINARY_INTEGER,
    what                  IN VARCHAR2 DEFAULT NULL,
    next_date             IN DATE DEFAULT NULL,
    interval              IN VARCHAR2 DEFAULT NULL,
```

---

88-2
Two parameters, instance and force, appear in this example. The default value of instance is null indicating that job affinity will not change.

The default value of force is FALSE. Oracle displays error ORA-23428 if the specified instance is not running and error ORA-23319 if the instance number is negative.

DBMS_JOB.RUN

The force parameter for DBMS_JOB.RUN defaults to FALSE. If force is TRUE, instance affinity is irrelevant for running jobs in the foreground process. If force is FALSE, the job can run in the foreground only in the specified instance. Oracle displays error ORA-23428 if force is FALSE and the connected instance is the incorrect instance.

```sql
DBMS_JOB.RUN(
    job IN BINARY_INTEGER,
    force IN BOOLEAN DEFAULT FALSE);
```

### 88.3 Summary of DBMS_JOB Subprograms

This table lists the DBMS_JOB subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROKEN Procedure</td>
<td>Enables job execution</td>
</tr>
<tr>
<td>CHANGE Procedure</td>
<td>Alters any of the user-definable parameters associated with a job</td>
</tr>
<tr>
<td>INSTANCE Procedure</td>
<td>Assigns a job to be run by a instance</td>
</tr>
<tr>
<td>INTERVAL Procedure</td>
<td>Alters the interval between executions for a specified job</td>
</tr>
<tr>
<td>NEXT_DATE Procedure</td>
<td>Alters the next execution time for a specified job</td>
</tr>
<tr>
<td>REMOVE Procedure</td>
<td>Removes specified job from the job queue</td>
</tr>
<tr>
<td>RUN Procedure</td>
<td>Forces a specified job to run</td>
</tr>
<tr>
<td>SUBMIT Procedure</td>
<td>Submits a new job to the job queue</td>
</tr>
<tr>
<td>USER_EXPORT Procedures</td>
<td>Re-creates a given job for export, or re-creates a given job for export with instance affinity</td>
</tr>
<tr>
<td>WHAT Procedure</td>
<td>Alters the job description for a specified job</td>
</tr>
</tbody>
</table>

### 88.3.1 BROKEN Procedure

This procedure sets the broken flag. Broken jobs are never run.

**Syntax**

```sql
DBMS_JOB.BROKEN (
    job IN BINARY_INTEGER,
    broken IN BOOLEAN,
    next_date IN DATE DEFAULT SYSDATE);
```
Parameters

Table 88-2 BROKEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>broken</td>
<td>Sets the job as broken or not broken. TRUE sets it as broken; FALSE sets it as not broken.</td>
</tr>
<tr>
<td>next_date</td>
<td>Next date when the job will be run.</td>
</tr>
</tbody>
</table>

Note:
If you set job as broken while it is running, Oracle resets the job's status to normal after the job completes. Therefore, only execute this procedure for jobs that are not running.

Usage Notes

- Your job will not be available for processing by the job queue in the background until it is committed.
- If a job fails 16 times in a row, Oracle automatically sets it as broken and then stops trying to run it.

88.3.2 CHANGE Procedure

This procedure changes any of the fields a user can set in a job.

Syntax

```sql
DBMS_JOB.CHANGE (job IN BINARY_INTEGER,
                what IN VARCHAR2,
                next_date IN DATE,
                interval IN VARCHAR2,
                instance IN BINARY_INTEGER DEFAULT NULL,
                force IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 88-3 CHANGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>what</td>
<td>PL/SQL procedure to run.</td>
</tr>
<tr>
<td>next_date</td>
<td>Next date when the job will be run.</td>
</tr>
</tbody>
</table>
Table 88-3  (Cont.) CHANGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval</td>
<td>Date function; evaluated immediately before the job starts running.</td>
</tr>
<tr>
<td>instance</td>
<td>When a job is submitted, specifies which instance can run the job. This defaults to NULL, which indicates that instance affinity is not changed.</td>
</tr>
<tr>
<td>force</td>
<td>If this is FALSE, then the specified instance (to which the instance number change) must be running. Otherwise, the routine raises an exception. If this is TRUE, then any positive integer is acceptable as the job instance.</td>
</tr>
</tbody>
</table>

Usage Notes

- Your job will not be available for processing by the job queue in the background until it is committed.
- The parameters instance and force are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.
- If the parameters what, next_date, or interval are NULL, then leave that value as it is.

Example

BEGIN
  DBMS_JOB.CHANGE(14144, null, null, 'sysdate+3');
  COMMIT;
END;

88.3.3 INSTANCE Procedure

This procedure changes job instance affinity.

Syntax

```sql
DBMS_JOB.INSTANCE (
  job        IN BINARY_INTEGER,
  instance   IN BINARY_INTEGER,
  force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 88-4  INSTANCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>instance</td>
<td>When a job is submitted, a user can specify which instance can run the job.</td>
</tr>
</tbody>
</table>
Table 88-4  (Cont.) INSTANCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>If this is TRUE, then any positive integer is acceptable as the job instance. If this is FALSE (the default), then the specified instance must be running; otherwise the routine raises an exception.</td>
</tr>
</tbody>
</table>

Usage Notes

Your job will not be available for processing by the job queue in the background until it is committed.

88.3.4 INTERVAL Procedure

This procedure changes how often a job runs.

Syntax

```sql
DBMS_JOB.INTERVAL (
    job       IN  BINARY_INTEGER,
    interval  IN  VARCHAR2);
```

Parameters

Table 88-5  INTERVAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>interval</td>
<td>Date function, evaluated immediately before the job starts running.</td>
</tr>
</tbody>
</table>

Usage Notes

- If the job completes successfully, then this new date is placed in next_date. interval is evaluated by plugging it into the statement select interval into next_date from dual;
- The interval parameter must evaluate to a time in the future. Legal intervals include:

<table>
<thead>
<tr>
<th>Interval</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'sysdate + 7'</td>
<td>Run once a week.</td>
</tr>
<tr>
<td>'next_day(sysdate,'''TUESDAY'''')'</td>
<td>Run once every Tuesday.</td>
</tr>
<tr>
<td>'null'</td>
<td>Run only once.</td>
</tr>
</tbody>
</table>

- If interval evaluates to NULL and if a job completes successfully, then the job is automatically deleted from the queue.
- Your job will not be available for processing by the job queue in the background until it is committed.
88.3.5 NEXT_DATE Procedure

This procedure changes when an existing job next runs.

Syntax

```sql
DBMS_JOB.NEXT_DATE (
    job       IN  BINARY_INTEGER,
    next_date IN  DATE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>next_date</td>
<td>Date of the next refresh: it is when the job will be automatically run, assuming there are background processes attempting to run it.</td>
</tr>
</tbody>
</table>

Usage Notes

Your job will not be available for processing by the job queue in the background until it is committed.

88.3.6 REMOVE Procedure

This procedure removes an existing job from the job queue. This currently does not stop a running job.

Syntax

```sql
DBMS_JOB.REMOVE (
    job       IN  BINARY_INTEGER );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
</tbody>
</table>

Usage Notes

Your job will not be available for processing by the job queue in the background until it is committed.

Example

```sql
BEGIN
    DBMS_JOB.REMOVE(14144);
END;
```
88.3.7 RUN Procedure

This procedure runs job JOB now. It runs it even if it is broken.

Running the job recomputes next_date. See data dictionary view USER_JOBS or DBA_JOBS.

Syntax

```sql
DBMS_JOB.RUN (
  job       IN  BINARY_INTEGER,
  force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>force</td>
<td>If this is TRUE, then instance affinity is irrelevant for running jobs in the foreground process. If this is FALSE, then the job can be run in the foreground only in the specified instance.</td>
</tr>
</tbody>
</table>

Example

```sql
EXECUTE DBMS_JOB.RUN(14144);
```

⚠️ WARNING:

This re-initializes the current session’s packages.

Exceptions

An exception is raised if force is FALSE, and if the connected instance is the wrong one.

88.3.8 SUBMIT Procedure

This procedure submits a new job. It chooses the job from the sequence sys.jobseq.

Syntax

```sql
DBMS_JOB.SUBMIT (
  job       OUT BINARY_INTEGER,
  what      IN  VARCHAR2,
  next_date IN  DATE DEFAULT SYSDATE,
  interval  IN  VARCHAR2 DEFAULT 'null',
  no_parse  IN  BOOLEAN DEFAULT FALSE,
```
instance IN BINARY_INTEGER DEFAULT any_instance,
force IN BOOLEAN DEFAULT FALSE);

Parameters

Table 88-9 SUBMIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>what</td>
<td>PL/SQL text o the job to be run. This must be a valid PL/SQL statement or block of code. For example, to run a stored procedure P, you could pass the string P; (with the semi-colon) to this routine. The SQL that you submit in the what parameter is wrapped in the following PL/SQL block:</td>
</tr>
</tbody>
</table>

```plaintext
DECLARE
  job BINARY_INTEGER := :job;
  next_date DATE := :mydate;
  broken BOOLEAN := FALSE;
BEGIN
  WHAT
    :mydate := next_date;
    IF broken THEN :b := 1; ELSE :b := 0; END IF;
END;
```

Ensure that you include the ; semi-colon with the statement. |
| next_date | Next date when the job will be run. |
| interval  | Date function that calculates the next time to run the job. The default is NULL. This must evaluate to a either a future point in time or NULL. |
| no_parse  | A flag. The default is FALSE. If this is set to FALSE, then Oracle parses the procedure associated with the job. If this is set to TRUE, then Oracle parses the procedure associated with the job the first time that the job is run. For example, if you want to submit a job before you have created the tables associated with the job, then set this to TRUE. |
| instance  | When a job is submitted, specifies which instance can run the job. |
| force     | If this is TRUE, then any positive integer is acceptable as the job instance. If this is FALSE (the default), then the specified instance must be running; otherwise the routine raises an exception. |

Usage Notes

- Your job will not be available for processing by the job queue in the background until it is committed.
- The parameters instance and force are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.

Example

This submits a new job to the job queue. The job calls the procedure DBMS_DDL.ANALYZE_OBJECT to generate optimizer statistics for the table DQUON ACCOUNTS.
statistics are based on a sample of half the rows of the ACCOUNTS table. The job is run every 24 hours:

```sql
VARIABLE jobno number;
BEGIN
    DBMS_JOB.SUBMIT(:jobno,
        'dbms_ddl.analyze_object('''TABLE'',
            ''DQUON''', ''ACCOUNTS'',
            ''ESTIMATE'', NULL, 50);
    SYSDATE, 'SYSDATE + 1');
    COMMIT;
END;
/
Statement processed.
```

88.3.9 USER_EXPORT Procedures

There are two overloaded procedures. The first produces the text of a call to re-create the given job. The second alters instance affinity (8i and after) and preserves the compatibility.

**Syntax**

```sql
DBMS_JOB.USER_EXPORT (  
    job IN BINARY_INTEGER,  
    mycall IN OUT VARCHAR2);

DBMS_JOB.USER_EXPORT (  
    job IN BINARY_INTEGER,  
    mycall IN OUT VARCHAR2,  
    myinst IN OUT VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>mycall</td>
<td>Text of a call to re-create the given job.</td>
</tr>
<tr>
<td>myinst</td>
<td>Text of a call to alter instance affinity.</td>
</tr>
</tbody>
</table>

88.3.10 WHAT Procedure

This procedure changes what an existing job does, and replaces its environment.

**Syntax**

```sql
DBMS_JOB.WHAT (  
    job IN BINARY_INTEGER,  
    what IN VARCHAR2);
```
Parameters

Table 88-11  WHAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>System-assigned ID of the job being run. To find this ID, query the JOB column of the USER_JOBS or DBA_JOBS view.</td>
</tr>
<tr>
<td>what</td>
<td>PL/SQL procedure to run.</td>
</tr>
</tbody>
</table>

Usage Notes

• Your job will not be available for processing by the job queue in the background until it is committed.

• Some legal values of what (assuming the routines exist) are:
  - 'myproc(''10-JAN-82'', next_date, broken);'
  - 'scott.emppackage.give_raise(''JENKINS'', 30000.00);'
  - 'dbms_job.remove(job);'
The **DBMS_JSON** package provides an interface for data-guide operations.

This chapter contains the following topics:

- DBMS_JSON Overview
- DBMS_JSON Security Model
- DBMS_JSON Constants
- Summary of DBMS_JSON Subprograms

### 89.1 DBMS_JSON Overview

Package **DBMS_JSON** provides subprograms for manipulating JavaScript Object Notation (JSON) data that is stored in Oracle Database.

### 89.2 DBMS_JSON Security Model

**PUBLIC** is granted the **EXECUTE** privilege on package **DBMS_JSON**. Its subprograms execute with invoker's rights privileges.

### 89.3 DBMS_JSON Constants

The **DBMS_JSON** package uses these constants to define the JSON schema types and data-guide formatting options.

**Table 89-1**  **DBMS_JSON Constants Defined for JSON Data-Guide Formatting**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT_FLAT</td>
<td>2</td>
<td>Display flat format</td>
</tr>
<tr>
<td>FORMAT_HIERARCHICAL</td>
<td>1</td>
<td>Display hierarchical format</td>
</tr>
<tr>
<td>PRETTY</td>
<td>1</td>
<td>Use appropriate indentation to improve readability</td>
</tr>
</tbody>
</table>

**Table 89-2**  **DBMS_JSON Constants for JSON Schema Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ARRAY</td>
<td>NUMBER(2)</td>
<td>6</td>
<td>A JSON array</td>
</tr>
<tr>
<td>TYPE_BOOLEAN</td>
<td>NUMBER(2)</td>
<td>2</td>
<td>A JSON boolean</td>
</tr>
<tr>
<td>TYPE_OBJECT</td>
<td>NUMBER(2)</td>
<td>5</td>
<td>A JSON object</td>
</tr>
<tr>
<td>TYPE_NULL</td>
<td>NUMBER(2)</td>
<td>1</td>
<td>The JSON NULL value</td>
</tr>
<tr>
<td>TYPE_NUMBER</td>
<td>NUMBER(2)</td>
<td>3</td>
<td>A JSON number</td>
</tr>
<tr>
<td>TYPE_STRING</td>
<td>NUMBER(2)</td>
<td>4</td>
<td>A JSON string</td>
</tr>
</tbody>
</table>
89.4 Summary of DBMS_JSON Subprograms

This table lists the DBMS_JSON subprograms and briefly describes them.

### DBMS_JSON Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_VIRTUAL_COLUMNS Procedure</td>
<td>Add virtual columns based on data-guide information. This has no effect when running on the shard catalog server — no virtual column is added.</td>
</tr>
<tr>
<td>CREATE_VIEW Procedure</td>
<td>Create a view with relational columns and scalar JSON fields as specified in a data guide.</td>
</tr>
<tr>
<td>CREATE_VIEW_ON_PATH Procedure</td>
<td>Create a view based on data-guide information, with relational columns, top-level scalar types, and fully expanded sub-tree under a given path. When running on the shard catalog server this raises an error stating that the data guide is empty.</td>
</tr>
<tr>
<td>DROP_VIRTUAL_COLUMNS Procedure</td>
<td>Drop virtual columns created by procedure add_virtual_columns. This has no effect when running on the shard catalog server.</td>
</tr>
<tr>
<td>GET_INDEX_DATAGUIDE Function</td>
<td>Get JSON data guide from a data guide-enabled JSON search index. When running on the shard catalog server this returns a single empty row as result.</td>
</tr>
<tr>
<td>RENAME_COLUMN Procedure</td>
<td>Set the preferred name for a view column or a virtual column creating using a data guide. This has no effect when running on the shard catalog server.</td>
</tr>
</tbody>
</table>

**Note:**

In the context of sharding, each individual shard maintains its own data-guide information, which is obtained from the JSON documents stored in that shard. When running on individual shard, procedures in this package that use data-guide information use only the information that is maintained for that shard.

89.4.1 ADD_VIRTUAL_COLUMNS Procedure

This procedure adds virtual columns based on the data guide.

The virtual column name is the value of `o:preferred_vc_name` in the data guide. The procedure ignores JSON objects, arrays, and fields under arrays in the data guide.
Before it adds virtual columns, procedure \texttt{add\_virtual\_columns} first drops any existing virtual columns that were projected from fields in the same JSON column by a previous invocation of \texttt{add\_virtual\_columns} or by data-guide change-trigger procedure \texttt{add\_vc} (in effect, it does what procedure \texttt{DBMS\_JSON\_drop\_virtual\_columns} does).

\begin{itemize}
\item \textbf{See Also:}
\begin{itemize}
\item \texttt{DROP\_VIRTUAL\_COLUMNS} Procedure
\item \textit{Oracle Database JSON Developer’s Guide}
\end{itemize}
\end{itemize}

\textbf{Syntax}

\begin{verbatim}
DBMS\_JSON\_.ADD\_VIRTUAL\_COLUMNS (  
tablename  IN  VARCHAR2,  
jcolname   IN  VARCHAR2,  
dataguide  IN  CLOB);
\end{verbatim}

For the following signature you must have a data guide-enabled search index on the JSON column. (This is not needed for the previous signature.)

\begin{verbatim}
DBMS\_JSON\_.ADD\_VIRTUAL\_COLUMNS (  
tablename  IN  VARCHAR2,  
jcolname   IN  VARCHAR2,  
frequency      NUMBER    DEFAULT 0,  
hidden         BOOLEAN   DEFAULT FALSE);
\end{verbatim}

\textbf{Parameters}

\begin{table}[h]
\centering
\begin{tabular}{|c|p{15cm}|}
\hline
\textbf{Parameter} & \textbf{Description} \\
\hline
\texttt{tablename} & Name of the table containing JSON column \texttt{jcolname}. \\
\texttt{jcolname} & Name of the JSON column in table \texttt{tablename} that contains the data from which to create the virtual column. \\
\texttt{frequency} & Sets the minimum frequency threshold to display JSON columns. A frequency of 0 means display all JSON columns. Also, all JSON columns are displayed if statistics have not been collected, effectively overriding any value set by this parameter. \\
\texttt{hidden} & TRUE means the added virtual column is hidden; FALSE means it is not. The default is FALSE. \\
\texttt{dataguide} & The data guide. When \texttt{o:hidden} in the data guide for a particular JSON field is set to TRUE, the corresponding virtual column is added as a hidden column. The default value of \texttt{o:hidden} is FALSE. \\
\hline
\end{tabular}
\end{table}

\textbf{Usage Notes}

Procedure \texttt{DBMS\_STATS\_GATHER\_STATS} collects statistics in the data guide. If the frequency statistic has not been collected, \texttt{frequency} is \texttt{NULL}. Setting the frequency to a value greater than zero means do not include columns for which there are no frequency statistics collected.
(statistic is NULL), unless DBMS_STATS.GATHER_STATS has never been executed. In that case, the frequency parameter is ignored and all columns are displayed in the view.

89.4.2 CREATE_VIEW Procedure

This procedure creates a view with relational columns, using scalar JSON fields as specified in the data guide. A data guide-enabled JSON search index is not required for this procedure; the data guide is passed to the procedure.

See Also:

Oracle Database JSON Developer’s Guide

Syntax

(Optional) Enter syntax information here.

PROCEDURE CREATE_VIEW (
    viewname VARCHAR2,
    tablename VARCHAR2,
    jcolname VARCHAR2,
    dataguide CLOB);

Parameters

Table 89-4  DBMS_JSON.CREATE_VIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewname</td>
<td>Name of the view.</td>
</tr>
<tr>
<td>tablename</td>
<td>Name of the table containing JSON column jcolname.</td>
</tr>
<tr>
<td>jcolname</td>
<td>Name of the JSON column in table tablename that is used to create the view.</td>
</tr>
<tr>
<td>dataguide</td>
<td>The data guide.</td>
</tr>
</tbody>
</table>

89.4.3 CREATE_VIEW_ON_PATH Procedure

This procedure creates a view with relational columns, using top-level scalar values and the scalar values in the expanded sub-tree under a given path. The JSON column must have a data guide-enabled search index.

See Also:

Oracle Database JSON Developer’s Guide
Syntax

PROCEDURE CREATE_VIEW_ON_PATH(
    viewname VARCHAR2,
    tablename VARCHAR2,
    jcolname VARCHAR2,
    path VARCHAR2,
    frequency NUMBER DEFAULT 0);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewname</td>
<td>Name of the view.</td>
</tr>
<tr>
<td>tablename</td>
<td>Name of the table containing JSON column jcolname.</td>
</tr>
<tr>
<td>jcolname</td>
<td>Name of the JSON column in table tablename that is used to create the view. The column must have a data guide-enabled JSON search index, or else an error is raised.</td>
</tr>
<tr>
<td>path</td>
<td>The path of the JSON field to be expanded. It uses JSON path-expression syntax. It expands the descendants under the specified path, and creates view columns for each scalar value in the resulting sub-tree. The path $ creates a view starting from the JSON document root.</td>
</tr>
<tr>
<td>frequency</td>
<td>The minimum frequency threshold for displaying the JSON columns. A frequency of 0 means display all JSON columns. All JSON columns are also displayed if statistics have not been collected, effectively overriding any value set by this parameter. The view only displays JSON fields with frequency greater than the given frequency. It does not display JSON fields added after collecting statistics if the given frequency is greater than 0, if their statistic columns are NULL.</td>
</tr>
</tbody>
</table>

89.4.4 DROP_VIRTUAL_COLUMNS Procedure

Drop all virtual columns that were added using PL/SQL procedure DBMS_JSON.add_virtual_columns or using data-guide change-trigger procedure add_vc.

See Also:

- ADD_VIRTUAL_COLUMNS Procedure
- Oracle Database JSON Developer’s Guide

Syntax

PROCEDURE DROP_VIRTUAL_COLUMNS(
    tablename VARCHAR2,
    jcolname VARCHAR2);
89.4.5 GET_INDEX_DATAGUIDE Function

GET_INDEX_DATAGUIDE gets JSON data guide from data guide-enabled JSON search index.

See Also:

Oracle Database JSON Developer’s Guide

Syntax

FUNCTION GET_INDEX_DATAGUIDE(
    tablename VARCHAR2,
    jcolname VARCHAR2,
    format NUMBER,
    pretty NUMBER DEFAULT 0)
RETURN CLOB;

Parameters

Table 89-7  DBMS_JSON.GET_INDEX_DATAGUIDE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Name of the table containing JSON column jcolname.</td>
</tr>
<tr>
<td>jcolname</td>
<td>Name of the JSON column in table tablename that has a data guide-enabled JSON search index.</td>
</tr>
<tr>
<td>format</td>
<td>The data-guide format:</td>
</tr>
<tr>
<td></td>
<td>· FORMAT_HIERARCHICAL — hierarchical format</td>
</tr>
<tr>
<td></td>
<td>· FORMAT_FLAT — flat format</td>
</tr>
<tr>
<td>pretty</td>
<td>A value of DBMS_JSONPRETTY means pretty-print the data guide, using indentation to improve readability.</td>
</tr>
</tbody>
</table>

Example 89-1  Example Get Data Guide in Hierarchical Pretty Format

This example returns the data guide in hierarchical format.

SELECT DBMS_JSON.GET_INDEX_DATAGUIDE('T1', 'PO', DBMS_JSON.FORMAT_HIERARCHICAL, DBMS_JSONPRETTY) FROM DUAL;
89.4.6 RENAME_COLUMN Procedure

This procedure sets the preferred name for a JSON column, to be used by the create view, or add virtual columns procedure.

See Also:

Oracle Database JSON Developer’s Guide

Syntax

PROCEDURE RENAME_COLUMN(
    tablename VARCHAR2,
    jcolname VARCHAR2,
    path VARCHAR2,
    type NUMBER,
    preferred_name VARCHAR2);

Parameters

Table 89-8  RENAME_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Name of the table containing JSON column jcolname.</td>
</tr>
<tr>
<td>jcolname</td>
<td>Name of the JSON column in table tablename. It must have a data guide-enabled JSON search index, or else an error is raised.</td>
</tr>
<tr>
<td>path</td>
<td>Path to the JSON field on which to set the preferred column name.</td>
</tr>
<tr>
<td>type</td>
<td>The type of the JSON field targeted by path. Two JSON fields can have the same path if they are of different types. Possible values:</td>
</tr>
<tr>
<td></td>
<td>• TYPE_NULL</td>
</tr>
<tr>
<td></td>
<td>• TYPE_STRING</td>
</tr>
<tr>
<td></td>
<td>• TYPE_NUMBER</td>
</tr>
<tr>
<td></td>
<td>• TYPE_BOOLEAN</td>
</tr>
<tr>
<td></td>
<td>• TYPE_OBJECT</td>
</tr>
<tr>
<td></td>
<td>• TYPE_ARRAY</td>
</tr>
<tr>
<td>preferred_name</td>
<td>Preferred name for the JSON field specified by path. If there is a name conflict, a system generated name is used instead.</td>
</tr>
</tbody>
</table>

Example 89-2  Example Renaming a Column

This example renames a field to item_name.

EXEC DBMS_JSON.RENAME_COLUMN('T1', 'PO', '$.purchaseOrder.items.name', DBMS_JSON.TYPE_STRING, 'item_name');
The DBMS_LDAP package lets you access data from LDAP servers.

For a complete description of this package within the context of Oracle Internet Directory, see DBMS_LDAP in the Oracle Identity Management Application Developer's Guide.
The DBMS_LDAP_UTL package contains the Oracle Extension utility functions.

For a complete description of this package within the context of Oracle Internet Directory, see DBMS_LDAP_UTL in the Application Developer's Guide for Oracle Identity Management.
92

DBMS_LIBCACHE

The DBMS_LIBCACHE package consists of one subprogram that prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution. The value of compiling the cache of an instance is to prepare the information the application requires to execute in advance of failover or switchover.

This chapter contains the following topics:

• Overview
• Security Model
• Summary of DBMS_LIBCACHE Subprograms

92.1 DBMS_LIBCACHE Overview

Compiling a shared cursor consists of open, parse, and bind operations, plus the type-checking and execution plan functions performed at the first execution. All of these steps are executed in advance by the package DBMS_LIBCACHE for SELECT statements.

The open and parse functions are executed in advance for PL/SQL and DML. For PL/SQL, executing the parse phase has the effect of loading all library cache heaps other than the MCODE.

92.2 DBMS_LIBCACHE Security Model

To execute DBMS_LIBCACHE you must directly access the same objects as do SQL statements. You can best accomplish this by utilizing the same user id as the original system on the remote system.

When there are multiple schema users, DBMS_LIBCACHE should be called for each.

Alternatively, DBMS_LIBCACHE may be called with the generic user PARSER. However, this user cannot parse the SQL that uses objects with access granted through roles. This is a standard PL/SQL security limitation.

92.3 Summary of DBMS_LIBCACHE Subprograms

The DBMS_LIBCACHE package includes the COMPILE_FROM_REMOTE procedure subprogram.

Table 92-1  DBMS_LIBCACHE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPILE_FROM_REMOTE Procedure</td>
<td>Extracts SQL in batch from the source instance and compiles the SQL at the target instance</td>
</tr>
</tbody>
</table>
92.3.1 COMPILE_FROM_REMOTE Procedure

This procedure extracts SQL in batch from the source instance and compiles the SQL at the target instance.

Syntax

```sql
DBMS_LIBCACHE.COMPILE_FROM_REMOTE (  
    p_db_link                 IN     dbms_libcache$def.db_link%type,  
    p_username                IN     VARCHAR2 default null,  
    p_threshold_executions    IN     NATURAL  default 3,  
    p_threshold_sharable_mem  IN     NATURAL  default 1000,  
    p_parallel_degree         IN     NATURAL  default 1);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_db_link</td>
<td>Database link to the source name (mandatory). The database link pointing to the instance that will be used for extracting the SQL statements. The user must have the role SELECT_ON_CATALOG at the source instance. For improved security, the connection may use a password file or LDAP authentication. The database link is mandatory only for releases with dbms_libcache$def.ACCESS_METHOD = DB_LINK_METHOD.</td>
</tr>
<tr>
<td>p_instance_name</td>
<td>(Reserved for future use). The name of the instance that will be used for extracting the SQL statements. The instance name must be unique for all instances excluding the local instance. The name is not case sensitive.</td>
</tr>
<tr>
<td>p_username</td>
<td>Source username (default is all users). The name of the username that will be used for extracting the SQL statements. The username is an optional parameter that is used to ensure the parsing user id is the same as that on the source instance. For an application where users connect as a single user_id, for example APPS, APPS is the parsing user_id that is recorded in the shared pool. To select only SQL statements parsed by APPS, enter the string 'APPS' in this field. To also select statements executed by batch, repeat the executing the procedure with the schema owner, for example GL. If the username is supplied, it must be valid. The name is not case sensitive.</td>
</tr>
<tr>
<td>p_threshold_executions</td>
<td>The lower bound for the number of executions, below which a SQL statement will not be selected for parsing. This parameter is optional. It allows the application to extract and compile statements with executions, for example, greater than 3. The default value is 1. This means SQL statements that have never executed, including invalid SQL statements, will not be extracted.</td>
</tr>
</tbody>
</table>
Table 92-2  (Cont.) COMPARE_FROM_REMOTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_threshold_sharable_mem</td>
<td>The lower bound for the size of the shared memory consumed by the cursors on the source instance. Below this value a SQL statement will not be selected for parsing. This parameter is optional. It allows the application to extract and compile statements with shared memory for example, greater than 10000 bytes.</td>
</tr>
<tr>
<td>p_parallel_degree</td>
<td>The number of parallel jobs that execute to complete the parse operation. These tasks are spawned as parallel jobs against a sub-range of the SQL statements selected for parsing. This parameter is reserved for parallel compile jobs which are currently not implemented.</td>
</tr>
</tbody>
</table>
The **DBMS_LOB** package provides subprograms to operate on BLOBs, CLOBs, NCLOBs, BFILEs, and temporary LOBs. You can use **DBMS_LOB** to access and manipulate specific parts of a LOB or complete LOBs.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Constants**
- **Datatypes**
- **Operational Notes**
- **Rules and Limits**
- **Exceptions**
- **Summary of DBMS_LOB Subprograms**

---

**See Also:**

*Oracle Database SecureFiles and Large Objects Developer's Guide*

---

### 93.1 DBMS_LOB Overview

**DBMS_LOB** can read and modify BLOBs, CLOBs, and NCLOBs; it provides read-only operations for BFILEs. The bulk of the LOB operations are provided by this package.

### 93.2 DBMS_LOB Deprecated Subprograms

The **DBMS_LOB.LOADFROMFILE** program is deprecated from the **DBMS_LOB** package in Oracle Database 12c release 12.2

---

**Note:**

Oracle recommends that do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

Use **DBMS_LOB.LoadClobFromFile** or **DBMS_LOB.LoadBlobFromFile** instead.
93.3 DBMS_LOB Security Model

This package must be created under SYS. Operations provided by this package are performed under the current calling user, not under the package owner SYS.

Any DBMS_LOB subprogram called from an anonymous PL/SQL block is executed using the privileges of the current user. Any DBMS_LOB subprogram called from a stored procedure is executed using the privileges of the owner of the stored procedure.

When creating the procedure, users can set the AUTHID to indicate whether they want definer’s rights or invoker’s rights. For example:

CREATE PROCEDURE proc1 AUTHID DEFINER ...

or

CREATE PROCEDURE proc1 AUTHID CURRENT_USER ...

See Also:

For more information on AUTHID and privileges, see Oracle Database PL/SQL Language Reference

You can provide secure access to BFILEs using the DIRECTORY feature discussed in BFILENAME function in the Oracle Database SecureFiles and Large Objects Developer's Guide and the Oracle Database SQL Language Reference.

For information about the security model pertaining to temporary LOBs, see Operational Notes.

93.4 DBMS_LOB Constants

This topic describes the constants used by the DBMS_LOB package

These are shown in following tables:

- Table 93-1
- Table 93-2
- Table 93-3
- Table 93-4
- Table 93-5
- Table 93-6
### Table 93-1  DBMS_LOB Constants - Basic

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>PLS_INTEGER</td>
<td>12</td>
<td>Create the TEMP LOB with call duration</td>
</tr>
<tr>
<td>FILE_READONLY</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>Open the specified BFILE read-only</td>
</tr>
<tr>
<td>LOB_READONLY</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>Open the specified LOB read-only</td>
</tr>
<tr>
<td>LOB_READWRITE</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Open the specified LOB read-write</td>
</tr>
<tr>
<td>LOBMAXSIZE</td>
<td>INTEGER</td>
<td>18446744073709551615</td>
<td>Maximum size of a LOB in bytes</td>
</tr>
<tr>
<td>SESSION</td>
<td>PLS_INTEGER</td>
<td>10</td>
<td>Create the TEMP LOB with session duration</td>
</tr>
</tbody>
</table>

### Table 93-2  DBMS_LOB Constants - Option Types

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT_COMPRESS</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Set/Get the SECUREFILE compress option value</td>
</tr>
<tr>
<td>OPT_DEDUPLICATE</td>
<td>BINARY_INTEGER</td>
<td>4</td>
<td>Set/Get the SECUREFILE Deduplicate option value</td>
</tr>
<tr>
<td>OPT_ENCRYPT</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Get the SECUREFILE encrypt option value</td>
</tr>
</tbody>
</table>

### Table 93-3  DBMS_LOB Constants - Option Values

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESS_OFF</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>For SETOPTIONS Procedures, set compress off; for GETOPTIONS Functions, compress is off</td>
</tr>
<tr>
<td>COMPRESS_ON</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>For SETOPTIONS Procedures, set compress on; for GETOPTIONS Functions, compress is on</td>
</tr>
<tr>
<td>DEDUPLICATE_OFF</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>For SETOPTIONS Procedures, set deduplicate is off; for GETOPTIONS Functions, deduplicate is off</td>
</tr>
<tr>
<td>DEDUPLICATE_ON</td>
<td>BINARY_INTEGER</td>
<td>4</td>
<td>For SETOPTIONS Procedures, set deduplicate is on; for GETOPTIONS Functions, deduplicate is on</td>
</tr>
<tr>
<td>ENCRYPT_OFF</td>
<td>BINARY_INTEGER</td>
<td>0</td>
<td>For GETOPTIONS Functions, encrypt is off</td>
</tr>
<tr>
<td>ENCRYPT_ON</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>For GETOPTIONS Functions, encrypt is on</td>
</tr>
</tbody>
</table>
Table 93-4  DBMS_LOB Constants - DBFS State Value Types

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBFS_LINK_NEVER</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>LOB has never been archived</td>
</tr>
<tr>
<td>DBFS_LINK_NO</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>LOB was archived, but as been read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>back in to the RDBMS</td>
</tr>
<tr>
<td>DBFS_LINK_YES</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>LOB is currently archived</td>
</tr>
</tbody>
</table>

Table 93-5  DBMS_LOB Constants - DBFS Cache Flags

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBFS_LINK_CACHE</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Put the LOB data to the archive, but keep the data in the RDBMS as a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cached version</td>
</tr>
<tr>
<td>DBFS_LINK_NOCACHE</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>Put the LOB data to the archive, and remove the data from the RDBMS.</td>
</tr>
</tbody>
</table>

Table 93-6  DBMS_LOB Constants - Miscellaneous

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENTTYPE_MAX_SIZE</td>
<td>PLS_INTEGER</td>
<td>128</td>
<td>Maximum number of bytes allowed in the content type string</td>
</tr>
<tr>
<td>DBFS_LINK_PATH_MAX</td>
<td>PLS_INTEGER</td>
<td>1024</td>
<td>The maximum length of DBFS pathnames</td>
</tr>
</tbody>
</table>

93.5 DBMS_LOB Datatypes

The table in this topic describes the datatypes used by DBMS_LOB.

Table 93-7  Datatypes Used by DBMS_LOB

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>Source or destination binary LOB.</td>
</tr>
<tr>
<td>RAW</td>
<td>Source or destination RAW buffer (used with BLOB).</td>
</tr>
<tr>
<td>CLOB</td>
<td>Source or destination character LOB (including NCLOB).</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>Source or destination character buffer (used with CLOB and NCLOB).</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Specifies the size of a buffer or LOB, the offset into a LOB, or the amount to access.</td>
</tr>
<tr>
<td>BFILE</td>
<td>Large, binary object stored outside the database.</td>
</tr>
</tbody>
</table>

The DBMS_LOB package defines no special types.
An **NCLOB** is a **CLOB** for holding fixed-width and varying-width, multibyte national character sets.

The clause **ANY_CS** in the specification of **DBMS_LOB** subprograms for **CLOBs** enables the **CLOB** type to accept a **CLOB** or **NCLOB** locator variable as input.

### 93.6 DBMS_LOB Operational Notes

All **DBMS_LOB** subprograms work based on LOB locators. For the successful completion of **DBMS_LOB** subprograms, you must provide an input locator that represents a LOB that already exists in the database tablespaces or external file system.

See also Chapter 1 of *Oracle Database SecureFiles and Large Objects Developer's Guide*

Starting from 12.2 release, you can select a persistent LOB locator from a remote table into a local variable. The remote column can be of type BLOB, CLOB, or NCLOB. You cannot select BFILE from a remote table. The LOB variable that refers to the LOB value in a remote table is called a remote locator.

All the **DBMS_LOB** APIs other than the ones that are meant for BFILEs will now accept and support operations on remote LOB locators. All the APIs that take in two locators must have both LOBs collocated at one database.

---

**See Also:**

*Distributed LOBs* chapter in *Oracle Database SecureFiles and Large Objects Developer's Guide*.

To use LOBs in your database, you must first use SQL data definition language (DDL) to define the tables that contain LOB columns.

- Internal LOBs
- External LOBs
- Temporary LOBs

**Internal LOBs**

To populate your table with internal LOBs after LOB columns are defined in a table, you use the SQL data manipulation language (DML) to initialize or populate the locators in the LOB columns.

**External LOBs**

For an external LOB (BFILE) to be represented by a LOB locator, you must:

- Ensure that a **DIRECTORY** object representing a valid, existing physical directory has been defined, and that physical files (the LOBs you plan to add) exist with read permission for the database. If your operating system uses case-sensitive path names, then be sure you specify the directory in the correct format.

- Pass the **DIRECTORY** object and the filename of the external LOB you are adding to the **BFILENAME** function to create a LOB locator for your external LOB.
Once you have completed these tasks, you can insert or update a row containing a 
LOB column using the specified LOB locator.

After the LOBs are defined and created, you can then select from a LOB locator into 
a local PL/SQL LOB variable and use this variable as an input parameter to DBMS_LOB 
for access to the LOB value.

For details on the different ways to do this, see Oracle Database SecureFiles and 
Large Objects Developer's Guide

Temporary LOBs

The database supports the definition, creation, deletion, access, and update of 
temporary LOBs. Your temporary tablespace stores the temporary LOB data. 
Temporary LOBs are not permanently stored in the database. Their purpose is mainly 
to perform transformations on LOB data.

For temporary LOBs, you must use the OCI, PL/SQL, or another programmatic 
interface to create or manipulate them. Temporary LOBs can be either BLOBs, CLOBs, or 
NCLOBs.

A temporary LOB is empty when it is created. By default, all temporary LOBs are 
deleted at the end of the session in which they were created. If a process dies 
unexpectedly or if the database crashes, then temporary LOBs are deleted, and the 
space for temporary LOBs is freed.

There is also an interface to let you group temporary LOBs together into a logical 
bucket. The duration represents this logical store for temporary LOBs. Each temporary 
LOB can have separate storage characteristics, such as CACHE/NOCACHE. There is a 
default store for every session into which temporary LOBs are placed if you don’t 
specify a specific duration. Additionally, you are able to perform a free operation on 
durations, which causes all contents in a duration to be freed.

There is no support for consistent read (CR), undo, backup, parallel processing, or 
transaction management for temporary LOBs. Because CR and roll backs are not 
supported for temporary LOBs, you must free the temporary LOB and start over again 
if you encounter an error.

Because CR, undo, and versions are not generated for temporary LOBs, there is 
potentially a performance impact if you assign multiple locators to the same temporary 
LOB. Semantically, each locator should have its own copy of the temporary LOB.

A copy of a temporary LOB is created if the user modifies the temporary LOB while 
another locator is also pointing to it. The locator on which a modification was 
performed now points to a new copy of the temporary LOB. Other locators no longer 
see the same data as the locator through which the modification was made. A deep 
copy was not incurred by permanent LOBs in these types of situations, because CR 
snapshots and version pages enable users to see their own versions of the LOB 
cheaply.

You can gain pseudo-REF semantics by using pointers to locators in OCI and by having 
multiple pointers to locators point to the same temporary LOB locator, if necessary. In 
PL/SQL, you must avoid using more than one locator for each temporary LOB. The 
temporary LOB locator can be passed by reference to other procedures.

Because temporary LOBs are not associated with any table schema, there are no 
meanings to the terms in-row and out-of-row temporary LOBs. Creation of a temporary 
LOB instance by a user causes the engine to create and return a locator to the LOB
data. The PL/SQL DBMS_LOB package, PRO*C/C++, OCI, and other programmatic interfaces operate on temporary LOBs through these locators just as they do for permanent LOBs.

There is no support for client side temporary LOBs. All temporary LOBs reside in the server. Temporary LOBs do not support the EMPTY_BLOB or EMPTY_CLOB functions that are supported for permanent LOBs. The EMPTY_BLOB function specifies the fact that the LOB is initialized, but not populated with any data.

A temporary LOB instance can only be destroyed by using OCI or the DBMS_LOB package by using the appropriate FREETEMPORARY or OCIDurationEnd statement.

A temporary LOB instance can be accessed and modified using appropriate OCI and DBMS_LOB statements, just as for regular permanent internal LOBs. To make a temporary LOB permanent, you must explicitly use the OCI or DBMS_LOB COPY command, and copy the temporary LOB into a permanent one.

Security is provided through the LOB locator. Only the user who created the temporary LOB is able to see it. Locators are not expected to be able to pass from one user's session to another. Even if someone did pass a locator from one session to another, they would not access the temporary LOBs from the original session. Temporary LOB lookup is localized to each user's own session. Someone using a locator from somewhere else is only able to access LOBs within his own session that have the same LOB ID. Users should not try to do this, but if they do, they are not able to affect anyone else's data.

The database keeps track of temporary LOBs for each session in a $ view called V$TEMPORARY_LOBS, which contains information about how many temporary LOBs exist for each session. V$ views are for DBA use. From the session, the database can determine which user owns the temporary LOBs. By using V$TEMPORARY_LOBS in conjunction with DBA_SEGMENTS, a DBA can see how much space is being used by a session for temporary LOBs. These tables can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

The following notes are specific to temporary LOBs:

1. All functions in DBMS_LOB return NULL if any of the input parameters are NULL. All procedures in DBMS_LOB raise an exception if the LOB locator is input as NULL.

2. Operations based on CLOBs do not verify if the character set IDs of the parameters (CLOB parameters, VARCHAR2 buffers and patterns, and so on) match. It is the user's responsibility to ensure this.

3. Data storage resources are controlled by the DBA by creating different temporary tablespaces. DBAs can define separate temporary tablespaces for different users, if necessary.

See Also:

Oracle Database PL/SQL Language Reference for more information on NOCOPY syntax
93.7 DBMS_LOB Rules and Limits

This topic describes general DBMS_LOB rules and limits, rules and limits specific to external files (BFILEs), and maximum LOB and buffer sizes.

General Rules and Limits

- Oracle Database does not support constraints on columns or attributes whose type is a LOB, with the following exception: NOT NULL constraints are supported for a LOB column or attribute.
- The following rules apply in the specification of subprograms in this package:
  - `newlen`, `offset`, and `amount` parameters for subprograms operating on BLOBs and BFILEs must be specified in terms of bytes.
  - `newlen`, `offset`, and `amount` parameters for subprograms operating on CLOBs must be specified in terms of characters.

In multi-byte character sets, it is not possible to interpret these offsets correctly. As a result, `SUBSTR` raises the following error: ORA-22998: CLOB or NCLOB in multibyte character set not supported.
- A subprogram raises an `INVALID_ARGVAL` exception if the following restrictions are not followed in specifying values for parameters (unless otherwise specified):
  1. Only positive, absolute offsets from the beginning of LOB data are permitted: Negative offsets from the tail of the LOB are not permitted.
  2. Only positive, nonzero values are permitted for the parameters that represent size and positional quantities, such as `amount`, `offset`, `newlen`, `nth`, and so on. Negative offsets and ranges observed in SQL string functions and operators are not permitted.
  3. The value of `offset`, `amount`, `newlen`, `nth` must not exceed the value `lobmaxsize` 18446744073709551615 (2^64) in any DBMS_LOB subprogram.
  4. For CLOBs consisting of fixed-width multibyte characters, the maximum value for these parameters must not exceed \( (\text{lobmaxsize} / \text{character_width_in_bytes}) \) characters.

For example, if the CLOB consists of 2-byte characters, such as:

`JA16SJISFIXED`

Then, the maximum `amount` value should not exceed:

\[ 18446744073709551615 / 2 = 9223372036854775807 \]

- PL/SQL language specifications stipulate an upper limit of 32767 bytes (not characters) for RAW and VARCHAR2 parameters used in DBMS_LOB subprograms. For example, if you declare a variable to be:

```
charbuf VARCHAR2(3000)
```

Then, `charbuf` can hold 3000 single byte characters or 1500 2-byte fixed width characters. This has an important consequence for DBMS_LOB subprograms for CLOBs and NCLOBs.
• The %CHARSET clause indicates that the form of the parameter with %CHARSET must match the form of the ANY_CS parameter to which it refers.

For example, in DBMS_LOB subprograms that take a VARCHAR2 buffer parameter, the form of the VARCHAR2 buffer must match the form of the CLOB parameter. If the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

For DBMS_LOB subprograms that take two CLOB parameters, both CLOB parameters must have the same form; that is, they must both be NCLOBs, or they must both be CLOBs.

• If the value of amount plus the offset exceeds the maximum LOB size allowed by the database, then access exceptions are raised.

Under these input conditions, read subprograms, such as READ, COMPARE, INSTR, and SUBSTR, read until End of Lob/File is reached. For example, for a READ operation on a BLOB or BFILE, if the user specifies offset value of 3 GB and an amount value of 2 GB on a LOB that is 4GB in size, then READ returns only 1GB (4GB-3GB) bytes.

• Functions with NULL or invalid input values for parameters return a NULL. Procedures with NULL values for destination LOB parameters raise exceptions.

• Operations involving patterns as parameters, such as COMPARE, INSTR, and SUBSTR do not support regular expressions or special matching characters (such as % in the LIKE operator in SQL) in the pattern parameter or substrings.

• The End Of LOB condition is indicated by the READ procedure using a NO_DATA_FOUND exception. This exception is raised only upon an attempt by the user to read beyond the end of the LOB. The READ buffer for the last read contains 0 bytes.

• For consistent LOB updates, you must lock the row containing the destination LOB before making a call to any of the procedures (mutators) that modify LOB data.

• Unless otherwise stated, the default value for an offset parameter is 1, which indicates the first byte in the BLOB or BFILE data, and the first character in the CLOB or NCLOB value. No default values are specified for the amount parameter — you must input the values explicitly.

• You must lock the row containing the destination internal LOB before calling any subprograms that modify the LOB, such as APPEND, COPY, ERASE, TRIM, or WRITE. These subprograms do not implicitly lock the row containing the LOB.

Rules and Limits Specific to External Files (BFILES)

• The subprograms COMPARE, INSTR, READ, SUBSTR, FILECLOSE, FILECLOSEALL and LOADFROMFILE operate only on an opened BFILE locator; that is, a successful FILEOPEN call must precede a call to any of these subprograms.

• For the functions FILEEXISTS, FILEGETNAME and GETLENGTH, a file's open/close status is unimportant; however, the file must exist physically, and you must have adequate privileges on the DIRECTORY object and the file.

• DBMS_LOB does not support any concurrency control mechanism for BFILE operations.

• In the event of several open files in the session whose closure has not been handled properly, you can use the FILECLOSEALL subprogram to close all files opened in the session and resume file operations from the beginning.

• If you are the creator of a DIRECTORY, or if you have system privileges, then use the CREATE OR REPLACE, DROP, and REVOKE statements in SQL with extreme caution.
If you, or other grantees of a particular directory object, have several open files in a session, then any of the preceding commands can adversely affect file operations. In the event of such abnormal termination, your only choice is to invoke a program or anonymous block that calls FILECLOSEALL, reopen your files, and restart your file operations.

- All files opened during a user session are implicitly closed at the end of the session. However, Oracle strongly recommends that you close the files after both normal and abnormal termination of operations on the BFILE.

In the event of normal program termination, proper file closure ensures that the number of files that are open simultaneously in the session remains less than SESSION_MAX_OPEN_FILES.

In the event of abnormal program termination from a PL/SQL program, it is imperative that you provide an exception handler that ensures closure of all files opened in that PL/SQL program. This is necessary because after an exception occurs, only the exception handler has access to the BFILE variable in its most current state.

After the exception transfers program control outside the PL/SQL program block, all references to the open BFILEs are lost. The result is a larger open file count which may or may not exceed the SESSION_MAX_OPEN_FILES value.

For example, consider a READ operation past the end of the BFILE value, which generates a NO_DATA_FOUND exception:

```sql
-- This assumes a directory 'DDD' whose path is already known
DECLARE
  fil BFILE:= bfilename('DDD', 'filename.foo');
  pos INTEGER;
  amt BINARY_INTEGER;
  buf RAW(40);
BEGIN
  SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
  dbms_lob.open(fil, dbms_lob.lob_readonly);
  amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
  dbms_lob.read(fil, amt, pos, buf);
  dbms_output.put_line('Read F1 past EOF: '||
    utl_raw.cast_to_varchar2(buf));
  dbms_lob.close(fil);
END;
ORA-01403: no data found
ORA-06512: at "SYS.DBMS_LOB", line 373
ORA-06512: at line 10

After the exception has occurred, the BFILE locator variable file goes out of scope, and no further operations on the file can be done using that variable. Therefore, the solution is to use an exception handler:

```sql
DECLARE
  fil BFILE;
  pos INTEGER;
  amt BINARY_INTEGER;
  buf RAW(40);
BEGIN
  SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
  dbms_lob.open(fil, dbms_lob.lob_readonly);
  amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
  dbms_lob.read(fil, amt, pos, buf);
```
In general, you should ensure that files opened in a PL/SQL block using `DBMS_LOB` are closed before normal or abnormal termination of the block.

**Maximum LOB Size**

The maximum size for LOBs supported by the database is equal to the value of the blocksize of the tablespace the LOB column resides in times the value $2^{32}-1$ (4294967295). This allows for a maximum LOB size ranging from 8 terabytes to 128 terabytes.

**Maximum Buffer Size**

The maximum buffer size, 32767 bytes.

For BLOBs, where buffer size is expressed in bytes, the number of bytes cannot exceed 32767.

For CLOBs or NCLOBs, where buffer size is expressed in characters, the number of characters cannot result in a buffer larger than 32767 bytes. For example, if you are using fixed-width, two-byte characters, then specifying 20000 characters is an error ($20000 \times 2 = 40000$, which is greater than 32767).

### 93.8 DBMS_LOB Exceptions

The table in the topic describes the exceptions for DBMS_LOB.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_ERROR</td>
<td>22925</td>
<td>You are trying to write too much data to the LOB: LOB size is limited to 4 gigabytes.</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>22279</td>
<td>Cannot perform operation with LOB buffering enabled</td>
</tr>
<tr>
<td>CONTENTTYPE_TOOLONG</td>
<td>43859</td>
<td>The length of the contenttype string exceeds the defined maximum. Modify the length of the contenttype string and retry the operation.</td>
</tr>
<tr>
<td>CONTENTTYPEBUF_WRONG</td>
<td>43862</td>
<td>The length of the contenttype buffer is less than defined constant. Modify the length of the contenttype buffer and retry the operation.</td>
</tr>
</tbody>
</table>
Table 93-8  (Cont.) DBMS_LOB Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>21560</td>
<td>The argument is expecting a non-NULL, valid value but the argument value passed in is NULL, invalid, or out of range.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>22287</td>
<td>The directory used for the current operation is not valid if being accessed for the first time, or if it has been modified by the DBA since the last access.</td>
</tr>
<tr>
<td>NO_DATA_FOUND</td>
<td>1403</td>
<td>ENDOFLOB indicator for looping read operations. This is not a hard error.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>22285</td>
<td>The directory leading to the file does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>22286</td>
<td>The user does not have the necessary access privileges on the directory or the file for the operation.</td>
</tr>
<tr>
<td>OPEN_TOOMANY</td>
<td>22290</td>
<td>The number of open files has reached the maximum limit.</td>
</tr>
<tr>
<td>OPERATION_FAILED</td>
<td>22288</td>
<td>The operation attempted on the file failed.</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>14553</td>
<td>Cannot perform a LOB write inside a query or PDML parallel execution server</td>
</tr>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>43856</td>
<td>A non-SECUREFILE LOB type was used in a SECUREFILE only call</td>
</tr>
<tr>
<td>SECUREFILE_BADPARAM</td>
<td>43857</td>
<td>An invalid argument was passed to a SECUREFILE subprogram</td>
</tr>
<tr>
<td>SECUREFILE_MARKERASED</td>
<td>43861</td>
<td>The mark provided to a FRAGMENT_* operation has been deleted</td>
</tr>
<tr>
<td>SECUREFILE_OUTOFBOUNDS</td>
<td>43883</td>
<td>Attempted to perform a FRAGMENT_* operation past the LOB end</td>
</tr>
<tr>
<td>UNOPENED_FILE</td>
<td>22289</td>
<td>The file is not open for the required operation to be performed.</td>
</tr>
<tr>
<td>VALUE_ERROR</td>
<td>6502</td>
<td>PL/SQL error for invalid values to subprogram's parameters.</td>
</tr>
</tbody>
</table>

93.9 Summary of DBMS_LOB Subprograms

This table lists the DBMS_LOB subprograms and briefly describes them.

Table 93-9  DBMS_LOB Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND Procedures</td>
<td>Appends the contents of the source LOB to the destination LOB</td>
</tr>
<tr>
<td>CLOB2FILE Procedure</td>
<td>Writes the content of a CLOB into a file.</td>
</tr>
<tr>
<td>CLOSE Procedure</td>
<td>Closes a previously opened internal or external LOB</td>
</tr>
<tr>
<td>COMPARE Functions</td>
<td>Compares two entire LOBs or parts of two LOBs</td>
</tr>
<tr>
<td>CONVERTTOBLOB Procedure</td>
<td>Reads character data from a source CLOB or NCLOB instance, converts the character data to the specified character, writes the converted data to a destination BLOB instance in binary format, and returns the new offsets</td>
</tr>
</tbody>
</table>
## Table 93-9  (Cont.) DBMS_LOB Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONVERTTOCLOB Procedure</strong></td>
<td>Takes a source BLOB instance, converts the binary data in the source instance to character data using the specified character, writes the character data to a destination CLOB or NCLOB instance, and returns the new offsets</td>
</tr>
<tr>
<td><strong>COPY Procedures</strong></td>
<td>Copies all, or part, of the source LOB to the destination LOB</td>
</tr>
<tr>
<td><strong>COPY_DBFS_LINK Procedures</strong></td>
<td>Copies the DBFS Link in the source LOB to the destination LOB</td>
</tr>
<tr>
<td><strong>COPY_FROM_DBFS_LINK Procedures</strong></td>
<td>Retrieves the data for the LOB from the DBFS store</td>
</tr>
<tr>
<td><strong>CREATETEMPORARY Procedures</strong></td>
<td>Creates a temporary BLOB or CLOB and its corresponding index in the user's default temporary tablespace</td>
</tr>
<tr>
<td><strong>DBFS_LINK_GENERATE_PATH H Functions</strong></td>
<td>Returns a unique file path name for use in creating a DBFS Link</td>
</tr>
<tr>
<td><strong>ERASE Procedures</strong></td>
<td>Erases all or part of a LOB</td>
</tr>
<tr>
<td><strong>FILECLOSE Procedure</strong></td>
<td>Closes the file</td>
</tr>
<tr>
<td><strong>FILECLOSEALL Procedure</strong></td>
<td>Closes all previously opened files</td>
</tr>
<tr>
<td><strong>FILEEXISTS Function</strong></td>
<td>Checks if the file exists on the server</td>
</tr>
<tr>
<td><strong>FILEGETNAME Procedure</strong></td>
<td>Gets the directory object name and file name</td>
</tr>
<tr>
<td><strong>FILEISOPEN Function</strong></td>
<td>Checks if the file was opened using the input BFILE locators</td>
</tr>
<tr>
<td><strong>FILEOPEN Procedure</strong></td>
<td>Opens a file</td>
</tr>
<tr>
<td><strong>FRAGMENT_DELETE Procedure</strong></td>
<td>Deletes the data at the specified offset for the specified length from the LOB</td>
</tr>
<tr>
<td><strong>FRAGMENT_INSERT Procedures</strong></td>
<td>Inserts the specified data (limited to 32K) into the LOB at the specified offset</td>
</tr>
<tr>
<td><strong>FRAGMENT_MOVE Procedure</strong></td>
<td>Moves the amount of bytes (BLOB) or characters (CLOB/NCLOB) from the specified offset to the new offset specified</td>
</tr>
<tr>
<td><strong>FRAGMENT_REPLACE Procedures</strong></td>
<td>Replaces the data at the specified offset with the specified data (not to exceed 32k)</td>
</tr>
<tr>
<td><strong>FREETEMPORARY Procedures</strong></td>
<td>Frees the temporary BLOB or CLOB in the default temporary tablespace</td>
</tr>
<tr>
<td><strong>GET_DBFS_LINK Functions</strong></td>
<td>Returns the DBFS Link path associated with the specified SecureFile</td>
</tr>
<tr>
<td><strong>GET_DBFS_LINK_STATE Procedures</strong></td>
<td>Retrieves the current DBFS Link state of the specified SecureFile</td>
</tr>
<tr>
<td><strong>GETCHUNKSIZE Functions</strong></td>
<td>Returns the amount of space used in the LOB chunk to store the LOB value</td>
</tr>
<tr>
<td><strong>GETCONTENTTYPE Functions</strong></td>
<td>Returns the content ID string previously set by means of the SETCONTENTTYPE Procedure</td>
</tr>
<tr>
<td><strong>GETLENGTH Functions</strong></td>
<td>Gets the length of the LOB value</td>
</tr>
<tr>
<td><strong>GETOPTIONS Functions</strong></td>
<td>Obtains settings corresponding to the option_type field for a particular LOB</td>
</tr>
<tr>
<td><strong>GET_STORAGE_LIMIT Function</strong></td>
<td>Returns the storage limit for LOBs in your database configuration</td>
</tr>
</tbody>
</table>
### Table 93-9  (Cont.) DBMS_LOB Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INSTR Functions</strong></td>
<td>Returns the matching position of the ( n \text{th} ) occurrence of the pattern in the LOB</td>
</tr>
<tr>
<td><strong>ISOPEN Functions</strong></td>
<td>Checks to see if the LOB was already opened using the input locator</td>
</tr>
<tr>
<td><strong>ISREMOTE Function</strong></td>
<td>Checks to see if the LOB is local to the database or if it belongs to a remote database.</td>
</tr>
<tr>
<td><strong>ISSECUREFILE Function</strong></td>
<td>Returns <code>TRUE</code> if the LOB locator passed to is for a SecureFiles LOB, otherwise, returns <code>FALSE</code></td>
</tr>
<tr>
<td><strong>ISTEMPORARY Functions</strong></td>
<td>Checks if the locator is pointing to a temporary LOB</td>
</tr>
<tr>
<td><strong>LOADBLOBFROMFILE Procedure</strong></td>
<td>Loads BFILE data into an internal BLOB</td>
</tr>
<tr>
<td><strong>LOADCLOBFROMFILE Procedure</strong></td>
<td>Loads BFILE data into an internal CLOB</td>
</tr>
<tr>
<td><strong>LOADFROMFILE Procedure</strong></td>
<td>Loads BFILE data into an internal LOB. This procedure has been deprecated starting with Oracle Database 12c release 12.2.</td>
</tr>
<tr>
<td><strong>MOVE_TO_DBFS_LINK Procedures</strong></td>
<td>Writes the specified SecureFile data to the DBFS store</td>
</tr>
<tr>
<td><strong>OPEN Procedures</strong></td>
<td>Opens a LOB (internal, external, or temporary) in the indicated mode</td>
</tr>
<tr>
<td><strong>READ Procedures</strong></td>
<td>Reads data from the LOB starting at the specified offset</td>
</tr>
<tr>
<td><strong>SET_DBFS_LINK Procedures</strong></td>
<td>Links the specified SecureFile to the specified path name. It does not copy the data to the path</td>
</tr>
<tr>
<td><strong>SETCONTENTTYPE Procedure</strong></td>
<td>Sets the content type string for the data in the LOB</td>
</tr>
<tr>
<td><strong>SETOPTIONS Procedures</strong></td>
<td>Enables CSCE features on a per-LOB basis, overriding the default LOB column settings</td>
</tr>
<tr>
<td><strong>SUBSTR Functions</strong></td>
<td>Returns part of the LOB value starting at the specified offset</td>
</tr>
<tr>
<td><strong>TRIM Procedures</strong></td>
<td>Trims the LOB value to the specified shorter length</td>
</tr>
<tr>
<td><strong>WRITE Procedures</strong></td>
<td>Writes data to the LOB from a specified offset</td>
</tr>
<tr>
<td><strong>WRITEAPPEND Procedures</strong></td>
<td>Writes a buffer to the end of a LOB</td>
</tr>
</tbody>
</table>

#### 93.9.1 APPEND Procedures

This procedure appends the contents of a source internal LOB to a destination LOB. It appends the complete source LOB.

**Syntax**

```sql
DBMS_LOB.APPEND (    dest_lob IN OUT NOCOPY BLOB,    src_lob IN BLOB);
```

```sql
DBMS_LOB.APPEND (    dest_lob IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,    src_lob IN CLOB CHARACTER SET dest_lob%CHARSET);
```
Parameters

Table 93-10  APPEND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>Locator for the internal LOB to which the data is to be appended.</td>
</tr>
<tr>
<td>src_lob</td>
<td>Locator for the internal LOB from which the data is to be read.</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-11  APPEND Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either the source or the destination LOB is NULL.</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform a LOB write inside a query or PDML parallel execution server</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on either LOB</td>
</tr>
</tbody>
</table>

Usage Notes

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

- If APPEND is called on a LOB that has been archived, it implicitly gets the LOB before the first byte is written.
- If APPEND is called on a SecureFiles LOB that is a DBFS Link, an exception is thrown.

For additional details on usage of this procedure, see

Oracle Database SecureFiles and Large Objects Developer’s Guide
93.9.2 CLOB2FILE Procedure

This procedure writes the content of a CLOB into a bfile. This procedure gets called from the deprecated dbms_xslprocessor.clob2file internally.

Syntax

```
DBMS_LOB.CLOB2FILE(
    src_cl      IN  CLOB,
    file_loc    IN  VARCHAR2,
    file_name   IN  VARCHAR2,
    csid        IN  NUMBER   := 0,
    open_mode   IN  VARCHAR2 :='wb');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_cl</td>
<td>Source CLOB locator to write into a file</td>
</tr>
<tr>
<td>file_loc</td>
<td>Directory object name where the file is located</td>
</tr>
<tr>
<td>file_name</td>
<td>File name</td>
</tr>
<tr>
<td>csid</td>
<td>Character set id of the CLOB locator</td>
</tr>
<tr>
<td></td>
<td>• Must be a valid Oracle id; otherwise returns an error</td>
</tr>
<tr>
<td></td>
<td>• If the value is 0, then the content of the output file will be in the database character set</td>
</tr>
<tr>
<td>open_mode</td>
<td>The mode to open the output file in</td>
</tr>
<tr>
<td></td>
<td>• wb — write byte mode, overwrites the file</td>
</tr>
<tr>
<td></td>
<td>The default value is wb.</td>
</tr>
</tbody>
</table>

93.9.3 CLOSE Procedure

This procedure closes a previously opened internal or external LOB.

Syntax

```
DBMS_LOB.CLOSE (
    lob_loc     IN OUT NOCOPY BLOB);
```

```
DBMS_LOB.CLOSE (
    lob_loc     IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);
```

```
DBMS_LOB.CLOSE (
    file_loc    IN OUT NOCOPY BFILE);
```
Parameters

Table 93-13  CLOSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
</tbody>
</table>

Exceptions

No error is returned if the BFILE exists but is not opened. An error is returned if the LOB is not open.

Usage Notes

CLOSE requires a round-trip to the server for both internal and external LOBs. For internal LOBs, CLOSE triggers other code that relies on the close call, and for external LOBs (BFILEs), CLOSE actually closes the server-side operating system file.

It is not mandatory that you wrap all LOB operations inside the Open/Close interfaces. However, if you open a LOB, you must close it before you commit the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

See Also:

Oracle Database SecureFiles and Large Objects Developer’s Guide for additional details on usage of this procedure

93.9.4 COMPARE Functions

This function compares two entire LOBs or parts of two LOBs.

Syntax

DBMS_LOB.COMPARE ( lob_1 IN BLOB, lob_2 IN BLOB, amount IN INTEGER := DBMS_LOB.LOBMAXSIZE, offset_1 IN INTEGER := 1, offset_2 IN INTEGER := 1) RETURN INTEGER;

DBMS_LOB.COMPARE ( lob_1 IN CLOB CHARACTER SET ANY_CS, lob_2 IN CLOB CHARACTER SET lob_1%CHARSET,
amount           IN INTEGER := DBMS_LOB.LOBMAXSIZE,
offset_1         IN INTEGER := 1,
offset_2         IN INTEGER := 1)
RETURN INTEGER;

DBMS_LOB.COMPARE (  
  lob_1            IN BFILE,
  lob_2            IN BFILE,
  amount           IN INTEGER,
  offset_1         IN INTEGER := 1,
  offset_2         IN INTEGER := 1)
RETURN INTEGER;

Pragmas
pragma restrict_references(COMPARE, WNDS, WNPS, RNDS, RNPS);

Parameters

Table 93-14  COMPARE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_1</td>
<td>LOB locator of first target for comparison.</td>
</tr>
<tr>
<td>lob_2</td>
<td>LOB locator of second target for comparison.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBS) or characters (for CLOBS/NCLOBSs) to compare.</td>
</tr>
<tr>
<td>offset_1</td>
<td>Offset in bytes or characters on the first LOB (origin: 1) for the comparison.</td>
</tr>
<tr>
<td>offset_2</td>
<td>Offset in bytes or characters on the second LOB (origin: 1) for the comparison.</td>
</tr>
</tbody>
</table>

Return Values

- INTEGER: 0 if the comparison succeeds, nonzero if not.
- NULL, if any of amount, offset_1 or offset_2 is not a valid LOB offset value. A valid offset is within the range of 1 to LOBMAXSIZE inclusive.

Usage Notes

- You can only compare LOBs of the same datatype (LOBs of BLOB type with other BLOBs, and CLOBs with CLOBs, and BFILEs with BFILEs). For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.
- COMPARE returns 0 if the data exactly matches over the range specified by the offset and amount parameters. COMPARE returns -1 if the first CLOB is less than the second, and 1 if it is greater.
- For fixed-width n-byte CLOBs, if the input amount for COMPARE is specified to be greater than (DBMS_LOB.LOBMAXSIZE/n), then COMPARE matches characters in a range of size (DBMS_LOB.LOBMAXSIZE/n), or Max(length(clob1), length(clob2)), whichever is lesser.
- If COMPARE is called on any LOB that has been archived, it implicitly gets the LOB before the compare begins.
• If COMPARE() is called on a SecureFiles LOB that is a DBFS Link, the linked LOB is stream from DBFS, if possible, otherwise an exception is thrown.

Exceptions

Table 93-15 COMPARE Function Exceptions for BFILE operations

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>File was not opened using the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on either LOB</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database SecureFiles and Large Objects Developer’s Guide for additional details on usage of this procedure

93.9.5 CONVERTTOBLOB Procedure

This procedure reads character data from a source CLOB or NCLOB instance, converts the character data to the character set you specify, writes the converted data to a destination BLOB instance in binary format, and returns the new offsets.

You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

Syntax

```sql
DBMS_LOB.CONVERTTOBLOB(
    dest_lob       IN OUT NOCOPY  BLOB,
    src_clob       IN         CLOB CHARACTER SET ANY_CS,
    amount         IN         INTEGER,
    dest_offset    IN OUT     INTEGER,
    src_offset     IN OUT     INTEGER,
    blob_csid      IN         NUMBER,
    lang_context   IN OUT     INTEGER,
    warning        OUT        INTEGER);
```

Parameters

Table 93-16 CONVERTTOBLOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>LOB locator of the destination LOB instance.</td>
</tr>
</tbody>
</table>
Table 93-16  (Cont.) CONVERTTOBLOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_clob</td>
<td>LOB locator of the source LOB instance.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of characters to convert from the source LOB. If you want to copy the entire LOB, pass the constant DBMS_LOB.LOBMAXSIZE. If you pass any other value, it must be less than or equal to the size of the LOB.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>(IN) Offset in bytes in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB. (OUT) The new offset in bytes after the end of the write.</td>
</tr>
<tr>
<td>src_offset</td>
<td>(IN) Offset in characters in the source LOB for the start of the read. (OUT) Offset in characters in the source LOB right after the end of the read.</td>
</tr>
<tr>
<td>blob_csid</td>
<td>Desired character set ID of the converted data.</td>
</tr>
<tr>
<td>lang_context</td>
<td>(IN) Language context, such as shift status, for the current conversion. (OUT) The language context at the time when the current conversion is done. This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.</td>
</tr>
<tr>
<td>warning</td>
<td>(OUT) Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message. Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, “?”) is used in place of the inconvertible character. The return value of this error message is defined as the constant warn_inconvertible_char in the DBMS_LOB package.</td>
</tr>
</tbody>
</table>

Usage Notes

Preconditions

Before calling the CONVERTTOBLOB procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked. To lock the row, select the LOB using the FOR UPDATE clause of the SELECT statement.

Constants and Defaults

All parameters are required. You must pass a variable for each OUT or IN OUT parameter. You must pass either a variable or a value for each IN parameter.

Table 93-17 gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the dbmslob.sql package specification file.
Table 93-17  DBMS_LOB.CONVERTTOBLOB Typical Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>LOBMAXSIZE (IN)</td>
<td>convert the entire file</td>
</tr>
<tr>
<td>dest_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
<tr>
<td>src_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
<tr>
<td>blob_csid</td>
<td>DEFAULT_CSID (IN)</td>
<td>default CSID, use same CSID as source LOB</td>
</tr>
<tr>
<td>lang_context</td>
<td>DEFAULT_LANG_CTX (IN)</td>
<td>default language context</td>
</tr>
<tr>
<td>warning</td>
<td>NO_WARNING (OUT)</td>
<td>no warning message, success</td>
</tr>
<tr>
<td></td>
<td>WARN_INCONVERTIBLE_CHAR</td>
<td>character in source cannot be properly converted</td>
</tr>
</tbody>
</table>

General Notes

- You must specify the desired character set for the destination LOB in the `blob_csid` parameter. You can pass a zero value for `blob_csid`. When you do so, the database assumes that the desired character set is the same as the source LOB character set.

- You must specify the offsets for both the source and destination LOBs, and the number of characters to copy from the source LOB. The `amount` and `src_offset` values are in characters and the `dest_offset` is in bytes. To convert the entire LOB, you can specify LOBMAXSIZE for the `amount` parameter.

- `CONVERTTOBLOB` gets the source and/or destination LOBs as necessary prior to conversion and write of the data.

Exceptions

Table 93-18 gives possible exceptions this procedure can throw. The first column lists the exception string and the second column describes the error conditions that can cause the exception.

Table 93-18  CONVERTTOBLOB Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>One or more of the following:</td>
</tr>
<tr>
<td></td>
<td>- <code>src_offset</code> or <code>dest_offset</code> &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- <code>src_offset</code> or <code>dest_offset</code> &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td></td>
<td>- <code>amount</code> &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- <code>amount</code> &gt; LOBMAXSIZE.</td>
</tr>
</tbody>
</table>

See Also:  
*Oracle Database SecureFiles and Large Objects Developer's Guide* for more information on using LOBs in application development
93.9.6 CONVERTTOCLOB Procedure

This procedure takes a source BLOB instance, converts the binary data in the source instance to character data using the character set you specify, writes the character data to a destination CLOB or NCLOB instance, and returns the new offsets.

You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

**Syntax**

```sql
DBMS_LOB.CONVERTTOCLOB(
    dest_lob       IN OUT NOCOPY  CLOB CHARACTER SET ANY_CS,
    src_blob       IN             BLOB,
    amount         IN             INTEGER,
    dest_offset    IN OUT         INTEGER,
    src_offset     IN OUT         INTEGER,
    blob_csid      IN             NUMBER,
    lang_context   IN OUT         INTEGER,
    warning        OUT            INTEGER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>LOB locator of the destination LOB instance.</td>
</tr>
<tr>
<td>src_blob</td>
<td>LOB locator of the source LOB instance.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes to convert from the source LOB. If you want to copy the entire BLOB, pass the constant DBMS_LOB.LOBMAXSIZE. If you pass any other value, it must be less than or equal to the size of the BLOB.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>Offset in characters in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB. (OUT) The new offset in characters after the end of the write. This offset always points to the beginning of the first complete character after the end of the write.</td>
</tr>
<tr>
<td>src_offset</td>
<td>Offset in bytes in the source LOB for the start of the read. (OUT) Offset in bytes in the source LOB right after the end of the read.</td>
</tr>
<tr>
<td>blob_csid</td>
<td>The character set ID of the source data</td>
</tr>
<tr>
<td>lang_context</td>
<td>Language context, such as shift status, for the current conversion. (OUT) The language context at the time when the current conversion is done. This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.</td>
</tr>
</tbody>
</table>
Table 93-19  (Cont.) CONVERTTOCLOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning</td>
<td>Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message. Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, ‘?’) is used in place of the inconvertible character. The return value of this error message is defined as the constant \texttt{warn_inconvertible_char} in the \texttt{DBMS_LOB} package.</td>
</tr>
</tbody>
</table>

Usage Notes

Preconditions

Before calling the \texttt{CONVERTTOCLOB} procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked before calling the \texttt{CONVERTTOCLOB} procedure. To lock the row, select the LOB using the \texttt{FOR UPDATE} clause of the \texttt{SELECT} statement.

Constants and Defaults

All parameters are required. You must pass a variable for each \texttt{OUT} or \texttt{IN OUT} parameter. You must pass either a variable or a value for each \texttt{IN} parameter.

\textbf{Table 93-20} gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the \texttt{dbmslob.sql} package specification file.

\textbf{Table 93-20  DBMS\_LOB.CONVERTTOCLOB Typical Values}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>\texttt{LOBMAXSIZE} (IN)</td>
<td>convert the entire file</td>
</tr>
<tr>
<td>dest_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
<tr>
<td>src_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
<tr>
<td>csid</td>
<td>\texttt{DEFAULT_CSID} (IN)</td>
<td>default \texttt{CSID}, use destination \texttt{CSID}</td>
</tr>
<tr>
<td>lang_context</td>
<td>\texttt{DEFAULT_LANG_CTX} (IN)</td>
<td>default language context</td>
</tr>
<tr>
<td>warning</td>
<td>\texttt{NO_WARNING} (OUT)</td>
<td>no warning message, success</td>
</tr>
<tr>
<td></td>
<td>\texttt{WARN_INCONVERTIBLE_CHAR} (OUT)</td>
<td>character in source cannot be properly converted</td>
</tr>
</tbody>
</table>

General Notes

- You must specify the desired character set for the source LOB in the \texttt{blob\_csid} parameter. You can pass a zero value for \texttt{blob\_csid}. When you do so, the database
assumes that the desired character set is the same as the destination LOB character set.

- You must specify the offsets for both the source and destination LOBs, and the number of characters to copy from the source LOB. The amount and src_offset values are in bytes and the dest_offset is in characters. To convert the entire LOB, you can specify LOBMAXSIZE for the amount parameter.

- CONVERTTOCLOB gets the source and/or destination LOBs as necessary prior to conversion and write of the data.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>One or more of the following:</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; LOBMAXSIZE.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for more information on using LOBs in application development

93.9.7 COPY Procedures

This procedure copies all, or a part of, a source internal LOB to a destination internal LOB. You can specify the offsets for both the source and destination LOBs, and the number of bytes or characters to copy.

Syntax

```
DBMS_LOB.COPY (  
    dest_lob    IN OUT NOCOPY BLOB,  
    src_lob     IN            BLOB,  
    amount      IN            INTEGER,  
    dest_offset IN            INTEGER := 1,  
    src_offset  IN            INTEGER := 1);  
```

```
DBMS_LOB.COPY (  
    dest_lob    IN OUT NOCOPY CLOB  CHARACTER SET ANY_CS,  
    src_lob     IN            CLOB  CHARACTER SET dest_lob%CHARSET,  
    amount      IN            INTEGER,  
    dest_offset IN            INTEGER := 1,  
    src_offset  IN            INTEGER := 1);  
```
Parameters

Table 93-22 COPY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>LOB locator of the copy target.</td>
</tr>
<tr>
<td>src_lob</td>
<td>LOB locator of source for the copy.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to copy.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>Offset in bytes or characters in the destination LOB (origin: 1) for the start of the copy.</td>
</tr>
<tr>
<td>src_offset</td>
<td>Offset in bytes or characters in the source LOB (origin: 1) for the start of the copy.</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-23 COPY Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or invalid.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &gt; LOBMAXSIZE</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; LOBMAXSIZE</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform a LOB write inside a query or PDML parallel execution server</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on either LOB</td>
</tr>
</tbody>
</table>

Usage Notes

- If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.
- It is not an error to specify an amount that exceeds the length of the data in the source LOB. Thus, you can specify a large amount to copy from the source LOB, which copies data from the src_offset to the end of the source LOB.
- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect
performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

- Prior to copy, the source and destination LOBs are retrieved, if they are currently archived. For a complete over-write, the destination LOB is not retrieved.
- If the source LOB is a DBFS Link, the data is streamed from DBFS, if possible, otherwise an exception is thrown. If the destination LOB is a DBFS Link, an exception is thrown.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.8 COPY_DBFS_LINK Procedures

This procedure copies the DBFS Link in the source LOB to the destination LOB.

Syntax

```sql
DBMS_LOB.COPY_DBFS_LINK (  
  lob_loc_dst    IN OUT BLOB,  
  lob_loc_src    IN     BLOB,  
  flags          IN     PLS_INTEGER DEFAULT DBFS_LINK_NOCACHE);  
```

```sql
DBMS_LOB.COPY_DBFS_LINK (  
  lob_loc_dst    IN OUT CLOB CHARACTER SET ANY_CS,  
  lob_loc_src    IN     CLOB CHARACTER SET ANY_CS,  
  flags          IN     PLS_INTEGER DEFAULT DBFS_LINK_NOCACHE);  
```

Parameters

Table 93-24  COPY_DBFS_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc_dst</td>
<td>LOB to be made to reference the same storage data as lob_loc_src</td>
</tr>
<tr>
<td>lob_loc_src</td>
<td>LOB from which to copy the reference</td>
</tr>
<tr>
<td>flags</td>
<td>Options to COPY_DBFS_LINK:</td>
</tr>
<tr>
<td></td>
<td>• DBFS_LINK_NOCACHE specifies to only copy the DBFS Link</td>
</tr>
<tr>
<td></td>
<td>• DBFS_LINK_CACHE specifies to copy the DBFS Link and read the data into the database LOB specified by lob_loc_dst so that the data is cached</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-25  COPY_DBFS_LINK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>Either lob_loc_src or lob_loc_dst is not a SECUREFILE</td>
</tr>
</tbody>
</table>
Table 93-25  (Cont.) COPY_DBFS_LINK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>lob_loc_src LOB has not been archived</td>
</tr>
<tr>
<td>ORA-01555</td>
<td>If the source LOB has been retrieved, never archived, or if the LOB has been migrated in and out (modified or not) since the locator was gotten.</td>
</tr>
</tbody>
</table>

93.9.9 COPY_FROM_DBFS_LINK

This procedure retrieves the archived SecureFiles LOB data from the DBFS HSM store and to the database.

Syntax

```plaintext
DBMS_LOB.COPY_FROM_DBFS_LINK (lob_loc IN OUT BLOB);
```

```plaintext
DBMS_LOB.COPY_FROM_DBFS_LINK (lob_loc IN OUT CLOB CHARACTER SET ANY_CS);
```

Parameters

Table 93-26  COPY_FROM_DBFS_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB to be retrieved from the archive</td>
</tr>
</tbody>
</table>

Usage Note

COPY_FROM_DBFS_LINK does not remove the underlying DBFS file.

If the LOB is successfully retrieved, COPY_FROM_DBFS_LINK silently returns success.

Exceptions

Table 93-27  COPY_FROM_DBFS_LINK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>lob_loc is not a SECUREFILE</td>
</tr>
<tr>
<td>ORA-01555</td>
<td>If the LOB has already been retrieved and has been modified since retrieval, if the LOB has been migrated in and out (modified or not) since the locator was retrieved</td>
</tr>
</tbody>
</table>
93.9.10 CREATETEMPORARY Procedures

This procedure creates a temporary BLOB or CLOB and its corresponding index in your default temporary tablespace.

Syntax

DBMS_LOB.CREATETEMPORARY (  
  lob_loc IN OUT NOCOPY BLOB,  
  cache IN BOOLEAN,  
  dur IN PLS_INTEGER := DBMS_LOB.SESSION);

DBMS_LOB.CREATETEMPORARY (  
  lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,  
  cache IN BOOLEAN,  
  dur IN PLS_INTEGER := 10);

Parameters

Table 93-28  CREATETEMPORARY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>cache</td>
<td>Specifies if LOB should be read into buffer cache or not.</td>
</tr>
<tr>
<td>dur</td>
<td>1 of 2 predefined duration values (SESSION or CALL) which specifies a hint as to whether the temporary LOB is cleaned up at the end of the session or call. If dur is omitted, then the session duration is used.</td>
</tr>
</tbody>
</table>

See Also:

- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure
- Oracle Database PL/SQL Language Reference for more information about NOCOPY and passing temporary lobs as parameters

93.9.11 DBFS_LINK_GENERATE_PATH Functions

This subprogram returns a unique file path name for use in creating a DBFS Link.

Syntax

DBMS_LOB.DBFS_LINK_GENERATE_PATH (  
  lob_loc IN BLOB,  
  storage_dir IN VARCHAR2)  
RETURN VARCHAR2;

DBMS_LOB.DBFS_LINK_GENERATE_PATH (  
  lob_loc IN CLOB CHARACTER SET ANY_CS,
storage_dir   IN VARCHAR2)
RETURN VARCHAR2;

Pragmas
PRAGMA RESTRICT_REFERENCES(dbfs_link_generate_path,
  WNDS, Rnds, WNPS, RNPS);

Parameters

Table 93-29  DBFS_LINK_GENERATE_PATH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB to be retrieved from DBFS</td>
</tr>
<tr>
<td>storage_dir</td>
<td>DBFS directory that will be the parent directory of the file</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-30  DBFS_LINK_GENERATE_PATH Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_WRONGTYPE</td>
<td>lob_loc is not a SECUREFILE</td>
</tr>
</tbody>
</table>

Usage Notes

Returns a globally unique file pathname that can be used for archiving. This is guaranteed to be globally unique across all calls to this function for different LOBs and versions of that LOB. It is always the same for the same LOB and version.

93.9.12 ERASE Procedures

This procedure erases an entire internal LOB or part of an internal LOB.

Syntax

DBMS_LOB.ERASE (lob_loc           IN OUT   NOCOPY   BLOB,
    amount            IN OUT   NOCOPY   INTEGER,
    offset            IN                INTEGER := 1);

DBMS_LOB.ERASE (lob_loc           IN OUT   NOCOPY   CLOB CHARACTER SET ANY_CS,
    amount            IN OUT   NOCOPY   INTEGER,
    offset            IN                INTEGER := 1);

Parameters

Table 93-31  ERASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be erased. For more information, see Operational Notes.</td>
</tr>
</tbody>
</table>
Table 93-31  (Cont.) ERASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs or BFILES) or characters (for CLOBs or NCLOBs) to be erased.</td>
</tr>
<tr>
<td>offset</td>
<td>Absolute offset (origin: 1) from the beginning of the LOB in bytes (for BLOBs) or characters (CLOBs).</td>
</tr>
</tbody>
</table>

Usage Notes

- When data is erased from the middle of a LOB, zero-byte fillers or spaces are written for BLOBs or CLOBs respectively.
- The actual number of bytes or characters erased can differ from the number you specified in the amount parameter if the end of the LOB value is reached before erasing the specified number. The actual number of characters or bytes erased is returned in the amount parameter.
- ERASE gets the LOB if it is archived, unless the erase covers the entire LOB.
- If the LOB to be erased is a DBFS Link, an exception is thrown.

Note:
The length of the LOB is not decreased when a section of the LOB is erased. To decrease the length of the LOB value, see the "TRIM Procedures".

Exceptions

Table 93-32  ERASE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any input parameter is NULL.</td>
</tr>
</tbody>
</table>
| INVALID_ARGVAL       | Either:
|                      | - amount < 1 or amount > LOBMAXSIZE |
|                      | - offset < 1 or offset > LOBMAXSIZE |
| QUERY_WRITE          | Cannot perform a LOB write inside a query or PDML parallel execution server |
| BUFFERING_ENABLED    | Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB |

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.

**See Also:**
- “TRIM Procedures”
- *Oracle Database SecureFiles and Large Objects Developer’s Guide* for additional details on usage of this procedure

### 93.9.13 FILECLOSE Procedure

This procedure closes a `BFILE` that has already been opened through the input locator.

**Note:**

The database has only read-only access to `BFILEs`. This means that `BFILEs` cannot be written through the database.

**Syntax**

```
DBMS_LOB.FILECLOSE (  
    file_loc IN OUT NOCOPY BFILE);
```

**Parameters**

**Table 93-33  FILECLOSE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the <code>BFILE</code> to be closed.</td>
</tr>
</tbody>
</table>

**Exceptions**

**Table 93-34  FILECLOSE Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>NULL input value for file_loc.</td>
</tr>
<tr>
<td>UNOPENED_FILE</td>
<td>File was not opened with the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
</tbody>
</table>
93.9.14 FILECLOSEALL Procedure

This procedure closes all BFILES opened in the session.

Syntax

```sql
DBMS_LOB.FILECLOSEALL;
```

Exceptions

Table 93-35 FILECLOSEALL Procedure Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>No file has been opened in the session.</td>
</tr>
</tbody>
</table>

See Also:

- "FILEOPEN Procedure"
- "FILECLOSE Procedure"
- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.15 FILEEXISTS Function

This function finds out if a specified BFILE locator points to a file that actually exists on the server's file system.

Syntax

```sql
DBMS_LOB.FILEEXISTS (
    file_loc    IN    BFILE
) RETURN INTEGER;
```

Pragmas

```sql
pragma restrict_references(FILEEXISTS, WNDS, RNDS, WNPS, RNPS);
```
Parameters

Table 93-36  FILEEXISTS Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE.</td>
</tr>
</tbody>
</table>

Return Values

Table 93-37  FILEEXISTS Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Physical file does not exist.</td>
</tr>
<tr>
<td>1</td>
<td>Physical file exists.</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-38  FILEEXISTS Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
</tbody>
</table>

See Also:

- "FILEISOPEN Function".
- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.16 FILEGETNAME Procedure

This procedure determines the directory object and filename, given a BFILE locator.

This function only indicates the directory object name and filename assigned to the locator, not if the physical file or directory actually exists.

The maximum constraint values for the dir_alias buffer is 30, and for the entire path name, it is 2000.

Syntax

DBMS_LOB.FILEGETNAME (   file_loc   IN   BFILE,    dir_alias OUT   VARCHAR2,    filename OUT   VARCHAR2);
Parameters

Table 93-39 FILEGETNAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE</td>
</tr>
<tr>
<td>dir_alias</td>
<td>Directory object name</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the BFILE</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-40 FILEGETNAME Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>dir_alias or filename are NULL.</td>
</tr>
</tbody>
</table>

See Also:
Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.17 FILEISOPEN Function

This function finds out whether a BFILE was opened with the specified FILE locator.

Syntax

```sql
DBMS_LOB.FILEISOPEN (  
    file_loc IN BFILE)  
RETURN INTEGER;
```

Pragmas

```sql
PRAGMA RESTRICT_REFERENCES(fileisopen, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 93-41 FILEISOPEN Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE</td>
</tr>
</tbody>
</table>

Return Values

INTEGER: 0 = file is not open, 1 = file is open
Usage Notes

If the input FILE locator was never passed to the FILEOPEN procedure, then the file is considered not to be opened by this locator. However, a different locator may have this file open. In other words, openness is associated with a specific locator.

Exceptions

Table 93-42  FILEISOPEN Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
</tbody>
</table>

See Also:

- "FILEEXISTS Function"
- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.18 FILEOPEN Procedure

This procedure opens a BFILE for read-only access. BFILE data may not be written through the database.

Syntax

```
DBMS_LOB.FILEOPEN (  
    file_loc   IN OUT NOCOPY  BFILE,  
    open_mode  IN             BINARY_INTEGER := file_readonly);  
```

Parameters

Table 93-43  FILEOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>Locator for the BFILE.</td>
</tr>
<tr>
<td>open_mode</td>
<td>File access is read-only.</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-44  FILEOPEN Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>file_loc or open_mode is NULL.</td>
</tr>
</tbody>
</table>
### Table 93-44 (Cont.) FILEOPEN Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td><code>open_mode</code> is not equal to <code>FILE_READONLY</code>.</td>
</tr>
<tr>
<td>OPEN_TOOMANY</td>
<td>Number of open files in the session exceeds <code>session_max_open_files</code>.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory associated with <code>file_loc</code> does not exist.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
</tbody>
</table>

### See Also:

- "FILECLOSE Procedure"
- "FILECLOSEALL Procedure"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

### 93.9.19 FRAGMENT_DELETE Procedure

This procedure deletes the data at the specified offset for the specified length from the LOB without having to rewrite all the data in the LOB following the specified offset.

#### Syntax

```sql
DBMS_LOB.FRAGMENT_DELETE (     
    lob_loc     IN OUT NOCOPY BLOB,     
    amount      IN            INTEGER,  
    offset      IN            INTEGER);
```

```sql
DBMS_LOB.FRAGMENT_DELETE (     
    lob_loc     IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,     
    amount      IN            INTEGER,  
    offset      IN            INTEGER);
```

#### Parameters

### Table 93-45 FRAGMENT_DELETE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (BLOB) or characters (CLOB/NCLOB) to be removed from the LOB</td>
</tr>
<tr>
<td>offset</td>
<td>Offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to begin the deletion</td>
</tr>
</tbody>
</table>
Exceptions

Table 93-46 FRAGMENT_DELETE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>A parameter value was invalid</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform operation during a query</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled</td>
</tr>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>A non-SECUREFILE LOB was used in a SECUREFILE LOB only call</td>
</tr>
<tr>
<td>SECUREFILE_OUTOFBOUNDS</td>
<td>Attempted to perform a FRAGMENT_* operation past LOB end</td>
</tr>
</tbody>
</table>

93.9.20 FRAGMENT_INSERT Procedures

This procedure inserts the specified data (limited to 32K) into the LOB at the specified offset.

Syntax

```sql
DBMS_LOB.FRAGMENT_INSERT (       
  lob_loc      IN OUT NOCOPY BLOB,       
  amount       IN            INTEGER,       
  offset       IN            INTEGER,       
  buffer       IN            RAW);

DBMS_LOB.FRAGMENT_INSERT (       
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,       
  amount       IN            INTEGER,       
  offset       IN            INTEGER,       
  buffer       IN            VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 93-47 FRAGMENT_INSERT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (BLOB) or characters (CLOB/NCLOB) to be inserted into the LOB</td>
</tr>
<tr>
<td>offset</td>
<td>Offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to be inserted into the LOB</td>
</tr>
<tr>
<td>buffer</td>
<td>Data to insert into the LOB</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-48 FRAGMENT_INSERT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>A parameter value was invalid</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform operation during a query</td>
</tr>
</tbody>
</table>
Table 93-48  (Cont.) FRAGMENT_INSERT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled</td>
</tr>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>A non-SECUREFILE LOB was used in a SECUREFILE LOB only call</td>
</tr>
<tr>
<td>SECUREFILE_OUTOFBOUNDS</td>
<td>Attempted to perform a FRAGMENT_* operation past LOB end</td>
</tr>
</tbody>
</table>

Usage Notes

FRAGMENT_INSERT gets the LOB, if necessary, before performing operations on the LOB.

93.9.21 FRAGMENT_MOVE Procedure

This procedure moves the amount of bytes (BLOB) or characters (CLOB/NCLOB) from the specified offset to the new offset specified.

Syntax

```sql
DBMS_LOB.FRAGMENT_MOVE (    
    lob_loc       IN OUT NOCOPY BLOB,
    amount        IN            INTEGER,
    src_offset    IN            INTEGER,
    dest_offset   IN            INTEGER);

DBMS_LOB.FRAGMENT_MOVE (    
    lob_loc       IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
    amount        IN            INTEGER,
    src_offset    IN            INTEGER,
    dest_offset   IN            INTEGER);
```

Parameters

Table 93-49  FRAGMENT_MOVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (BLOB) or characters (CLOB/NCLOB) to be moved in the LOB</td>
</tr>
<tr>
<td>src_offset</td>
<td>Beginning offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to put the data</td>
</tr>
<tr>
<td>dest_offset</td>
<td>Beginning offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to remove the data</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-50  FRAGMENT_MOVE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>A parameter value was invalid</td>
</tr>
</tbody>
</table>
Table 93-50 (Cont.) FRAGMENT_MOVE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform operation during a query</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled</td>
</tr>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>A non-SECUREFILE LOB was used in a SECUREFILE LOB only call</td>
</tr>
<tr>
<td>SECUREFILE_OUTOFBOUNDS</td>
<td>Attempted to perform a FRAGMENT_* operation past LOB end</td>
</tr>
</tbody>
</table>

Usage Notes

- All offsets are pre-move offsets.
- Offsets of more than 1 past the end of the LOB are not permitted.
- FRAGMENT_MOVE gets the LOB, if necessary, before performing operations on the LOB.

93.9.22 FRAGMENT_REPLACE Procedures

This procedure replaces the data at the specified offset with the specified data (not to exceed 32k).

Syntax

```sql
DBMS_LOB.FRAGMENT_REPLACE (  
  lob_loc     IN OUT NOCOPY BLOB,  
  old_amount  IN            INTEGER,  
  new_amount  IN            INTEGER,  
  offset      IN            INTEGER,  
  buffer      IN            RAW);
```

```sql
DBMS_LOB.FRAGMENT_REPLACE (  
  lob_loc     IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,  
  old_amount IN            INTEGER,  
  new_amount IN            INTEGER,  
  offset      IN            INTEGER,  
  buffer      IN            VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 93-51 FRAGMENT_REPLACE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>old_amount</td>
<td>Number of bytes (BLOB) or characters (CLOB/NCLOB) to be replaced in the LOB</td>
</tr>
<tr>
<td>new_amount</td>
<td>Number of bytes (BLOB) or characters (CLOB/NCLOB) to written to the LOB</td>
</tr>
<tr>
<td>offset</td>
<td>Beginning offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to put the data</td>
</tr>
<tr>
<td>buffer</td>
<td>Data to insert into the LOB</td>
</tr>
</tbody>
</table>
Exceptions

Table 93-52  FRAGMENT_REPLACE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>A parameter value was invalid</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform operation during a query</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled</td>
</tr>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>A non-SECUREFILE LOB was used in a SECUREFILE LOB only call</td>
</tr>
<tr>
<td>SECUREFILE_OUTOFPBOUNDS</td>
<td>Attempted to perform a FRAGMENT_* operation past LOB end</td>
</tr>
</tbody>
</table>

Usage Notes

- Invoking this procedure is equivalent to deleting the old amount of bytes/characters at offset and then inserting the new amount of bytes/characters at offset.
- FRAGMENT_REPLACE gets the LOB, if necessary, before performing operations on the LOB.

93.9.23 FREETEMPORARY Procedures

This procedure frees the temporary BLOB or CLOB in the default temporary tablespace.

Syntax

```sql
DBMS_LOB.FREETEMPORARY (    lob_loc IN OUT NOCOPY BLOB);

DBMS_LOB.FREETEMPORARY (    lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);
```

Parameters

Table 93-53  FREETEMPORARY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
</tbody>
</table>

Usage Notes

- When a new temporary LOB is created, and there is currently no temporary LOB in use with the same duration (session, call), a new temporary LOB segment is created. When the temporary LOB is freed, the space it consumed is released to the temporary segment. If there are no other temporary LOBs for the same duration, the temporary segment is also freed.
- After the call to FREETEMPORARY, the LOB locator that was freed is marked as invalid.
If an invalid LOB locator is assigned to another LOB locator using `OCILobLocatorAssign` in OCI or through an assignment operation in PL/SQL, then the target of the assignment is also freed and marked as invalid.

**See Also:**

*Oracle Database SecureFiles and Large Objects Developer’s Guide* for additional details on usage of this procedure

---

### 93.9.24 GET_DBFS_LINK Functions

This function returns the DBFS path name for the specified SecureFile LOB.

**Syntax**

```sql
DBMS_LOB.GET_DBFS_LINK (
  lob_loc             IN     BLOB,
  storage_path        OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),
  lob_length          OUT NUMBER);

DBMS_LOB.GET_DBFS_LINK (
  lob_loc             IN     CLOB CHARACTER SET ANY_CS,
  storage_path        OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),
  lob_length          OUT NUMBER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lob_loc</code></td>
<td>LOB to be retrieved from DBFS</td>
</tr>
<tr>
<td><code>storage_path</code></td>
<td>Path where the LOB is stored in DBFS</td>
</tr>
<tr>
<td><code>lob_length</code></td>
<td>LOB length at the time of write to DBFS</td>
</tr>
</tbody>
</table>

**Return Values**

The Archive ID

**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td><code>lob_loc</code> is not a SECUREFILE</td>
</tr>
<tr>
<td>ORA-01555</td>
<td>The LOB has already been retrieved and has been modified since retrieval or the LOB has been migrated in and out (modified or not) since the locator was retrieved</td>
</tr>
</tbody>
</table>
93.9.25 GET_DBFS_LINK_STATE Procedures

GET_DBFS_LINK_STATE retrieves the current link state of the specified SecureFile.

Syntax

```sql
DBMS_LOB.GET_DBFS_LINK_STATE (  
  lob_loc       IN BLOB,  
  storage_path  OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),  
  state         OUT NUMBER,  
  cached        OUT BOOLEAN);

DBMS_LOB.GET_DBFS_LINK_STATE (  
  lob_loc       IN CLOB CHARACTER SET ANY_CS,  
  storage_path  OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),  
  state         OUT NUMBER,  
  cached        OUT BOOLEAN);
```

Parameters

**Table 93-56  GET_DBFS_LINK_STATE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB to be retrieved from the archive</td>
</tr>
<tr>
<td>storage_path</td>
<td>Path where the LOB is stored in the DBFS HSM store</td>
</tr>
<tr>
<td>state</td>
<td>One of DBMS_LOB.DBFS_LINK_NEVER, DBMS_LOB.DBFS_LINK_NO or DBMS_LOB.DBFS_LINK_YES</td>
</tr>
<tr>
<td>cached</td>
<td>If the LOB is archived and the data was specified to be cached on put</td>
</tr>
</tbody>
</table>

Exceptions

**Table 93-57  GET_DBFS_LINK_STATE Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>lob_loc is not a SECUREFILE</td>
</tr>
</tbody>
</table>

Usage Notes

- If the LOB has never been archived, state is set to DBMS_LOB.DBFS_LINK_NEVER. If the LOB has been archived, state is set to DBMS_LOB.DBFS_LINK_YES. If the LOB has been previously retrieved from the archive, state is set to DBMS_LOB.DBFS_LINK_NO.
- If the LOB was archived, but the data was left in the RDBMS, cached is set to TRUE. If the data was removed after the link was created, cached is set to FALSE, and NULL if state is DBMS_LOB.DBFS_LINK_NEVER.
93.9.26 GETCONTENTTYPE Functions

This procedure returns the content type string previously set by means of the SETCONTENTTYPE Procedure.

Syntax

```sql
DBMS_LOB.GETCONTENTTYPE (    lob_loc  IN BLOB)    RETURN VARCHAR2;

DBMS_LOB.GETCONTENTTYPE (    lob_loc  IN CLOB CHARACTER SET ANY_CS)    RETURN VARCHAR2;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(getcontenttype, WDNS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB whose content type is to be retrieved</td>
</tr>
</tbody>
</table>

Return Values

The returned content type.

If the SecureFiles LOB does not have a contenttype associated with it, GETCONTENTTYPE() returns NULL.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>lob_loc is not a SECUREFILE</td>
</tr>
</tbody>
</table>

Related Topics

- SETCONTENTTYPE Procedure
  This procedure sets the content type string for the data in the LOB.

93.9.27 GET_STORAGE_LIMIT Function

This function returns the LOB storage limit for the specified LOB.

Syntax

```sql
DBMS_LOB.GET_STORAGE_LIMIT (    lob_loc  IN CLOB CHARACTER SET ANY_CS)    RETURN INTEGER;
```
DBMS_LOB.GET_STORAGE_LIMIT (  
    lob_loc  IN BLOB)  
RETURN INTEGER;

Pragmas
PRAGMA RESTRICT_REFERENCES(get_storage_limit, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 93-60 GET_STORAGE_LIMIT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
</tbody>
</table>

Return Value

The value returned from this function is the maximum allowable size for specified LOB locator. For BLOBs, the return value depends on the block size of the tablespace the LOB resides in and is calculated as \((2^{32})-1\) \((4294967295)\) times the block size of the tablespace. For CLOBs/NCLOBs, the value returned is \((2^{32})-1\) \((4294967295)\) times the block size of the tablespace divided by the character width of the CLOB/NCLOB.

Usage

See Also:
Oracle Database SecureFiles and Large Objects Developer's Guide for details on LOB storage limits

93.9.28 GETCHUNKSIZE Functions

When creating the table, you can specify the chunking factor, a multiple of tablespace blocks in bytes. This corresponds to the chunk size used by the LOB data layer when accessing or modifying the LOB value. Part of the chunk is used to store system-related information, and the rest stores the LOB value. This function returns the amount of space used in the LOB chunk to store the LOB value.

Syntax

DBMS_LOB.GETCHUNKSIZE (  
    lob_loc IN BLOB)  
RETURN INTEGER;

DBMS_LOB.GETCHUNKSIZE (  
    lob_loc IN CLOB CHARACTER SET ANY_CS)  
RETURN INTEGER;

Pragmas
PRAGMA RESTRICT_REFERENCES(getchunksize, WNDS, RNDS, WNPS, RNPS);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
</tbody>
</table>

Return Values

The return value is a usable chunk size in bytes.

Usage Notes

- With regard to basic LOB files, performance is improved if you enter read/write requests using a multiple of this chunk size. For writes, there is an added benefit, because LOB chunks are versioned, and if all writes are done on a chunk basis, then no extra or excess versioning is done or duplicated. You could batch up the WRITE until you have enough for a chunk, instead of issuing several WRITE calls for the same chunk.

These tactics of performance improvement do not apply to SecureFiles.

- Note that chunk size is independent of LOB type (BLOB, CLOB, NCLOB, Unicode or other character set).

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB</td>
</tr>
</tbody>
</table>

93.9.29 GETLENGTH Functions

This function gets the length of the specified LOB. The length in bytes or characters is returned.

The length returned for a BFILE includes the EOF, if it exists. Any 0-byte or space filler in the LOB caused by previous ERASE or WRITE operations is also included in the length count. The length of an empty internal LOB is 0.

Syntax

```sql
DBMS_LOB.GETLENGTH (    lob_loc    IN  BLOB)    RETURN INTEGER;
```
DBMS_LOB.GETLENGTH (  
lob_loc IN CLOB CHARACTER SET ANY_CS)  
RETURN INTEGER;

DBMS_LOB.GETLENGTH (  
file_loc IN BFILE)  
RETURN INTEGER;

Pragmas
pragma restrict_references(GETLENGTH, WNDS, WNPS, RNDS, RNPS);

Parameters

Table 93-63 GETLENGTH Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB whose length is to be returned.</td>
</tr>
</tbody>
</table>

Return Values

The length of the LOB in bytes or characters as an INTEGER. NULL is returned if the input LOB is NULL or if the input lob_loc is NULL. An error is returned in the following cases for BFILES:

- lob_loc does not have the necessary directory and operating system privileges
- lob_loc cannot be read because of an operating system read error

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

Exceptions

Table 93-64 GETLENGTH Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB</td>
</tr>
</tbody>
</table>

93.9.30 GETOPTIONS Functions

This function obtains compression, deduplication, and encryption settings corresponding to the option_type field for a particular LOB.

Syntax

DBMS_LOB.GETOPTIONS (  
lob_loc IN BLOB,
option_types        IN     PLS_INTEGER)
RETURN PLS_INTEGER;

DBMS_LOB.GETOPTIONS (  
   lob_loc             IN     CLOB CHARACTER SET ANY_CS,  
   option_types        IN     PLS_INTEGER)  
RETURN PLS_INTEGER;

**Parameters**

**Table 93-65  GETOPTIONS Function Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be examined. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>option_type</td>
<td>See Table 93-2</td>
</tr>
</tbody>
</table>

**Return Values**

The return values are a combination of COMPRESS_ON, ENCRYPT_ON and DEDUPLICATE_ON (see Table 93-3) depending on which option types (see Table 93-2) are passed in.

**Exceptions**

**Table 93-66  GETOPTIONS Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_ARGVAL</td>
<td>A parameter value was invalid</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform operation during a query</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled</td>
</tr>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>A non-SECUREFILE LOB was used in a SECUREFILE LOB only call</td>
</tr>
</tbody>
</table>

**Usage Notes**

You cannot turn compression or deduplication on or off for a SecureFile column that does not have those features on. The GetOptions Functions and SETOPTIONS Procedures work on individual SecureFiles. You can turn off a feature on a particular SecureFile and turn on a feature that has already been turned off by SetOptions, but you cannot turn on an option that has not been given to the SecureFile when the table was created.

**93.9.31 INSTR Functions**

This function returns the matching position of the *nth* occurrence of the pattern in the LOB, starting from the offset you specify.

**Syntax**

DBMS_LOB.INSTR (  
   lob_loc    IN   BLOB,  
   pattern    IN   RAW,  
   offset     IN   INTEGER := 1,  
   nth        IN   INTEGER := 1)  
RETURN INTEGER;
DBMS_LOB.INSTR (  
  lob_loc   IN   CLOB   CHARACTER SET ANY_CS,  
  pattern   IN   VARCHAR2 CHARACTER SET lob_loc%CHARSET,  
  offset    IN   INTEGER := 1,  
  nth       IN   INTEGER := 1)  
RETURN INTEGER;

DBMS_LOB.INSTR (  
  file_loc   IN   BFILE,  
  pattern    IN   RAW,  
  offset     IN   INTEGER := 1,  
  nth        IN   INTEGER := 1)  
RETURN INTEGER;

Pragmas
pragma restrict_references(INSTR, WNDS, WNPS, RNDS, RNPS);

Parameters

Table 93-67  INSTR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be examined. For more information, see <a href="#">Operational Notes</a>.</td>
</tr>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB to be examined.</td>
</tr>
<tr>
<td>pattern</td>
<td>Pattern to be tested for. The pattern is a group of RAW bytes for BLOBs, and a character string (VARCHAR2) for CLOBs. The maximum size of the pattern is 16383 bytes.</td>
</tr>
<tr>
<td>offset</td>
<td>Absolute offset in bytes (BLOBs) or characters (CLOBs) at which the pattern matching is to start. (origin: 1)</td>
</tr>
<tr>
<td>nth</td>
<td>Occurrence number, starting at 1.</td>
</tr>
</tbody>
</table>

Return Values

Table 93-68  INSTR Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Offset of the start of the matched pattern, in bytes or characters. It returns 0 if the pattern is not found.</td>
</tr>
</tbody>
</table>
| NULL     | Either:  
- any one or more of the IN parameters was NULL or INVALID.  
- offset < 1 or offset > LOBMAXSIZE.  
- nth < 1.  
- nth > LOBMAXSIZE. |

Usage Notes

The form of the VARCHAR2 buffer (the pattern parameter) must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the
buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

Operations that accept RAW or VARCHAR2 parameters for pattern matching, such as INSTR, do not support regular expressions or special matching characters (as in the case of SQL LIKE) in the pattern parameter or substrings.

Exceptions

Table 93-69  INSTR Function Exceptions for BFILES

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>File was not opened using the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB</td>
</tr>
</tbody>
</table>

See Also:

- "SUBSTR Functions"
- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.32 ISOPEN Functions

This function checks to see if the LOB was already opened using the input locator. This subprogram is for internal and external LOBs.

Syntax

```sql
DBMS_LOB.ISOPEN (lob_loc IN BLOB) RETURN INTEGER;

DBMS_LOB.ISOPEN (lob_loc IN CLOB CHARACTER SET ANY_CS) RETURN INTEGER;

DBMS_LOB.ISOPEN (file_loc IN BFILE) RETURN INTEGER;
```
Pragmas

PRAGMA RESTRICT_REFERENCES(isopen, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 93-70  ISOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>file_loc</td>
<td>File locator.</td>
</tr>
</tbody>
</table>

Return Values

The return value is 1 if the LOB is open, 0 otherwise.

Usage Notes

For BFILES, openness is associated with the locator. If the input locator was never passed to OPEN, the BFILE is not considered to be opened by this locator. However, a different locator may have opened the BFILE. More than one OPEN can be performed on the same BFILE using different locators.

For internal LOBs, openness is associated with the LOB, not with the locator. If locator1 opened the LOB, then locator2 also sees the LOB as open. For internal LOBs, ISOPEN requires a round-trip, because it checks the state on the server to see if the LOB is indeed open.

For external LOBs (BFILEs), ISOPEN also requires a round-trip, because that's where the state is kept.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.33 ISREMOTE Function

This function checks to see if the LOB is local to the database or if it belongs to a remote database.

Syntax

```sql
DBMS_LOB.ISREMOTE (    lob_loc IN BLOB)    RETURN BOOLEAN;

DBMS_LOB.ISREMOTE (    lob_loc IN CLOB CHARACTER SET ANY_CS)    RETURN BOOLEAN;
```
Pragmas

PRAGMA RESTRICT_REFERENCES(isremote, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 93-71    ISREMOTE Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB.</td>
</tr>
</tbody>
</table>

Return Values

BOOLEAN: TRUE for remote LOBs obtained over a database link; FALSE for LOBs obtained from local database

See Also:

- *Distributed LOBs* chapter in *Database SecureFiles and Large Objects Developer's Guide* for more details on the usage of this procedure.

93.9.34 ISSECUREFILE Function

This function returns TRUE if the LOB locator passed to it is for a SecureFile LOB. It returns FALSE otherwise.

Syntax

```
DBMS_LOB ISSECUREFILE(
    lob_loc IN     BLOB)
RETURN BOOLEAN;
```

Pragmas

PRAGMA RESTRICT_REFERENCES(issecurefile, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 93-72    ISSECUREFILE Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see <em>Operational Notes</em>.</td>
</tr>
</tbody>
</table>

Return Values

This function returns TRUE if the LOB locator passed to it is for a SecureFile LOB. It returns FALSE otherwise.
93.9.35 ISTEMPORARY Functions

This function determines whether a LOB instance is temporary.

Syntax

```
DBMS_LOB.ISTEMPORARY (    
   lob_loc IN BLOB)    
RETURN INTEGER;
```

```
DBMS_LOB.ISTEMPORARY (    
   lob_loc IN CLOB CHARACTER SET ANY_CS)    
RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(istemporary, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 93-73  ISTEMPORARY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
</tbody>
</table>

Return Values

The return value is 1 if the LOB is temporary and exists; 0 if the LOB is not temporary or does not exist; NULL if the given locator is NULL.

Usage Notes

When you free a Temporary LOB with FREETEMPORARY, the LOB locator is not set to NULL. Consequently, ISTEMPORARY will return 0 for a locator that has been freed but not explicitly reset to NULL.

See Also:

- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

93.9.36 LOADBLOBFROMFILE Procedure

This procedure loads data from BFILE to internal BLOB. This achieves the same outcome as LOADFROMFILE, and returns the new offsets.

Syntax

```
DBMS_LOB.LOADBLOBFROMFILE (                      
   dest_lob   IN OUT NOCOPY BLOB,    
   src_bfile IN BFILE,    
   amount    IN INTEGER,    
```

Chapter 93  Summary of DBMS_LOB Subprograms

93-52
Parameters

Table 93-74  LOADBLOBFROMFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>BLOB locator of the target for the load.</td>
</tr>
<tr>
<td>src_bfile</td>
<td>BFILE locator of the source for the load.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes to load from the BFILE. You can also use DBMS_LOB.LOBMAXSIZE to load until the end of the BFILE.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>(IN) Offset in bytes in the destination BLOB (origin: 1) for the start of the write. (OUT) New offset in bytes in the destination BLOB right after the end of this write, which is also where the next write should begin.</td>
</tr>
<tr>
<td>src_offset</td>
<td>(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read. (OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.</td>
</tr>
</tbody>
</table>

Usage Notes

- You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is in bytes for BLOBs.
- If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.
- There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is LOBMAXSIZE which you can specify to continue loading until the end of the BFILE is reached).
- It is not mandatory that you wrap the LOB operation inside the OPEN/CLOSE operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the OPEN/CLOSE, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.
- LOADBLOBFROMFILE gets the destination LOB prior to the load unless the load covers the entire LOB.

Constants and Defaults

There is no easy way to omit parameters. You must either declare a variable for IN/OUT parameter or provide a default value for the IN parameter. Here is a summary of the constants and the defaults that can be used.
### Table 93-75  Suggested Values of the Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>DBMS_LOB.LOBMAXSIZE (IN)</td>
<td>Load the entire file</td>
</tr>
<tr>
<td>dest_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
<tr>
<td>src_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
</tbody>
</table>

Constants defined in DBMSLOB.SQL

lobmaxsize CONSTANT INTEGER := DBMS_LOB.LOBMAXSIZE;

### Exceptions

#### Table 93-76  LOADBLOBFROMFILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the BLOB</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database SecureFiles and Large Objects Developer’s Guide for additional details on usage of this procedure

### 93.9.37 LOADCLOBFROMFILE Procedure

This procedure loads data from a BFILE to an internal CLOB/NCLOB with necessary character set conversion and returns the new offsets.

#### Syntax

```sql
DBMS_LOB.LOADCLOBFROMFILE (  
  dest_lob IN OUT NOCOPY   NOCOPY CLOB CHARACTER SET ANY_CS,  
  src_bfile IN              BFILE,  
  amount IN              INTEGER,  
  dest_offset IN OUT      INTEGER,  
  src_offset IN OUT      INTEGER,  
  bfile_csid IN              NUMBER,  
  lang_context IN OUT      INTEGER,  
  warning OUT              INTEGER);  
```
Parameters

Table 93-77     LOADCLOBFROMFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>CLOB/NCLOB locator of the target for the load.</td>
</tr>
<tr>
<td>src_bfile</td>
<td>BFILE locator of the source for the load.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes to load from the BFILE. Use DBMS_LOB.LOBMAXSIZE of load until the end of the BFILE.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>(IN) Offset in characters in the destination CLOB (origin: 1) for the start of the write. (OUT) The new offset in characters right after the end of this load, which is also where the next load should start. It always points to the beginning of the first complete character after the end of load. If the last character is not complete, offset goes back to the beginning of the partial character.</td>
</tr>
<tr>
<td>src_offset</td>
<td>(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read. (OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.</td>
</tr>
<tr>
<td>bfile_csid</td>
<td>Character set id of the source (BFILE) file.</td>
</tr>
<tr>
<td>lang_context</td>
<td>(IN) Language context, such as shift status, for the current load. (OUT) The language context at the time when the current load stopped, and what the next load should be using if continuing loading from the same source. This information is returned to the user so that they can use it for the continuous load without losing or misinterpreting any source data. For the very first load or if do not care, simply use the default 0. The details of this language context is hidden from the user. One does not need to know what it is or what's in it in order to make the call</td>
</tr>
<tr>
<td>warning</td>
<td>(OUT) Warning message. This indicates something abnormal happened during the loading. It may or may not be caused by the user's mistake. The loading is completed as required, and it's up to the user to check the warning message. Currently, the only possible warning is the inconvertible character. This happens when the character in the source cannot be properly converted to a character in destination, and the default replacement character (for example, '?' is used in place. The message is defined the constant value DBMS_LOB.WARN_INCONVERTIBLE_CHAR.</td>
</tr>
</tbody>
</table>

Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is in characters for CLOBs.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination CLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is LOBMAXSIZE which you can specify to continue loading until the end of the BFILE is reached).

Note the following requirements:
• The destination character set is always the same as the database character set in the case of CLOB and national character set in the case of NCLOB.

• `csid=0` indicates the default behavior that uses database `csid` for CLOB and national `csid` for NCLOB in the place of source `csid`. Conversion is still necessary if it is of varying width.

• It is not mandatory that you wrap the LOB operation inside the `OPEN/CLOSE` operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the `OPEN/CLOSE`, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.

The source BFILE can contain data in the Unicode character set. The Unicode standard defines many encoding schemes that provide mappings from Unicode characters to sequences of bytes. Table 93-78 lists Unicode encodings schemes supported by this subprogram.

### Table 93-78 Supported Unicode Encoding Schemes

<table>
<thead>
<tr>
<th>Encoding Scheme</th>
<th>Oracle Name</th>
<th>bfile_csid Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTF-8</td>
<td>AL32UTF8</td>
<td>873</td>
</tr>
<tr>
<td>UTF-16BE</td>
<td>AL16UTF16</td>
<td>2000</td>
</tr>
<tr>
<td>UTF-16LE</td>
<td>AL16UTF16LE</td>
<td>2002</td>
</tr>
<tr>
<td>CESU-8</td>
<td>UTF8</td>
<td>871</td>
</tr>
<tr>
<td>UTF-EBCDIC</td>
<td>UTFE</td>
<td>872</td>
</tr>
<tr>
<td>UTF-16</td>
<td>UTF16</td>
<td>1000</td>
</tr>
</tbody>
</table>

All three UTF-16 encoding schemes encode Unicode characters as 2-byte unsigned integers. Integers can be stored in big-endian or in little-endian byte order. The UTF-16BE encoding scheme defines big-endian data. The UTF-16LE scheme defines little-endian data. The UTF-16 scheme requires that the source BFILE contains the Byte Order Mark (BOM) character in the first two bytes to define the byte order. The BOM code is 0xFEFF. If the code is stored as `{0xFE,0xFF}`, the data is interpreted as big-endian. If it is stored as `{0xFF,0xFE}`, the data is interpreted as little-endian.

In UTF-8 and in CESU-8 encodings the Byte Order Mark is stored as `{0xEF,0xBB,0xBF}`. With any of the Unicode encodings, the corresponding BOM sequence at the beginning of the file is recognized and not loaded into the destination LOB.

### Constants

Here is a summary of the constants and the suggested values that can be used.
### Table 93-79  Suggested Values of the LOADCLOBFROMFILE Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Suggested Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>DBMS_LOB.LOBMAXSIZE (IN)</td>
<td>Load the entire file</td>
</tr>
<tr>
<td>dest_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
<tr>
<td>src_offset</td>
<td>1 (IN)</td>
<td>start from the beginning</td>
</tr>
<tr>
<td>csid</td>
<td>0 (IN)</td>
<td>default csid, use destination csid</td>
</tr>
<tr>
<td>lang_context</td>
<td>0 (IN)</td>
<td>default language context</td>
</tr>
<tr>
<td>warning</td>
<td>0 (OUT)</td>
<td>no warning message, everything is ok</td>
</tr>
</tbody>
</table>

#### Constants defined in DBMSLOB.SQL

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lobmaxsize</td>
<td>CONSTANT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>warn_inconvertible_char</td>
<td>CONSTANT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>default_csid</td>
<td>CONSTANT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>default_lang_ctx</td>
<td>CONSTANT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>no_warning</td>
<td>CONSTANT</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>

### Exceptions

#### Table 93-80  LOADCLOBFROMFILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- src_offset or dest_offset &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the CLOB</td>
</tr>
</tbody>
</table>
93.9.38 LOADFROMFILE Procedure

This deprecated procedure copies all, or a part of, a source external LOB (BFILE) to a destination internal LOB.

Note:

This procedure has been deprecated starting in Oracle Database 12c release 12.2.

Syntax

\[
\text{DBMS_LOB.LOADFROMFILE} (\text{dest_lob, src_file, amount, dest_offset, src_offset})
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_lob</td>
<td>LOB locator of the target for the load.</td>
</tr>
<tr>
<td>src_file</td>
<td>BFILE locator of the source for the load.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes to load from the BFILE.</td>
</tr>
<tr>
<td>dest_offset</td>
<td>Offset in bytes or characters in the destination LOB (origin: 1) for the start of the load.</td>
</tr>
<tr>
<td>src_offset</td>
<td>Offset in bytes in the source BFILE (origin: 1) for the start of the load.</td>
</tr>
</tbody>
</table>

Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is either in bytes or characters for BLOBs and CLOBs respectively.

Note:

The input BFILE must have been opened prior to using this procedure. No character set conversions are performed implicitly when binary BFILE data is loaded into a CLOB. The BFILE data must already be in the same character set as the CLOB in the database. No error checking is performed to verify this.
If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE.

Note:

If the character set is varying width, UTF-8 for example, the LOB value is stored in the fixed-width UCS2 format. Therefore, if you are using `DBMS_LOB.LOADFROMFILE`, the data in the BFILE should be in the UCS2 character set instead of the UTF-8 character set. However, you should use `sql*loader` instead of `LOADFROMFILE` to load data into a CLOB or NCLOB because `sql*loader` provides the necessary character set conversions.

It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.

Exceptions

Table 93-82  LOADFROMFILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of the input parameters are NULL or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- <code>src_offset</code> or <code>dest_offset</code> &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- <code>src_offset</code> or <code>dest_offset</code> &gt; LOBMAXSIZE.</td>
</tr>
<tr>
<td></td>
<td>- <code>amount</code> &lt; 1.</td>
</tr>
<tr>
<td></td>
<td>- <code>amount</code> &gt; LOBMAXSIZE.</td>
</tr>
</tbody>
</table>

See Also:

*Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure
93.9.39 MOVE_TO_DBFS_LINK Procedures

This procedure archives the specified LOB data (from the database) into the DBFS HSM Store.

Syntax

```sql
DBMS_LOB.MOVE_TO_DBFS_LINK (
    lob_loc       IN OUT BLOB,
    storage_path  IN     VARCHAR2(dbfs_link_path_max_size),
    flags         IN     BINARY INTEGER DEFAULT DBFS_LINK_NOCACHE);
```

```sql
DBMS_LOB.MOVE_TO_DBFS_LINK (
    lob_loc        IN OUT CLOB CHARACTER SET ANY_CS,
    storage_path   IN     VARCHAR2(dbfs_link_path_max_size),
    flags           IN     BINARY INTEGER DEFAULT DBFS_LINK_NOCACHE);
```

Parameters

Table 93-83  MOVE_TO_DBFS_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB to be archived</td>
</tr>
<tr>
<td>storage_path</td>
<td>Path where the LOB will be stored</td>
</tr>
<tr>
<td>flags</td>
<td>Either DBFS_LINK_CACHE or DBFS_LINK_NOCACHE. If DBFS_LINK_CACHE is specified, the LOB data continues to be stored in the RDBMS as well as being written to the DBFS store. DBFS_LINK_NOCACHE specifies that the LOB data should be deleted from the RDBMS once written to the DBFS.</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-84  MOVE_TO_DBFS_LINK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>lob_loc is not a SECUREFILE</td>
</tr>
</tbody>
</table>

Usage Notes

- If the LOB is already archived, the procedure silently returns as if the put was successful. In that case, if DBFS_LINK_NOCACHE is specified, or flags is defaulted, the LOB data is removed from the RDBMS.
- Calling this procedure multiple times on the same LOB with the same flags has no effect.
- Calling the procedure on a LOB that is already archived causes the LOB to be cached (DBFS_LINK_CACHE) or removed (DBFS_LINK_NOCACHE) according to the flag setting.
93.9.40 OPEN Procedures

This procedure opens a LOB, internal or external, in the indicated mode. Valid modes include read-only, and read/write.

Syntax

```sql
DBMS_LOB.OPEN (  
    lob_loc   IN OUT NOCOPY BLOB,  
    open_mode IN BINARY_INTEGER);

DBMS_LOB.OPEN (  
    lob_loc   IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,  
    open_mode IN BINARY_INTEGER);

DBMS_LOB.OPEN (  
    file_loc  IN OUT NOCOPY BFILE,  
    open_mode IN BINARY_INTEGER := file_readonly);
```

Parameters

Table 93-85 OPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB locator. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>open_mode</td>
<td>Mode in which to open. For BLOB and CLOB types, the mode can be either: LOB_READONLY or LOB_READWRITE. For BFILE types, the mode must be FILE_READONLY.</td>
</tr>
</tbody>
</table>

Usage Notes

**Note:**

If the LOB was opened in read-only mode, and if you try to write to the LOB, then an error is returned. BFILE can only be opened with read-only mode.

OPEN requires a round-trip to the server for both internal and external LOBs. For internal LOBs, OPEN triggers other code that relies on the OPEN call. For external LOBs (BFILES), OPEN requires a round-trip because the actual operating system file on the server side is being opened.

It is not mandatory that you wrap all LOB operations inside the Open/Close interfaces. However, if you open a LOB, you must close it before you commit the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are...
not updated. If this happens, you should rebuild the functional and domain indexes on
the LOB column.

See Also:
Oracle Database SecureFiles and Large Objects Developer’s Guide for 
additional details on usage of this procedure

93.9.41 READ Procedures

This procedure reads a piece of a LOB, and returns the specified amount into the 
buffer parameter, starting from an absolute offset from the beginning of the LOB.

The number of bytes or characters actually read is returned in the amount parameter. If 
the input offset points past the End of LOB, then amount is set to 0, and a 
NO_DATA_FOUND exception is raised.

Syntax

DBMS_LOB.READ (
    lob_loc IN             BLOB,
    amount    IN OUT  NOCOPY INTEGER,
    offset    IN             INTEGER,
    buffer    OUT            RAW);

DBMS_LOB.READ (
    lob_loc IN             CLOB CHARACTER SET ANY_CS,
    amount    IN OUT  NOCOPY INTEGER,
    offset    IN             INTEGER,
    buffer    OUT            VARCHAR2 CHARACTER SET lob_loc%CHARSET);

DBMS_LOB.READ (
    file_loc   IN             BFILE,
    amount    IN OUT   NOCOPY INTEGER,
    offset    IN              INTEGER,
    buffer    OUT             RAW);

Parameters

Table 93-86  READ Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be read. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB to be examined.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to read, or number that were read.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).</td>
</tr>
<tr>
<td>buffer</td>
<td>Output buffer for the read operation.</td>
</tr>
</tbody>
</table>
Exceptions

Table 93-87 lists exceptions that apply to any LOB instance. Table 93-88 lists exceptions that apply only to BFILEs.

Table 93-87  READ Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of lob_loc, amount, or offset parameters are NULL.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; 32767 bytes (or the character equivalent)</td>
</tr>
<tr>
<td></td>
<td>- offset &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- offset &gt; LOBMAXSIZE</td>
</tr>
<tr>
<td></td>
<td>- amount is greater, in bytes or characters, than the capacity of buffer.</td>
</tr>
<tr>
<td>NO_DATA_FOUND</td>
<td>End of the LOB is reached, and there are no more bytes or characters to</td>
</tr>
<tr>
<td></td>
<td>read from the LOB: amount has a value of 0.</td>
</tr>
</tbody>
</table>

Table 93-88  READ Procedure Exceptions for BFILEs

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>File is not opened using the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB</td>
</tr>
</tbody>
</table>

Usage Notes

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.
- When calling DBMS_LOB.READ from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. The database converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.
- READ gets the LOB, if necessary, before the read.
- If the LOB is a DBFS LINK, data is streamed from DBFS, if possible, otherwise an exception is thrown.
93.9.42 SET_DBFS_LINK Procedures

This function links the specified SecureFile to the specified path name. It does not copy the data to the path.

Syntax

```sql
DBMS_LOB.SET_DBFS_LINK (
  lob_loc        IN OUT BLOB,
  archive_id     IN     RAW(1024));
```

```sql
DBMS_LOB.SET_DBFS_LINK(
  lob_loc_dst    IN OUT CLOB CHARACTER SET ANY_CS,
  archive_id     IN     RAW(1024));
```

Parameters

Table 93-89  SET_DBFS_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB for which to store the reference value</td>
</tr>
<tr>
<td>archive_id</td>
<td>Archive ID as returned by calling either of the</td>
</tr>
<tr>
<td></td>
<td>GET_DBFS_LINK Functions</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-90  SET_DBFS_LINK Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>lob_loc is not a SECUREFILE</td>
</tr>
</tbody>
</table>

93.9.43 SETCONTENTTYPE Procedure

This procedure sets the content type string for the data in the LOB.

Syntax

```sql
DBMS_LOB.SETCONTENTTYPE (  
  lob_loc      IN OUT NOCOPY BLOB,  
  contenttype  IN            VARCHAR2);  
```

```sql
DBMS_LOB.SETCONTENTTYPE (  
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,  
  contenttype IN            VARCHAR2);  
```
Parameters

Table 93-91  SETCONTENTTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>LOB to be assigned the content type</td>
</tr>
<tr>
<td>contenttype</td>
<td>String to be assigned</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-92  SETCONTENTTYPE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>lob_loc is not a SECUREFILE</td>
</tr>
</tbody>
</table>

Usage Notes

To clear an existing content type associated with a SECUREFILE, invoke SETCONTENTTYPE with contenttype set to empty string.

93.9.44 SETOPTIONS Procedures

This procedure enables/disables compression and deduplication on a per-LOB basis, overriding the default LOB column settings.

Syntax

```sql
DBMS_LOB.SETOPTIONS (  
  lob_loc   IN     BLOB,  
  option_types IN     PLS_INTEGER,  
  options   IN     PLS_INTEGER);
```

```sql
DBMS_LOB.SETOPTIONS (  
  lob_loc   IN     CLOB CHARACTER SET ANY_CS,  
  option_types IN     PLS_INTEGER,  
  options   IN     PLS_INTEGER);
```

Parameters

Table 93-93  SETOPTIONS Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be examined. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>option_type</td>
<td>See Table 93-2</td>
</tr>
<tr>
<td>options</td>
<td>See Table 93-3</td>
</tr>
</tbody>
</table>
Exceptions

Table 93-94  SETOPTIONS Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECUREFILE_BADLOB</td>
<td>Unsupported object type for the operation</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>A parameter value was invalid</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform operation during a query</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled</td>
</tr>
</tbody>
</table>

Usage Notes

- `DBMS_LOB.SETOPTIONS` cannot be used to enable or disable encryption on individual LOBs.
- You cannot turn the compression or deduplication features on or off for a SecureFile column if they were not turned when the table was created.
- The `GETOPTIONS Functions` and `SETOPTIONS Procedures` work on individual SecureFiles. You can turn off compression or deduplication on a particular SecureFiles LOB and turn on them on, if they have already been turned off by `SETOPTIONS`.
- This call incurs a round-trip to the server to make the changes persistent.

93.9.45 SUBSTR Functions

This function returns `amount` bytes or characters of a LOB, starting from an absolute offset from the beginning of the LOB.

For fixed-width n-byte CLOBs, if the input amount for `SUBSTR` is greater than \((32767/n)\), then `SUBSTR` returns a character buffer of length \((32767/n)\), or the length of the CLOB, whichever is lesser. For CLOBs in a varying-width character set, \(n\) is the maximum byte-width used for characters in the CLOB.

Syntax

```
DBMS_LOB.SUBSTR ( lob_loc     IN    BLOB,
                 amount      IN    INTEGER := 32767,
                 offset      IN    INTEGER := 1)
RETURN RAW;

DBMS_LOB.SUBSTR ( lob_loc     IN    CLOB   CHARACTER SET ANY_CS,
                 amount      IN    INTEGER := 32767,
                 offset      IN    INTEGER := 1)
RETURN VARCHAR2 CHARACTER SET lob_loc%CHARSET;

DBMS_LOB.SUBSTR ( file_loc     IN    BFILE,
                 amount      IN    INTEGER := 32767,
                 offset      IN    INTEGER := 1)
RETURN RAW;
```
Pragmas

```
pragma restrict_references(SUBSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

**Table 93-95  SUBSTR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the LOB to be read. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>file_loc</td>
<td>The file locator for the LOB to be examined.</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to be read.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).</td>
</tr>
</tbody>
</table>

Return Values

**Table 93-96  SUBSTR Function Return Values**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Function overloading that has a BLOB or BFILE in parameter.</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>CLOB version.</td>
</tr>
<tr>
<td>NULL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- any input parameter is NULL</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; 32767</td>
</tr>
<tr>
<td></td>
<td>- offset &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- offset &gt; LOBMAXSIZE</td>
</tr>
</tbody>
</table>

Exceptions

**Table 93-97  SUBSTR Function Exceptions for BFILE operations**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNOPENED_FILE</td>
<td>File is not opened using the input locator.</td>
</tr>
<tr>
<td>NOEXIST_DIRECTORY</td>
<td>Directory does not exist.</td>
</tr>
<tr>
<td>NOPRIV_DIRECTORY</td>
<td>You do not have privileges for the directory.</td>
</tr>
<tr>
<td>INVALID_DIRECTORY</td>
<td>Directory has been invalidated after the file was opened.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File does not exist, or you do not have access privileges on the file.</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB</td>
</tr>
</tbody>
</table>

Usage Notes

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR
data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

- When calling DBMS_LOB.SUBSTR from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client’s character set. The database converts the LOB value from the server’s character set to the client’s character set before it returns the buffer to the user.

- DBMS_LOB.SUBSTR will return 8191 or more characters based on the characters stored in the LOBs. If all characters are not returned as a consequence of the character byte size exceeding the available buffer, the user should either call DBMS_LOB.SUBSTR with a new offset to read the remaining characters, or call the subprogram on loop until all the data is extracted.

- SUBSTR gets the LOB, if necessary, before read.

- If the LOB is a DBFS Link, the data is streamed from DBFS, if possible, otherwise, an exception is thrown.

### See Also:

- "INSTR Functions"
- "READ Procedures"
- *Oracle Database SecureFiles and Large Objects Developer’s Guide* for additional details on usage of this procedure

### 93.9.46 TRIM Procedures

This procedure trims the value of the internal LOB to the length you specify in the newlen parameter.

Specify the length in bytes for BLOBs, and specify the length in characters for CLOBs.

#### Note:

The TRIM procedure decreases the length of the LOB to the value specified in the newlen parameter.

If you attempt to TRIM an empty LOB, then nothing occurs, and TRIM returns no error. If the new length that you specify in newlen is greater than the size of the LOB, then an exception is raised.

#### Syntax

```sql
DBMS_LOB.TRIM (    lob_loc IN OUT NOCOPY BLOB,    newlen IN INTEGER);
```

```sql
DBMS_LOB.TRIM (    lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,    newlen IN INTEGER);
```
Parameters

Table 93-98  TRIM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the internal LOB whose length is to be trimmed. For more information, see Operational Notes.</td>
</tr>
<tr>
<td>newlen</td>
<td>New, trimmed length of the LOB value in bytes for BLOBs or characters for CLOBs.</td>
</tr>
</tbody>
</table>

Exceptions

Table 93-99  TRIM Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>lob_loc is NULL.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- new_len &lt; 0</td>
</tr>
<tr>
<td></td>
<td>- new_len &gt; LOBMAXSIZE</td>
</tr>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform a LOB write inside a query or PDML parallel execution server</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB</td>
</tr>
</tbody>
</table>

Usage Notes

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.
- TRIM gets the LOB, if necessary, before altering the length of the LOB, unless the new length specified is '0'

See Also:

- "ERASE Procedures"
- "WRITEAPPEND Procedures"
- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure
93.9.47 WRITE Procedures

This procedure writes a specified amount of data into an internal LOB, starting from an absolute offset from the beginning of the LOB. The data is written from the buffer parameter.

WRITE replaces (overwrites) any data that already exists in the LOB at the offset, for the length you specify.

Syntax

```sql
DBMS_LOB.WRITE (
    lob_loc  IN OUT NOCOPY  BLOB,
    amount   IN             INTEGER,
    offset   IN             INTEGER,
    buffer   IN             RAW);

DBMS_LOB.WRITE (
    lob_loc  IN OUT  NOCOPY CLOB   CHARACTER SET ANY_CS,
    amount   IN             INTEGER,
    offset   IN             INTEGER,
    buffer   IN             VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_loc</td>
<td>Locator for the internal LOB to be written to. For more information, see Operational Notes</td>
</tr>
<tr>
<td>amount</td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to write</td>
</tr>
<tr>
<td>offset</td>
<td>Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1) for the write operation.</td>
</tr>
<tr>
<td>buffer</td>
<td>Input buffer for the write</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.</td>
</tr>
<tr>
<td>INVALID_ARGVAL</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- amount &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- amount &gt; 32767 bytes (or the character equivalent)</td>
</tr>
<tr>
<td></td>
<td>- offset &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- offset &gt; LOBMAXSIZE</td>
</tr>
</tbody>
</table>
Table 93-101  (Cont.) WRITE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_WRITE</td>
<td>Cannot perform a LOB write inside a query or PDML parallel execution server</td>
</tr>
<tr>
<td>BUFFERING_ENABLED</td>
<td>Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB</td>
</tr>
<tr>
<td>SECUREFILE_OUTOFBOUNDS</td>
<td>Attempted to perform a write operation past the end of a LOB having FRAGMENT_* on it</td>
</tr>
</tbody>
</table>

Usage Notes

- There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer is written to the LOB. If the offset you specify is beyond the end of the data currently in the LOB, then zero-byte fillers or spaces are inserted in the BLOB or CLOB respectively.

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

- When calling DBMS_LOB.WRITE from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client’s character set. The database converts the client-side buffer to the server’s character set before it writes the buffer data to the LOB.

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

- WRITE gets the LOB, if necessary, before writing the LOB, unless the write is specified to overwrite the entire LOB.

See Also:

- "APPEND Procedures"
- "COPY Procedures"
- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure
93.9.48 WRITEAPPEND Procedures

This procedure writes a specified amount of data to the end of an internal LOB. The data is written from the `buffer` parameter.

Syntax

```
DBMS_LOB.WRITEAPPEND (  
    lob_loc IN OUT NOCOPY BLOB,  
    amount IN INTEGER,  
    buffer IN RAW);
```

```
DBMS_LOB.WRITEAPPEND (  
    lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,  
    amount IN INTEGER,  
    buffer IN VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 93-102  WRITEAPPEND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lob_loc</code></td>
<td>Locator for the internal LOB to be written to. For more information, see Operational Notes</td>
</tr>
<tr>
<td><code>amount</code></td>
<td>Number of bytes (for BLOBs) or characters (for CLOBs) to write</td>
</tr>
<tr>
<td><code>buffer</code></td>
<td>Input buffer for the write</td>
</tr>
</tbody>
</table>

Usage Notes

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer are written to the end of the LOB.

Exceptions

Table 93-103  WRITEAPPEND Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Any of <code>lob_loc</code>, <code>amount</code>, or <code>offset</code> parameters are NULL, out of range, or INVALID.</td>
</tr>
</tbody>
</table>
| INVALID_ARGVAL          | Either:  
  - `amount < 1`  
  - `amount > 32767` bytes (or the character equivalent) |
| QUERY_WRITE             | Cannot perform a LOB write inside a query or PDML parallel execution server |
| BUFFERING_ENABLED       | Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB |
Usage Notes

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

- When calling DBMS_LOB.WRITEAPPEND from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set. The database converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

- WRITEAPPEND gets the LOB, if necessary, before appending to the LOB.

See Also:

- "APPEND Procedures"
- "COPY Procedures"
- "WRITE Procedures"
- Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure
The **DBMS_LOCK** package provides an interface to Oracle Lock Management services.

You can request a lock of a specific mode, give it a unique name recognizable in another procedure in the same or another instance, change the lock mode, and release it.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Rules and Limits
- Operational Notes
- Summary of DBMS_LOCK Subprograms

See Also:

For more information, and an example of how to use the **DBMS_LOCK** package, see the *Oracle Database Development Guide.*

### 94.1 DBMS_LOCK Overview

The **DBMS_LOCK** package has many beneficial uses.

These uses include the following:

- Providing exclusive access to a device, such as a terminal
- Providing application-level enforcement of read locks
- Detecting when a lock is released and cleanup after the application
- Synchronizing applications and enforcing sequential processing

### 94.2 DBMS_LOCK Security Model

There might be operating system-specific limits on the maximum number of total locks available. This must be considered when using locks or making this package available to other users. Consider granting the **EXECUTE** privilege only to specific users or roles.

A better alternative would be to create a cover package limiting the number of locks used and grant **EXECUTE** privilege to specific users. An example of a cover package is documented in the **DBMS_LOCK.SQL** package specification file. The abbreviations for these locks as they appear in Enterprise Manager monitors are in parentheses.
94.3 DBMS_LOCK Constants

The DBMS_LOCK package includes several constants to use when specifying parameter values.

These constants are shown in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Alternate Name(s)</th>
<th>Type</th>
<th>Value</th>
<th>OEM Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL_MODE</td>
<td>NuL1</td>
<td>INTEGER</td>
<td>1</td>
<td>-</td>
<td>This can be used on an aggregate object to indicate that share locks are being acquired on subparts of the object.</td>
</tr>
<tr>
<td>SS_MODE</td>
<td>Sub Shared</td>
<td>INTEGER</td>
<td>2</td>
<td>ULRS</td>
<td>This can be used on an aggregate object to indicate that exclusive locks are being acquired on sub-parts of the object.</td>
</tr>
<tr>
<td>SX_MODE</td>
<td>Sub eXclusive</td>
<td>INTEGER</td>
<td>3</td>
<td>ULRX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Row Exclusive Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S_MODE</td>
<td>Shared</td>
<td>INTEGER</td>
<td>4</td>
<td>ULRSX</td>
<td>This indicates that the entire aggregate object has a share lock, but some of the sub-parts may additionally have exclusive locks.</td>
</tr>
<tr>
<td></td>
<td>Row Exclusive Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intended Exclusive</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSX_MODE</td>
<td>Shared Sub eXclusive</td>
<td>INTEGER</td>
<td>5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Share Row Exclusive Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X_MODE</td>
<td>Exclusive</td>
<td>INTEGER</td>
<td>6</td>
<td>ULX</td>
<td></td>
</tr>
</tbody>
</table>

These are the various lock modes (nl -> "NuL1", ss -> "Sub Shared", sx -> "Sub eXclusive", s -> "Shared", ssx -> "Shared Sub eXclusive", x -> "eXclusive").

94.4 DBMS_LOCK. Rules and Limits

When another process holds "held", an attempt to get "get" succeeds or fails, based on the held mode and type of get.

The following table describes the results:

<table>
<thead>
<tr>
<th>HELD MODE</th>
<th>GET NL</th>
<th>GET SS</th>
<th>GET SX</th>
<th>GET S</th>
<th>GET SSX</th>
<th>GET X</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
</tr>
<tr>
<td>SS</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
</tr>
</tbody>
</table>
Table 94-2    (Cont.) Lock Compatibility

<table>
<thead>
<tr>
<th>HELD MODE</th>
<th>GET NL</th>
<th>GET SS</th>
<th>GET SX</th>
<th>GET S</th>
<th>GET SSX</th>
<th>GET X</th>
</tr>
</thead>
<tbody>
<tr>
<td>SX</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>S</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>SSX</td>
<td>Success</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
<tr>
<td>X</td>
<td>Success</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
<td>Fail</td>
</tr>
</tbody>
</table>

maxwait constant integer := 32767;

The constant maxwait waits forever.

94.5 DBMS_LOCK Operational Notes

User locks never conflict with Oracle locks because they are identified with the prefix "UL". You can view these locks using the Enterprise Manager lock monitor screen or the appropriate fixed views.

User locks are automatically released when a session terminates. The lock identifier is a number in the range of 0 to 1073741823.

Because a reserved user lock is the same as an Oracle lock, it has all the functionality of an Oracle lock, such as deadlock detection. Be certain that any user locks used in distributed transactions are released upon COMMIT, or an undetected deadlock may occur.

DBMS_LOCK is most efficient with a limit of a few hundred locks for each session. Oracle strongly recommends that you develop a standard convention for using these locks in order to avoid conflicts among procedures trying to use the same locks. For example, include your company name as part of your lock names.

94.6 Summary of DBMS_LOCK Subprograms

This table lists the DBMS_LOCK subprograms and briefly describes them.

Table 94-3    DBMS_LOCK Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOCATE_UNIQUE Procedure</td>
<td>Allocates a unique lock ID to a named lock</td>
</tr>
<tr>
<td>CONVERT Function</td>
<td>Converts a lock from one mode to another</td>
</tr>
<tr>
<td>RELEASE Function</td>
<td>Releases a lock</td>
</tr>
<tr>
<td>REQUEST Function</td>
<td>Requests a lock of a specific mode.</td>
</tr>
</tbody>
</table>

94.6.1 ALLOCATE_UNIQUE Procedure

This procedure allocates a unique lock identifier (in the range of 1073741824 to 1999999999) a specified lock name. Lock identifiers are used to enable applications to coordinate their use.
of locks. This is provided because it may be easier for applications to coordinate their use of locks based on lock names rather than lock numbers.

**Syntax**

```sql
DBMS_LOCK.ALLOCATE_UNIQUE (  
    lockname IN VARCHAR2,  
    lockhandle OUT VARCHAR2,  
    expiration_secs IN INTEGER DEFAULT 864000);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockname</td>
<td>Name of the lock for which you want to generate a unique ID. Do not use lock names beginning with <code>ORA$</code>; these are reserved for products supplied by Oracle.</td>
</tr>
<tr>
<td>lockhandle</td>
<td>Returns the handle to the lock ID generated by <code>ALLOCATE_UNIQUE</code>. You can use this handle in subsequent calls to <code>REQUEST</code>, <code>CONVERT</code>, and <code>RELEASE</code>.</td>
</tr>
<tr>
<td>expiration_secs</td>
<td>Number of seconds to wait after the last <code>ALLOCATE_UNIQUE</code> has been performed on a specified lock, before permitting that lock to be deleted from the <code>DBMS_LOCK_ALLOCATED</code> table. The default waiting period is 10 days. You should not delete locks from this table. Subsequent calls to <code>ALLOCATE_UNIQUE</code> may delete expired locks to recover space.</td>
</tr>
</tbody>
</table>

**Usage Notes**

If you choose to identify locks by name, you can use `ALLOCATE_UNIQUE` to generate a unique lock identification number for these named locks.

The first session to call `ALLOCATE_UNIQUE` with a new lock name causes a unique lock ID to be generated and stored in the `dbms_lock_allocated` table. Subsequent calls (usually by other sessions) return the lock ID previously generated.

A lock name is associated with the returned lock ID for at least `expiration_secs` (defaults to 10 days) past the last call to `ALLOCATE_UNIQUE` with the specified lock name. After this time, the row in the `dbms_lock_allocated` table for this lock name may be deleted in order to recover space. `ALLOCATE_UNIQUE` performs a commit.
WARNING:

Named user locks may be less efficient, because Oracle uses SQL to determine the lock associated with a specified name.

Exceptions

ORA-20000, ORU-10003: Unable to find or insert lock <lockname> into catalog dbms_lock_allocated.

94.6.2 ALLOCATE_UNIQUE_AUTONOMOUS Procedure

This procedure allocates a unique lock identifier (in the range of 1073741824 to 1999999999) a specified lock name and is an autonomous version of the ALLOCATE_UNIQUE procedure. This procedure works exactly same as that of ALLOCATE_UNIQUE, except that the procedure will run as an autonomous transaction. Therefore the commits in ALLOCATE_UNIQUE_AUTONOMOUS procedure will not affect the calling procedure. The ALLOCATE_UNIQUE_AUTONOMOUS procedure is implemented in DB 12.1 and later releases.

Syntax

```
DBMS_LOCK.ALLOCATE_UNIQUE_AUTONOMOUS (
   lockname         IN  VARCHAR2,
   lockhandle       OUT VARCHAR2,
   expiration_secs  IN  INTEGER   DEFAULT 864000);
```

Parameters

Table 94-5  ALLOCATE_UNIQUE_AUTONOMOUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockname</td>
<td>Name of the lock for which you want to generate a unique ID. Do not use lock names beginning with ORA$; these are reserved for products supplied by Oracle.</td>
</tr>
<tr>
<td>lockhandle</td>
<td>Returns the handle to the lock ID generated by ALLOCATE_UNIQUE_AUTONOMOUS. You can use this handle in subsequent calls to REQUEST, CONVERT, and RELEASE. A handle is returned instead of the actual lock ID to reduce the chance that a programming error accidentally creates an incorrect, but valid, lock ID. This provides better isolation between different applications that are using this package. LOCKHANDLE can be up to VARCHAR2 (128). All sessions using a lock handle returned by ALLOCATE_UNIQUE_AUTONOMOUS with the same lock name are referring to the same lock. Therefore, do not pass lock handles from one session to another.</td>
</tr>
</tbody>
</table>
Table 94-5  (Cont.) ALLOCATE_UNIQUE_AUTONOMOUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expiration_secs</td>
<td>Number of seconds to wait after the last ALLOCATE_UNIQUE_AUTONOMOUS has been performed on a specified lock, before permitting that lock to be deleted from the ALLOCATE_UNIQUE_AUTONOMOUS table. The default waiting period is 10 days. You should not delete locks from this table. Subsequent calls to ALLOCATE_UNIQUE_AUTONOMOUS may delete expired locks to recover space.</td>
</tr>
</tbody>
</table>

Usage Notes

If you choose to identify locks by name, you can use ALLOCATE_UNIQUE_AUTONOMOUS to generate a unique lock identification number for these named locks.

The first session to call ALLOCATE_UNIQUE_AUTONOMOUS with a new lock name causes a unique lock ID to be generated and stored in the dbms_lock_allocated table. Subsequent calls (usually by other sessions) return the lock ID previously generated.

A lock name is associated with the returned lock ID for at least expiration_secs (defaults to 10 days) past the last call to ALLOCATE_UNIQUE_AUTONOMOUS with the specified lock name. After this time, the row in the dbms_lock_allocated table for this lock name may be deleted in order to recover space. ALLOCATE_UNIQUE_AUTONOMOUS performs a commit.

⚠️ WARNING: ⚠️

Named user locks may be less efficient, because Oracle uses SQL to determine the lock associated with a specified name.

Exceptions

ORA-20000, ORU-10003: Unable to find or insert lock <lockname> into catalog dbms_lock_allocated.

94.6.3 CONVERT Function

This function converts a lock from one mode to another. CONVERT is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```sql
DBMS_LOCK.CONVERT(
  id         IN INTEGER ||
  lockhandle IN VARCHAR2,
  lockmode   IN INTEGER,
  timeout    IN NUMBER DEFAULT MAXWAIT)
RETURN INTEGER;
```
Parameters

Table 94-6 CONVERT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id or lockhandle</td>
<td>User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change</td>
</tr>
<tr>
<td>lockmode</td>
<td>New mode that you want to assign to the specified lock. For the available modes and their associated integer identifiers, see Constants.</td>
</tr>
<tr>
<td>timeout</td>
<td>Number of seconds to continue trying to change the lock mode. If the lock cannot be converted within this time period, then the call returns a value of 1 (timeout).</td>
</tr>
</tbody>
</table>

Return Values

Table 94-7 CONVERT Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>Timeout</td>
</tr>
<tr>
<td>2</td>
<td>Deadlock</td>
</tr>
<tr>
<td>3</td>
<td>Parameter error</td>
</tr>
<tr>
<td>4</td>
<td>Don't own lock specified by id or lockhandle</td>
</tr>
<tr>
<td>5</td>
<td>Illegal lock handle</td>
</tr>
</tbody>
</table>

94.6.4 RELEASE Function

This function explicitly releases a lock previously acquired using the REQUEST function.

Locks are automatically released at the end of a session. RELEASE is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```sql
DBMS_LOCK.RELEASE (id IN INTEGER)
RETURN INTEGER;

DBMS_LOCK.RELEASE (lockhandle IN VARCHAR2)
RETURN INTEGER;
```
Parameters

Table 94-8  RELEASE Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id or lockhandle</td>
<td>User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change</td>
</tr>
</tbody>
</table>

Return Values

Table 94-9  RELEASE Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>3</td>
<td>Parameter error</td>
</tr>
<tr>
<td>4</td>
<td>Do not own lock specified by id or lockhandle</td>
</tr>
<tr>
<td>5</td>
<td>Illegal lock handle</td>
</tr>
</tbody>
</table>

94.6.5 REQUEST Function

This function requests a lock with a specified mode.

REQUEST is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```sql
DBMS_LOCK.REQUEST(
    id                 IN  INTEGER ||
    lockhandle         IN  VARCHAR2,
    lockmode           IN  INTEGER DEFAULT X_MODE,
    timeout            IN  INTEGER DEFAULT MAXWAIT,
    release_on_commit  IN  BOOLEAN DEFAULT FALSE)
RETURN INTEGER;
```

The current default values, such as X_MODE and MAXWAIT, are defined in the DBMS_LOCK package specification.

Parameters

Table 94-10  REQUEST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id or lockhandle</td>
<td>User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change</td>
</tr>
<tr>
<td>lockmode</td>
<td>Mode that you are requesting for the lock. For the available modes and their associated integer identifiers, see Constants.</td>
</tr>
</tbody>
</table>
### Table 94-10  (Cont.) REQUEST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Number of seconds to continue trying to grant the lock. If the lock cannot</td>
</tr>
<tr>
<td></td>
<td>be granted within this time period, then the call returns a value of 1 (timeout).</td>
</tr>
<tr>
<td>release_on_commit</td>
<td>Set this parameter to TRUE to release the lock on commit or rollback.</td>
</tr>
<tr>
<td></td>
<td>Otherwise, the lock is held until it is explicitly released or until the</td>
</tr>
<tr>
<td></td>
<td>end of the session.</td>
</tr>
</tbody>
</table>

### Return Values

#### Table 94-11  REQUEST Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>Timeout</td>
</tr>
<tr>
<td>2</td>
<td>Deadlock</td>
</tr>
<tr>
<td>3</td>
<td>Parameter error</td>
</tr>
<tr>
<td>4</td>
<td>Already own lock specified by id or lockhandle</td>
</tr>
<tr>
<td>5</td>
<td>Illegal lock handle</td>
</tr>
</tbody>
</table>
95

DBMS_LOGMNR

The DBMS_LOGMNR package, one of a set of LogMiner packages, contains the subprograms you use to initialize the LogMiner tool and to begin and end a LogMiner session.

This chapter contains the following topics:

• Overview
• Security Model
• Constants
• Views
• Operational Notes
• Summary of DBMS_LOGMNR Subprograms

See Also:
Oracle Database Utilities for information regarding LogMiner.

95.1 DBMS_LOGMNR Overview

Oracle LogMiner, which is part of Oracle Database, enables you to query online and archived redo log files through a SQL interface. The DBMS_LOGMNR package provides the majority of the tools needed to start and stop LogMiner and specify the redo log files of interest.

All changes made to user data or to the database dictionary are recorded in the Oracle redo log files so that database recovery operations can be performed. You can take advantage of the data recorded in the redo log files to accomplish other tasks, such as:

• Pinpointing when a logical corruption to a database, such as errors made at the application level, may have begun
• Determining what actions you would have to take to perform fine-grained recovery at the transaction level.
• Performance tuning and capacity planning through trend analysis.
• Track any data manipulation language (DML) and data definition language (DDL) statements executed on the database, the order in which they were executed, and who executed them.

See Also:
DBMS_LOGMNR_D for information on the package subprograms that extract a LogMiner dictionary and re-create LogMiner tables in alternate tablespaces.
95.2 DBMS_LOGMNR Security Model

You must have the `EXECUTE_CATALOG_ROLE` role to use the `DBMS_LOGMNR` package.

95.3 DBMS_LOGMNR Constants

The `DBMS_LOGMNR` package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, `DBMS_LOGMNR.NEW`.

The following table describes the constants for the `ADD_LOGFILE` options flag in the `DBMS_LOGMNR` package.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>Implicitly calls the <code>DBMS_LOGMNR.END_LOGMNR</code> procedure to end the current LogMiner session and then creates a new session. The new session starts a new list of redo log files to be analyzed, beginning with the redo log file you specify.</td>
</tr>
<tr>
<td>ADDFILE</td>
<td>Adds the specified redo log file to the list of redo log files to be analyzed. Any attempt to add a duplicate file raises an exception (ORA-01289). This is the default if no options flag is specified.</td>
</tr>
</tbody>
</table>

Table 95-2 describes the constants for the `START_LOGMNR` options flag in the `DBMS_LOGMNR` package.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMITTED_DATA_ONLY</td>
<td>If set, DML statements corresponding to committed transactions are returned. DML statements corresponding to a committed transaction are grouped together. Transactions are returned in their commit order. Transactions that are rolled back or in-progress are filtered out, as are internal redo records (those related to index operations, management, and so on). If this option is not set, all rows for all transactions (committed, rolled back, and in-progress) are returned in the order in which they are found in the redo logs (in order of SCN values).</td>
</tr>
<tr>
<td>SKIP_CORRUPTION</td>
<td>Directs a select operation on the $LOGMNR_CONTENTS view to skip any corruptions in the redo log file being analyzed and continue processing. This option works only when a block in the redo log file (and not the header of the redo log file) is corrupt. You should check the INFO column in the $LOGMNR_CONTENTS view to determine the corrupt blocks skipped by LogMiner. When a corruption in the redo log file is skipped, the OPERATION column contains the value CORRUPTED_BLOCKS, and the STATUS column contains the value 1343.</td>
</tr>
</tbody>
</table>
### Table 95-2 (Cont.) Constants for START_LOGMNR Options Flag

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDL_DICT_TRACKING</td>
<td>If the LogMiner dictionary in use is a flat file or in the redo log files, LogMiner updates its internal dictionary if a DDL event occurs. This ensures that correct SQL_REDO and SQL_UNDO information is maintained for objects that are modified after the LogMiner internal dictionary is built. The database to which LogMiner is connected must be open. This option cannot be used in conjunction with the DICT_FROM_ONLINE_CATALOG option and cannot be used when the LogMiner dictionary being used is one that was extracted to a flat file prior to Oracle9i.</td>
</tr>
<tr>
<td>DICT_FROM_ONLINE_CATALOG</td>
<td>Directs LogMiner to use the current online database dictionary rather than a LogMiner dictionary contained in a flat file or in the redo log files being analyzed. This option cannot be used in conjunction with the DDL_DICT_TRACKING option. The database to which LogMiner is connected must be the same one that generated the redo log files. Expect to see a value of 2 in the STATUS column of the V$LOGMNR_CONTENTS view if the table definition in the database does not match the table definition in the redo log file.</td>
</tr>
<tr>
<td>DICT_FROM_REDO_LOGS</td>
<td>If set, LogMiner expects to find a LogMiner dictionary in the redo log files that were specified. The redo log files are specified with the DBMS_LOGMNR.ADD_LOGFILE procedure or with the DBMS_LOGMNR.START_LOGMNR procedure with the CONTINUOUS_MINE option.</td>
</tr>
<tr>
<td>NO_SQL_DELIMITER</td>
<td>If set, the SQL delimiter (a semicolon) is not placed at the end of reconstructed SQL statements. This is helpful for applications that open a cursor and then execute the reconstructed statements.</td>
</tr>
<tr>
<td>NO_ROWID_IN_STMT</td>
<td>If set, the ROWID clause is not included in the reconstructed SQL statements. The redo log file may already contain logically unique identifiers for modified rows if supplemental logging is enabled. When using this option, you must be sure that supplemental logging was enabled in the source database at the appropriate level and that no duplicate rows exist in the tables of interest. LogMiner does not make any guarantee regarding the uniqueness of logical row identifiers.</td>
</tr>
<tr>
<td>PRINTPRETTY_SQL</td>
<td>If set, LogMiner formats the reconstructed SQL statements for ease of reading. These reconstructed SQL statements are not executable.</td>
</tr>
</tbody>
</table>
Table 95-2  (Cont.) Constants for START_LOGMNR Options Flag

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTINUOUS_MINE</td>
<td>Directs LogMiner to automatically add redo log files, as needed, to find the data of interest. You only need to specify the first log to start mining, or just the starting SCN or date to indicate to LogMiner where to begin mining logs. You are not required to specify any redo log files explicitly. LogMiner automatically adds and mines the (archived and online) redo log files for the data of interest. This option requires that LogMiner is connected to the same database instance that is generating the redo log files. It also requires that the database be mounted and that archiving be enabled.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> As of Oracle Database 12c Release 2 (12.2.0.1), the LogMiner CONTINUOUS_MINE option is deprecated. It is still supported for backward compatibility reasons, but Oracle recommends that you discontinue using it. There is no replacement functionality.</td>
</tr>
</tbody>
</table>

STRING_LITERALS_IN_STMT | If set, SQL_REDO and SQL_UNDO use literals for numbers and datetime and interval column types.  
To specify more than one option, use a plus sign (+) between them. For example:  
EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => DBMS_LOGMNR.DDL_DICT_TRACKING + DBMS_LOGMNR.DICT_FROM_REDO_LOGS);  

95.4 DBMS_LOGMNR Views

The DBMS_LOGMNR package uses the views listed under Accessing Logminer Operational Information In Views in Oracle Database Utilities.

See Also:

Oracle Database Utilities

95.5 DBMS_LOGMNR Operational Notes

A LogMiner session begins with a call to DBMS_LOGMNR.ADD_LOGFILE or DBMS_LOGMNR.START_LOGMNR (the former if you plan to specify log files explicitly; the
latter if you plan to use continuous mining). The session ends with a call to
\texttt{DBMS\_LOGMNR\_END\_LOGMNR}.

Within a LogMiner session, you can specify the redo log files to be analyzed and the SCN or
time range of interest; then you can issue SQL \texttt{SELECT} statements against the
\texttt{V\_LOGMNR\_CONTENTS} view to retrieve the data of interest.

\section*{95.6 Summary of DBMS\_LOGMNR Subprograms}

This table lists the \texttt{DBMS\_LOGMNR} subprograms and briefly describes them.

In a multitenant container database (CDB) some subprograms must be called from the root.
There may be other differences as well. See the individual subprogram descriptions for
details.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Subprogram & Description \\
\hline
\texttt{ADD\_LOGFILE Procedure} & Adds a redo log file to the existing or newly created list of redo log files for LogMiner to process, so that if a new list is created, this marks the beginning of a LogMiner session \\
\texttt{COLUMN\_PRESENT Function} & Call this function for any row returned from the \texttt{V\_LOGMNR\_CONTENTS} view to determine if undo or redo column values exist for the column specified by the \texttt{column\_name} input parameter to this function \\
\texttt{END\_LOGMNR Procedure} & Finishes a LogMiner session \\
\texttt{MINE\_VALUE Function} & Call this function for any row returned from the \texttt{V\_LOGMNR\_CONTENTS} view to retrieve the undo or redo column value of the column specified by the \texttt{column\_name} input parameter to this function \\
\texttt{REMOVE\_LOGFILE Procedure} & Removes a redo log file from the list of redo log files for LogMiner to process \\
\texttt{START\_LOGMNR Procedure} & Initializes the LogMiner utility and starts LogMiner (unless the session was already started with a call to \texttt{DBMS\_LOGMNR\_ADD\_LOGFILE}) \\
\hline
\end{tabular}
\end{table}

\subsection*{95.6.1 ADD\_LOGFILE Procedure}

This procedure adds a file to an existing or newly created list of log files for LogMiner to process.

In a CDB, the \texttt{ADD\_LOGFILE} procedure must be called from the root database. You must have the \texttt{LOGMINING} administrative privilege to use this procedure.

\textbf{Syntax}

\begin{verbatim}
DBMS\_LOGMNR\_ADD\_LOGFILE(
    LogFileName  IN VARCHAR2,
    options      IN BINARY\_INTEGER default ADDFILE );
\end{verbatim}
### Parameters

**Table 95-4 ADD_LOGFILE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogFileName</td>
<td>Specifies the name of the redo log file to add to the list of redo log files to be analyzed during this session.</td>
</tr>
</tbody>
</table>

**options**

Does one of the following:

- Starts a new LogMiner session and a new list of redo log files for analysis ([DBMS_LOGMNR.NEW](#)).
- Adds a file to an existing list of redo log files for analysis ([DBMS_LOGMNR.ADDFILE](#)).

See [Table 95-1](#).

### Exceptions

**Table 95-5 ADD_LOGFILE Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01284</td>
<td>Specified file cannot be opened.</td>
</tr>
<tr>
<td>ORA-01287</td>
<td>Specified file is from a different database incarnation.</td>
</tr>
<tr>
<td>ORA-01289</td>
<td>Specified file has already been added to the list. Duplicate redo log files cannot be added.</td>
</tr>
<tr>
<td>ORA-01290</td>
<td>Specified file is not in the current list and therefore cannot be removed from the list.</td>
</tr>
<tr>
<td>ORA-01324</td>
<td>Specified file cannot be added to the list because there is a DB_ID mismatch.</td>
</tr>
</tbody>
</table>

### Usage Notes

- Before querying the `V$LOGMNR_CONTENTS` view, you must make a successful call to the [DBMS_LOGMNR.START_LOGMNR](#) procedure (within the current LogMiner session).

- Unless you specify the [CONTINUOUS_MINE](#) option, the LogMiner session must be set up with a list of redo log files to be analyzed. Use the `ADD_LOGFILE` procedure to specify the list of redo log files to analyze.

- If you are not using the [CONTINUOUS_MINE](#) option and you want to analyze more than one redo log file, you must call the `ADD_LOGFILE` procedure separately for each redo log file. The redo log files do not need to be registered in any particular order.

- Both archived and online redo log files can be mined.

- After you have added the first redo log file to the list, each additional redo log file that you add to the list must be associated with the same database and database [RESETLOGS SCN](#) as the first redo log file. (The database [RESETLOGS](#) uniquely identifies each execution of an `ALTER DATABASE OPEN RESETLOGS` statement. When the online redo logs are reset, Oracle creates a new and unique incarnation of the database.)
To analyze the redo log files from a different database (or a database incarnation with a different database `RESETLOGS SCN`) than that with which the current list of redo log files is associated, use the `END_LOGMNR` procedure to end the current LogMiner session, and then build a new list using the `ADD_LOGFILE` procedure.

LogMiner matches redo log files by the log sequence number. Thus, two redo log files with different names but with the same log sequence number will return the ORA-01289 exception. For instance, the online counterpart of an archived redo log file has a different name from the archived redo log file, but attempting to register it with LogMiner after registering the archived counterpart will result in the ORA-01289 exception being returned.

95.6.2 COLUMN_PRESENT Function

This function is designed to be used in conjunction with the `MINE_VALUE` function.

If the `MINE_VALUE` function returns a `NULL` value, it can mean either:

- The specified column is not present in the redo or undo portion of the data.
- The specified column is present and has a `NULL` value.

To distinguish between these two cases, use the `COLUMN_PRESENT` function, which returns a `1` if the column is present in the redo or undo portion of the data. Otherwise, it returns a `0`.

Syntax

```sql
DBMS_LOGMNR.COLUMN_PRESENT (  
    sql_redo_undo IN RAW,  
    column_name  IN VARCHAR2 default '') RETURN NUMBER;
```

Parameters

**Table 95-6 COLUMN_PRESENT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sql_redo_undo</code></td>
<td>Specifies either the <code>REDO_VALUE</code> or the <code>UNDO_VALUE</code> column in the <code>V$LOGMNR_CONTENTS</code> view from which to extract data values. See the Usage Notes for more information.</td>
</tr>
<tr>
<td><code>column_name</code></td>
<td>Specifies the fully qualified name (<code>schema.table.column</code>) of the column for which this function will return information. In a CDB, the column name is specified as follows: <code>container_name:schema.table.column</code></td>
</tr>
</tbody>
</table>

Return Values

Table 95-7 describes the return values for the `COLUMN_PRESENT` function. The `COLUMN_PRESENT` function returns `1` if the self-describing record (the first parameter) contains the column specified in the second parameter. This can be used to determine the meaning of `NULL` values returned by the `DBMS_LOGMNR.MINE_VALUE` function.
Table 95-7 Return Values for COLUMN_PRESENT Function

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Specified column is not present in this row of V$LOGMNR_CONTENTS.</td>
</tr>
<tr>
<td>1</td>
<td>Column is present in this row of V$LOGMNR_CONTENTS.</td>
</tr>
</tbody>
</table>

Exceptions

Table 95-8 COLUMN_PRESENT Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01323</td>
<td>Currently, a LogMiner dictionary is not associated with the LogMiner session. You must specify a LogMiner dictionary for the LogMiner session.</td>
</tr>
<tr>
<td>ORA-00904</td>
<td>Value specified for the column_name parameter is not a fully qualified column name.</td>
</tr>
</tbody>
</table>

Usage Notes

- To use the COLUMN_PRESENT function, you must have successfully started LogMiner.
- The COLUMN_PRESENT function must be invoked in the context of a select operation on the V$LOGMNR_CONTENTS view.
- The COLUMN_PRESENT function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- The value for the sql_redo_undo parameter depends on the operation performed and the data of interest:
  - If an update operation was performed and you want to know what the value was prior to the update operation, specify UNDO_VALUE.
  - If an update operation was performed and you want to know what the value is after the update operation, specify REDO_VALUE.
  - If an insert operation was performed, typically you would specify REDO_VALUE (because the value of a column prior to an insert operation will always be NULL).
  - If a delete operation was performed, typically you would specify UNDO_VALUE (because the value of a column after a delete operation will always be NULL).

95.6.3 END_LOGMNR Procedure

This procedure finishes a LogMiner session. Because this procedure performs cleanup operations that may not otherwise be done, you must use it to properly end a LogMiner session. This procedure is called automatically when you log out of a
database session or when you call DBMS_LOGMNR.ADD_LOGFILE and specify the NEW option.

Syntax

DBMS_LOGMNR.END_LOGMNR;

Exceptions

Table 95-9  END_LOGMNR Procedure Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01307</td>
<td>No LogMiner session is currently active. The END_LOGMNR procedure was called without adding any log files or before the START_LOGMNR procedure was called</td>
</tr>
</tbody>
</table>

95.6.4 MINE_VALUE Function

This function facilitates queries based on a column's data value.

This function takes two arguments. The first one specifies whether to mine the redo (REDO_VALUE) or undo (UNDO_VALUE) portion of the data. The second argument is a string that specifies the fully qualified name of the column to be mined. The MINE_VALUE function always returns a string that can be converted back to the original datatype.

Syntax

DBMS_LOGMNR.MINE_VALUE (sql_redo_undo IN RAW, column_name IN VARCHAR2 default '') RETURN VARCHAR2;

Parameters

Table 95-10  MINE_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_redo_undo</td>
<td>Specifies either the REDO_VALUE or the UNDO_VALUE column in the V$LOGMNR_CONTENTS view from which to extract data values. See the Usage Notes for more information.</td>
</tr>
<tr>
<td>column_name</td>
<td>Specifies the fully qualified name (schema.table.column) of the column for which this function will return information. In a CDB, the column name is specified as follows: container_name:schema.table.column</td>
</tr>
</tbody>
</table>

Return Values

Table 95-11  Return Values for MINE_VALUE Function

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>The column is not contained within the self-describing record, or the column value is NULL. To distinguish between the two different null possibilities, use the DBMS_LOGMNR.COLUMN_PRESENT function.</td>
</tr>
</tbody>
</table>
Table 95-11  (Cont.) Return Values for MINE_VALUE Function

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON-NULL</td>
<td>The column is contained within the self-describing record; the value is returned in string format.</td>
</tr>
</tbody>
</table>

Exceptions

Table 95-12  MINE_VALUE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01323</td>
<td>Invalid state. Currently, a LogMiner dictionary is not associated with the LogMiner session. You must specify a LogMiner dictionary for the LogMiner session.</td>
</tr>
<tr>
<td>ORA-00904</td>
<td>Invalid identifier. The value specified for the column_name parameter was not a fully qualified column name.</td>
</tr>
</tbody>
</table>

Usage Notes

- To use the MINE_VALUE function, you must have successfully started LogMiner.
- The MINE_VALUE function must be invoked in the context of a select operation from the V$LOGMNR_CONTENTS view.
- The MINE_VALUE function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- The value for the sql_redo_undo parameter depends on the operation performed and the data of interest:
  - If an update operation was performed and you want to know what the value was prior to the update operation, specify UNDO_VALUE.
  - If an update operation was performed and you want to know what the value is after the update operation, specify REDO_VALUE.
  - If an insert operation was performed, typically you would specify REDO_VALUE (because the value of a column prior to an insert operation will always be null).
  - If a delete operation was performed, typically you would specify UNDO_VALUE (because the value of a column after a delete operation will always be null).
- If the DBMS_LOGMNR.MINE_VALUE function is used to get an NCHAR value that includes characters not found in the database character set, then those characters are returned as the replacement character (for example, an inverted question mark) of the database character set.

95.6.5 REMOVE_LOGFILE Procedure

This procedure removes a redo log file from an existing list of redo log files for LogMiner to process.

In a CDB, the REMOVE_LOGFILE procedure must be called from the root database. You must have the LOGMINING administrative privilege to use this procedure.
Syntax

DBMS_LOGMNR.REMOVE_LOGFILE (  
    LogFileName IN VARCHAR2);

Parameters

Table 95-13  REMOVE_LOGFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogFileName</td>
<td>Specifies the name of the redo log file to be removed from the list of redo log files to be analyzed during this session.</td>
</tr>
</tbody>
</table>

Exceptions

Table 95-14  REMOVE_LOGFILE Procedure Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01290</td>
<td>Cannot remove unlisted log file</td>
</tr>
</tbody>
</table>

Usage Notes

- Before querying the V$LOGMNR_CONTENTS view, you must make a successful call to the DBMS_LOGMNR.START_LOGMNR procedure (within the current LogMiner session).
- You can use this procedure to remove a redo log file from the list of redo log files for LogMiner to process if you know that redo log file does not contain any data of interest.
- Multiple redo log files can be removed by calling this procedure repeatedly.
- The redo log files do not need to be removed in any particular order.
- To start a new list of redo log files for analysis, use the END_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD_LOGFILE procedure.
- Even if you remove all redo log files from the list, any subsequent calls you make to the ADD_LOGFILE procedure must match the database ID and RESETLOGS SCN of the removed redo log files. Therefore, to analyze the redo log files from a different database (or a database incarnation with a different database RESETLOGS SCN) than that with which the current list of redo log files is associated, use the END_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD_LOGFILE procedure.

95.6.6 START_LOGMNR Procedure

This procedure starts LogMiner by loading the dictionary that LogMiner will use to translate internal schema object identifiers to names.

In a CDB, the START_LOGMNR procedure must be called from the root database. You must have the LOGMINING administrative privilege to use this procedure.

Syntax

DBMS_LOGMNR.START_LOGMNR (  
    startScn IN NUMBER default 0,  
    endScn  IN NUMBER default 0,  
)
Parameters

Table 95-15  START_LOGMNR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startScn</td>
<td>Directs LogMiner to return only redo records with an SCN greater than or equal to the startScn specified. This fails if there is no redo log file containing the specified startScn value. (You can query the FILENAME, LOW_SCN, and NEXT_SCN columns in the V$LOGMNR_LOGS view for each redo log file to determine the range of SCN values contained in each redo log file.)</td>
</tr>
<tr>
<td>endScn</td>
<td>Directs LogMiner to return only redo records with an SCN less than or equal to the endScn specified. If you specify an endScn value that is beyond the value in any redo log file, then LogMiner uses the greatest endScn value in the redo log file that contains the most recent changes. (You can query the FILENAME, LOW_SCN, and NEXT_SCN columns in the V$LOGMNR_LOGS view for each redo log file to determine the range of SCN values contained in each redo log file.)</td>
</tr>
<tr>
<td>startTime</td>
<td>Directs LogMiner to return only redo records with a timestamp greater than or equal to the startTime specified. This fails if there is no redo log file containing the specified startTime value. (You can query the FILENAME, LOW_TIME, and HIGH_TIME columns in the V$LOGMNR_LOGS view for each redo log file to determine the range of time covered in each redo log file.)</td>
</tr>
<tr>
<td>endTime</td>
<td>Directs LogMiner to return only redo records with a timestamp less than or equal to the endTime specified. If you specify an endTime value that is beyond the value in any redo log file, then LogMiner will use the greatest endTime in the redo log file that contains the most recent changes. You can query the FILENAME, LOW_TIME, and HIGH_TIME columns in the V$LOGMNR_LOGS view for each redo log file to determine the range of time covered in each redo log file.)</td>
</tr>
<tr>
<td>DictFileName</td>
<td>Specifies the flat file that contains the LogMiner dictionary. It is used to reconstruct SQL_REDO and SQL_UNDO columns in V$LOGMNR_CONTENTS, as well as to fully translate SEG_NAME, SEG_OWNER, SEG_TYPE_NAME, TABLE_NAME, and TABLE_SPACE columns. The fully qualified path name for the LogMiner dictionary file must be specified. (This file must have been created previously through the DBMS_LOGMNR_D.BUILD procedure.) You need to specify this parameter only if neither DICT_FROM_REDO_LOGS nor DICT_FROM_ONLINE_CATALOG is specified.</td>
</tr>
<tr>
<td>options</td>
<td>See Table 95-2.</td>
</tr>
</tbody>
</table>

This parameter is ignored if startScn is specified. See the Usage Notes for additional information.

This parameter is ignored if endScn is specified. See the Usage Notes for additional information.

This parameter is ignored if endTime is specified. See the Usage Notes for additional information.

You need to specify this parameter only if neither DICT_FROM_REDO_LOGS nor DICT_FROM_ONLINE_CATALOG is specified.
Exceptions

Table 95-16  START_LOGMNR Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01280</td>
<td>Internal error encountered.</td>
</tr>
<tr>
<td>ORA-01281</td>
<td>startScn or endScn parameter value is not a valid SCN, or endScn is less than startScn.</td>
</tr>
<tr>
<td>ORA-01282</td>
<td>value for the startTime parameter was greater than the value specified for the endTime parameter, or there was no redo log file that was compatible with the date range specified with the startTime and endTime parameters.</td>
</tr>
<tr>
<td>ORA-01283</td>
<td>Options parameter specified is invalid.</td>
</tr>
<tr>
<td>ORA-01284</td>
<td>LogMiner dictionary file specified in the DictFileName parameter has a full path length greater than 256 characters, or the file cannot be opened.</td>
</tr>
<tr>
<td>ORA-01285</td>
<td>Error reading specified file.</td>
</tr>
<tr>
<td>ORA-01291</td>
<td>Redo log files that are needed to satisfy the user’s requested SCN or time range are missing.</td>
</tr>
<tr>
<td>ORA-01292</td>
<td>No log file has been specified for the current LogMiner session.</td>
</tr>
<tr>
<td>ORA-01293</td>
<td>Mounted database required for specified LogMiner options.</td>
</tr>
<tr>
<td>ORA-01294</td>
<td>Error occurred while processing information in the specified dictionary file, possible corruption.</td>
</tr>
<tr>
<td>ORA-01295</td>
<td>Specified LogMiner dictionary does not correspond to the database that produced the log files being analyzed.</td>
</tr>
<tr>
<td>ORA-01296</td>
<td>Character set mismatch between specified LogMiner dictionary and log files.</td>
</tr>
<tr>
<td>ORA-01297</td>
<td>Redo version mismatch between LogMiner dictionary and log files.</td>
</tr>
<tr>
<td>ORA-01299</td>
<td>Specified LogMiner dictionary corresponds to a different database incarnation.</td>
</tr>
<tr>
<td>ORA-01300</td>
<td>Writable database required for specified LogMiner options.</td>
</tr>
</tbody>
</table>

Usage Notes

- LogMiner can use a dictionary that you previously extracted to the redo log files or to a flat file, or you can specify that LogMiner use the online catalog if LogMiner is mining data from the source system.
- After executing the START_LOGMNR procedure, you can query the following views:
  - V$LOGMNR_CONTENTS - contains history of information in redo log files
  - V$LOGMNR_DICTIONARY - contains current information about the LogMiner dictionary file extracted to a flat file
  - V$LOGMNR_PARAMETERS - contains information about the LogMiner session

(You can query the V$LOGMNR_LOGS view after a redo log file list has been added to the list of files that LogMiner is to mine.)
• Parameters and options are not persistent across calls to 
\texttt{DBMS\_LOGMNR\_START\_LOGMNR}. You must specify all desired parameters and options 
(including SCN and time ranges) each time you call \texttt{DBMS\_LOGMNR\_START\_LOGMNR}.

• Be aware that specifying redo log files using a timestamp is not precise.

• The \texttt{CONTINUOUS\_MINE} option directs LogMiner to automatically add redo log files, 
as needed, to find the data of interest. You need to specify only the first log to start 
mining, or just the starting SCN or date to indicate to LogMiner where to begin 
mining logs. Keep the following in mind when using the \texttt{CONTINUOUS\_MINE} option:

  – The database control file will hold information about a limited number of 
arched redo log files, although the number of entries can be quite large. 
Query the \texttt{V\_ARCHIVED\_LOGS} view to determine which redo log file entries will 
be found by LogMiner.

  Even if an entry is listed in the database control file (and the \texttt{V\_ARCHIVED\_LOGS} 
view), the archived redo log file may not be accessible by LogMiner for various 
reasons. For example, the archived redo log file may have been deleted or 
moved from its location (maybe because of a backup operation to tape), or the 
directory where it resides may not be not available.

  – If you specify the \texttt{CONTINUOUS\_MINE} option and an ending time or SCN that will 
occur in the future (or you do not specify an end time or SCN), a query of the 
\texttt{V\_LOGMNR\_CONTENTS} view will not finish until the database has generated redo 
log files beyond the specified time or SCN. In this scenario, LogMiner will 
automatically add archived redo log files to the LogMiner redo log file list as 
they are generated. In addition, in this scenario only, LogMiner may 
automatically remove redo log files from the list to keep it at 50 processed 
redo files. This is to save PGA memory as LogMiner automatically adds redo 
log files to the list. If LogMiner did not perform automated removal, memory 
could eventually be exhausted.

  – LogMiner can mine online redo logs. However, if the \texttt{CONTINUOUS\_MINE} option 
is not specified, it is possible that the database is writing to the online redo log 
file at the same time that LogMiner is reading the online redo log file. If a log 
switch occurs while LogMiner is reading an online redo log file, the database 
will overwrite what LogMiner is attempting to read. The data that LogMiner 
returns if the file it is trying to read gets overwritten by the database is 
unpredictable.

• Keep the following in mind regarding starting and ending times or SCN ranges:

  – If you specify neither a \texttt{startTime} nor a \texttt{startScn} parameter, LogMiner will set 
the \texttt{startScn} parameter to use the lowest SCN value from the redo log file 
that contains the oldest changes.

  – If you specify both time and SCN values, LogMiner uses the SCN value or 
values and ignores the time values.

  – If you specify starting and ending time or SCN values and they are found in 
the LogMiner redo log file list, then LogMiner mines the logs indicated by those 
values.

  – If you specify starting and ending times or SCN values that are not in the 
LogMiner redo log file list, and you specify \texttt{DBMS\_LOGMNR\_START\_LOGMNR} without 
the \texttt{CONTINUOUS\_MINE} option, and you specify:

    * 0 for the \texttt{startTime} or \texttt{startScn} value, then the lowest SCN in the 
LogMiner redo log file list will be used as the \texttt{startScn}
* A nonzero number for the `startTime` or `startScn` value, then an error is returned.

* 0 or a nonzero number for the `endTime` or `endScn` value, then the highest SCN in the LogMiner redo log file list will be used as the `endScn`.

If you specify starting and ending times or SCN values and they are not found in the LogMiner redo log file list, and you specify `DBMS_LOGMNR.START_LOGMNR` with the `CONTINUOUS_MINE` option, and you specify:

* 0 for the `startTime` or `startScn` value, then an error is returned.

* A `startTime` or `startScn` value that is greater than any value in the database's archived redo log files, then LogMiner starts mining in the online redo log file. LogMiner will continue to process the online redo log file until it finds a change at, or beyond, the requested starting point before it returns rows from the `V$LOGMNR_CONTENTS` view.

* An `endTime` or `endScn` parameter value that indicates a time or SCN in the future, then LogMiner includes the online redo log files when it mines. When you query the `V$LOGMNR_CONTENTS` view, rows will be returned from this view as changes are made to the database, and will not stop until LogMiner sees a change beyond the requested ending point.

* 0 for the `endTime` or `endScn` parameter value, then LogMiner includes the online redo log files when it mines. When you query the `V$LOGMNR_CONTENTS` view, rows will be returned from this view as changes are made to the database, and will not stop until you enter CTL+C or you terminate the PL/SQL cursor.
DBMS_LOGMNR_D

The `DBMS_LOGMNR_D` package, one of a set of LogMiner packages, contains two subprograms: the `BUILD` procedure and the `SET_TABLESPACE` procedure.

- The `BUILD` procedure extracts the LogMiner data dictionary to either the redo log files or to a flat file. This information is saved in preparation for future analysis of redo log files using the LogMiner tool.
- The `SET_TABLESPACE` procedure re-creates all LogMiner tables in an alternate tablespace.

The LogMiner data dictionary consists of the memory data structures and the database tables that are used to store and retrieve information about objects and their versions. It is referred to as the LogMiner dictionary throughout the LogMiner documentation.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_LOGMNR_D Subprograms

See Also:

*Oracle Database Utilities* for information regarding LogMiner.

96.1 DBMS_LOGMNR_D Overview

LogMiner requires a dictionary to translate object IDs into object names when it returns redo data to you.

LogMiner gives you three options for supplying the dictionary:

- Using the online catalog
- Extracting a LogMiner dictionary to the redo log files
- Extracting a LogMiner dictionary to a flat file

Use the `BUILD` procedure to extract the LogMiner dictionary to the redo log files or a flat file. If you want to specify the online catalog as the dictionary source, you do so when you start LogMiner with the `DBMS_LOGMNR.START_LOGMNR` package.

Use the `SET_TABLESPACE` procedure if you want LogMiner tables to use a tablespace other than the default SYSAUX tablespace.
96.2 DBMS_LOGMNR_D Security Model

You must have the EXECUTE_CATALOG_ROLE role to use the DBMS_LOGMNR_D package.

96.3 Summary of DBMS_LOGMNR_D Subprograms

This table lists and briefly describes the DBMS_LOGMNR_D subprograms.

In a multitenant container database (CDB), some subprograms must be called from the root. There may be other differences as well. See the individual subprogram descriptions for details.

<table>
<thead>
<tr>
<th>Table 96-1 DBMS_LOGMNR_D Package Subprograms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subprogram</strong></td>
</tr>
<tr>
<td>BUILD Procedure</td>
</tr>
<tr>
<td>SET_TABLESPACE Procedure</td>
</tr>
</tbody>
</table>

96.3.1 BUILD Procedure

This procedure extracts the LogMiner data dictionary to either the redo log files or to a flat file.

The following considerations apply to a multitenant container database (CDB) environment.

- In a CDB environment, when you extract to the redo log files, the BUILD procedure must be called from the root database. The LogMiner data dictionary for the entire CDB is extracted to the redo log files.
- In a CDB environment, when you extract to flat files, you can mine only one pluggable database (PDB) at a time. You must first explicitly build the flat file dictionary for that specific PDB by running the BUILD procedure from it. The LogMiner data dictionary for each PDB for which the BUILD procedure is run is extracted to a flat file.
- You cannot add or remove PDBs from a CDB while this procedure is executing.

Syntax

```sql
DBMS_LOGMNR_D.BUILD (  
dictionary_filename IN VARCHAR2,  
dictionary_location IN VARCHAR2,  
options IN NUMBER);  
```
Parameters

Table 96-2  BUILD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary_filename</td>
<td>Specifies the name of the LogMiner dictionary file.</td>
</tr>
<tr>
<td>dictionary_location</td>
<td>Specifies the directory object for the LogMiner dictionary file.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies that the LogMiner dictionary is written to either a flat file (STORE_IN_FLAT_FILE) or the redo log files (STORE_IN_REDO_LOGS).</td>
</tr>
</tbody>
</table>

Exceptions

Table 96-3  BUILD Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01302</td>
<td>Dictionary build options are missing or incorrect.</td>
</tr>
<tr>
<td></td>
<td>This error is returned under the following conditions:</td>
</tr>
<tr>
<td></td>
<td>• If the value of the OPTIONS parameter is not one of the supported values (STORE_IN_REDO_LOGS, STORE_IN_FLAT_FILE) or is not specified</td>
</tr>
<tr>
<td></td>
<td>• If the STORE_IN_REDO_LOGS option is not specified and neither the dictionary_filename nor the dictionary_location parameter is specified</td>
</tr>
<tr>
<td></td>
<td>• If the STORE_IN_REDO_LOGS option is specified and either the dictionary_filename or the dictionary_location parameter is specified</td>
</tr>
<tr>
<td>ORA-01308</td>
<td>Initialization parameter UTL_FILE_DIR is not set.</td>
</tr>
<tr>
<td>ORA-01336</td>
<td>Specified dictionary file cannot be opened.</td>
</tr>
<tr>
<td></td>
<td>This error is returned under the following conditions:</td>
</tr>
<tr>
<td></td>
<td>• The dictionary file is read-only.</td>
</tr>
</tbody>
</table>

Note:

In prior releases, you used the UTL_FILE_DIR initialization parameter to specify a directory location. However, as of Oracle Database 18c, the UTL_FILE_DIR initialization parameter is desupported. It is still supported for backward compatibility, but Oracle recommends that you instead use directory objects.
Table 96-3  (Cont.) BUILD Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01308</td>
<td>Dictionary directory is not set.</td>
</tr>
<tr>
<td></td>
<td>This error is returned under the following conditions:</td>
</tr>
<tr>
<td></td>
<td>· The specified value for the dictionary_location is not a directory object.</td>
</tr>
<tr>
<td></td>
<td>· The specified value for the dictionary_location is a directory object that is defined to be a file path that cannot be accessed.</td>
</tr>
</tbody>
</table>

Usage Notes

- To extract the LogMiner dictionary to a flat file, you must supply a filename and location. The location must be specified as a database directory object. (See CREATE DIRECTORY in the Oracle Database SQL Language Reference.)

To extract the LogMiner dictionary to the redo log files, specify only the STORE_IN_REDO_LOGS option. The size of the LogMiner dictionary may cause it to be contained in multiple redo log files.

The combinations of parameters used result in the following behavior:
- If you do not specify any parameters, an error is returned.
- If you specify a filename and location, without any options, the LogMiner dictionary is extracted to a flat file with that name.
- If you specify a filename and location, as well as the STORE_IN_FLAT_FILE option, the LogMiner dictionary is extracted to a flat file with the specified name.
- If you do not specify a filename and location, but do specify the STORE_IN_REDO_LOGS option, the LogMiner dictionary is extracted to the redo log files.
- If you specify a filename and location, as well as the STORE_IN_REDO_LOGS option, an error is returned.
- If you do not specify a filename and location, but do specify the STORE_IN_FLAT_FILE option, an error is returned.

- Ideally, the LogMiner dictionary file will be created after all database dictionary changes have been made and prior to the creation of any redo log files that are to be analyzed. As of Oracle9i release 1 (9.0.1), you can use LogMiner to dump the LogMiner dictionary to the redo log files or a flat file, perform DDL operations, and dynamically apply the DDL changes to the LogMiner dictionary.

- The database must be open when you run the DBMS_LOGMNR_D.BUILD procedure.

- When extracting a LogMiner dictionary to a flat file, the procedure queries the dictionary tables of the current database and creates a text-based file containing the contents of the tables. To extract a LogMiner dictionary to a flat file, the following conditions must be met:
  - You must specify a directory for use by the PL/SQL procedure. To do so, first create an Oracle directory object with a pathname that specifies where the dictionary file is to be generated. For example:
create directory my_dictionary_dir as '/oracle/dictionary';

Be aware that the `DDL_DICT_TRACKING` option to the `DBMS_LOGMNR.START_LOGMNR` procedure is not supported for flat file dictionaries created prior to Oracle9i. If you attempt to use the `DDL_DICT_TRACKING` option with a LogMiner database extracted to a flat file prior to Oracle9i, the ORA-01330 error (problem loading a required build table) is returned.

- To extract a LogMiner dictionary file to the redo log files, the following conditions must be met:
  - The `DBMS_LOGMNR_D.BUILD` procedure must be run on a system that is running Oracle9i or later.
  - Archivelog mode must be enabled in order to generate usable redo log files.
  - The `COMPATIBLE` parameter in the initialization parameter file must be set to 9.2.0 or higher.
  - The database to which LogMiner is attached must be Oracle9i or later.

In addition, supplemental logging (at least the minimum level) should be enabled to ensure that you can take advantage of all the features that LogMiner offers.

Examples

Example 1: Extracting the LogMiner Dictionary to a Flat File

The following example extracts the LogMiner dictionary file to a flat file named `dictionary.ora` in a specified path (`/oracle/database`).

```sql
SQL> create directory my_dictionary_dir as '/oracle/database';
SQL> EXECUTE dbms_logmnr_d.build(dictionary_location=>my_dictionary_dir, dictionary_filename=>'dictionary.ora', options => dbms_logmnr_d.store_in_flat_file);
```

Example 2: Extracting the LogMiner Dictionary to the Redo Log Files

The following example extracts the LogMiner dictionary to the redo log files.

```sql
SQL> EXECUTE dbms_logmnr_d.build( options => dbms_logmnr_d.store_in_redo_logs);
```

96.3.2 SET_TABLESPACE Procedure

This procedure moves LogMiner tables from the default `SYSAUX` tablespace to an alternate tablespace.

By default, all LogMiner tables are created to use the `SYSAUX` tablespace. However, it may be desirable to have LogMiner tables use an alternate tablespace. Use this procedure to move LogMiner tables to this alternate tablespace.

In a CDB, only the LogMiner metadata in the local container is moved to the requested tablespace.

Syntax

```sql
DBMS_LOGMNR_D.SET_TABLESPACE ( new_tablespace IN VARCHAR2);
```
Parameters

Table 96-4  SET_TABLESPACE Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_tablespace</td>
<td>A string naming a preexisting tablespace. To move all LogMiner tables to employ this tablespace, supply this parameter.</td>
</tr>
</tbody>
</table>

Usage Notes

- Users upgrading from earlier versions of Oracle Database may find LogMiner tables in the SYSTEM tablespace. Oracle encourages such users to consider using the SET_TABLESPACE procedure to move the tables to the SYSAUX tablespace once they are confident that they will not be downgrading to an earlier version of Oracle Database.
- Users of this routine must supply an existing tablespace.

Example: Using the DBMS_LOGMNR_D.SET_TABLESPACE Procedure

The following example shows the creation of an alternate tablespace and execution of the DBMS_LOGMNR_D.SET_TABLESPACE procedure.

```
SQL> CREATE TABLESPACE logmnrts$ datafile '/usr/oracle/dbs/logmnrts.f'
       SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

SQL> EXECUTE dbms_logmnr_d.set_tablespace('logmnrts$');
```
The DBMS_LOGSTDBY package provides subprograms for configuring and managing the logical standby database environment.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Summary of DBMS_LOGSTDBY Subprograms

See Also:

Oracle Data Guard Concepts and Administration

97.1 DBMS_LOGSTDBY Overview

The DBMS_LOGSTDBY package helps you manage the SQL Apply (logical standby database) environment.

The subprograms in the DBMS_LOGSTDBY package help you to accomplish the following main objectives:

- Manage configuration parameters used by SQL Apply.
  For example, controlling how transactions are applied on the logical standby database, how much shared pool is used, and how many processes are used by SQL Apply to mine and apply the changes.
- Ensure an appropriate level of supplemental logging is enabled, and a LogMiner dictionary is built correctly for logical standby database creation.
- Provide a way to skip the application of changes to selected tables or entire schemas in the logical standby database, and specify ways to handle exceptions encountered by SQL Apply.
- Allow controlled access to tables in the logical standby database that may require maintenance.

97.2 DBMS_LOGSTDBY Security Model

You must have the DBA role to use the DBMS_LOGSTDBY package.

A prototype role, LOGSTDBY_ADMINISTRATOR, is created by default with RESOURCE and EXECUTE privileges on DBMS_LOGSTDBY. If you choose to use this role, consider granting ALTER
DATABASE and ALTER SESSION privileges to the role so that the grantee can start and stop SQL Apply and can enable and disable the database guard.

The procedures associated with skipping transactions (SKIP and UNSKIP, SKIP_ERROR and UNSKIP_ERROR, and SKIP_TRANSACTION and UNSKIP_TRANSACTION) all require DBA privileges to execute because their scope may contain wildcard schemas. Oracle recommends that where SKIP procedures are specified, these be owned by a secure account with appropriate privileges on the schemas they act on (for example, SYS).

The procedures associated with EDS-based replication require the EXECUTE privilege on the DBMS_LOGSTDBY package. They also require the CREATE TABLE and DROP TABLE system privileges to create (and drop) shadow tables in the target schema. If materialized views are used, then the CREATE MATERIALIZED VIEW and DROP Any MATERIALIZED VIEW system privileges are also required.

Note:
As of Oracle Database 12c Release 2 (12.2.0.2), Extended Datatype Support (EDS) is deprecated. All EDS-supported Oracle data types are now supported natively by logical standbys or Oracle GoldenGate.

97.3 DBMS_LOGSTDBY Constants

The DBMS_LOGSTDBY package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, DBMS_LOGSTDBY.SKIP_ACTION_SKIP.

The following table describes the constants for the proc_name parameter in the DBMS_LOGSTDBY.SKIP procedure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_EVENTS</td>
<td>Maximum number of events to log in the DBA_LOGSTDBY_EVENTS view. See the DBMS_LOGSTDBY APPLY_SET Procedure.</td>
</tr>
<tr>
<td>SKIP_ACTION_APPLY</td>
<td>Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to apply the DDL or PL/SQL statement.</td>
</tr>
<tr>
<td>SKIP_ACTION_ERROR</td>
<td>Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to error out.</td>
</tr>
<tr>
<td>SKIP_ACTION_REPLACE</td>
<td>Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to apply the replacement DDL. (This constant should not be used while handling an Oracle Supplied PL/SQL procedure call).</td>
</tr>
</tbody>
</table>
Table 97-1    (Cont.) Constants for SKIP Options Flag

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKIP_ACTION_SKIP</td>
<td>Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to skip the associated DDL or Oracle supplied PL/SQL procedure call.</td>
</tr>
</tbody>
</table>

97.4 Summary of DBMS_LOGSTDBY Subprograms

This table describes each subprogram of the DBMS_LOGSTDBY procedure, including a reference to the section where each procedure is described in more detail.

In a multitenant container database (CDB), some subprograms must be called from the root. There may be other differences as well. See the individual subprogram descriptions for details.

Note:

As of Oracle Database 12c Release 2 (12.2.0.2), DBMS_LOGSTDBY subprograms related to Extended Datatype Support (EDS) are deprecated. All EDS-supported Oracle data types are now supported natively by logical standbys or Oracle GoldenGate.

Table 97-2    DBMS_LOGSTDBY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY_SET Procedure</td>
<td>Sets the values of various parameters that configure and maintain SQL Apply.</td>
</tr>
<tr>
<td>APPLY_UNSET Procedure</td>
<td>Restores the default values of various parameters that configure and maintain SQL Apply.</td>
</tr>
<tr>
<td>BUILD Procedure</td>
<td>Ensures supplemental logging is enabled properly and builds the LogMiner dictionary.</td>
</tr>
<tr>
<td>EDS_ADD_TABLE</td>
<td>Adds EDS-based replication for the table. It should be invoked on the primary database first and then on the standby database.</td>
</tr>
<tr>
<td>EDS_EVOLVE_AUTOMATIC</td>
<td>Enables or disables automatic DDL handling for EDS tables.</td>
</tr>
<tr>
<td>EDS_EVOLVE_MANUAL</td>
<td>Allows you to evolve EDS tables manually (that is, manually take compensating actions based on DDLs on the base tables with EDS-replication).</td>
</tr>
<tr>
<td>EDS_REMOVE_TABLE</td>
<td>Removes EDS-based replication for the specified table. It also drops the shadow table and triggers.</td>
</tr>
<tr>
<td>INSTANTIATE_TABLE</td>
<td>Creates and populates a table in the standby database from a corresponding table in the primary database.</td>
</tr>
</tbody>
</table>
### Table 97-2  (Cont.) DBMS_LOGSTDBY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_APPLY_SERVER Function</td>
<td>This function returns <strong>TRUE</strong> if it is executed from PL/SQL in the context of a logical standby apply server process. This function is used in conjunction with triggers that have the <code>fire_once</code> parameter in the <code>DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY</code> subprogram set to <strong>FALSE</strong> (the default is <strong>TRUE</strong>). Such triggers are executed when the relevant target is updated by an apply process. This function can be used within the body of the trigger to ensure that the trigger takes different (or no) actions on the primary or on the standby.</td>
</tr>
<tr>
<td>MAP_PRIMARY_SCN Function</td>
<td>Maps an SCN relevant to the primary database to a corresponding SCN at the logical standby database. The mapped SCN is conservative in nature, and can thus be used to flash back the logical standby database to compensate for a flashback database operation performed at the primary database.</td>
</tr>
<tr>
<td>PREPARE_FOR_NEW_PRIMARY Procedure</td>
<td>Used after a failover, this procedure ensures a local logical standby database that was not involved in the failover has not processed more redo than the new primary database and reports the set of archive redo log files that must be replaced to ensure consistency.</td>
</tr>
<tr>
<td>PURGE_SESSION Procedure</td>
<td>Identifies the archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply.</td>
</tr>
<tr>
<td>REBUILD Procedure</td>
<td>Records relevant metadata (including the LogMiner dictionary) in the redo stream in case a database that has recently changed its role to a primary database following a failover operation fails to do so during the failover process.</td>
</tr>
<tr>
<td>SET_TABLESPACE Procedure</td>
<td>Moves metadata tables required by SQL Apply to the user-specified tablespace. By default, the metadata tables are created in the SYSAUX tablespace.</td>
</tr>
<tr>
<td>SKIP Procedure</td>
<td>Specifies rules that control database operations that should not be applied to the logical standby database.</td>
</tr>
<tr>
<td>SKIP_ERROR Procedure</td>
<td>Specifies rules regarding what action to take upon encountering errors.</td>
</tr>
<tr>
<td>SKIP_TRANSACTION Procedure</td>
<td>Specifies transactions that should not be applied on the logical standby database. Be careful in using this procedure, because not applying specific transactions may cause data corruption at the logical standby database.</td>
</tr>
<tr>
<td>UNSKIP Procedure</td>
<td>Deletes rules specified by the <strong>SKIP</strong> procedure.</td>
</tr>
<tr>
<td>UNSKIP_ERROR Procedure</td>
<td>Deletes rules specified by the <strong>SKIP_ERROR</strong> procedure.</td>
</tr>
<tr>
<td>UNSKIP_TRANSACTION Procedure</td>
<td>Deletes rules specified by the <strong>SKIP_TRANSACTION</strong> procedure.</td>
</tr>
</tbody>
</table>
97.4.1 APPLY_SET Procedure

Use this procedure to set values of parameters that configure and manage SQL Apply in a logical standby database environment. All parameters, except for PRESERVE_COMMIT_ORDER, can be changed without having to stop SQL Apply.

In a CDB, the APPLY_SET procedure must be called from the root database.

Syntax

```
DBMS_LOGSTDBY.APPLY_SET (  
    inname      IN VARCHAR,  
    value       IN VARCHAR);  
```

Parameters

Table 97-3  APPLY_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY_SERVERS</td>
<td>Controls the number of APPLIER processes used to apply changes. The maximum number allowed is 1024, provided the MAX_SERVERS parameter is set to accommodate this.</td>
</tr>
</tbody>
</table>
| EVENT_LOG_DEST     | Controls where SQL Apply records the occurrence of an interesting event. It takes the following values:  
                      • DEST_ALL - All events will be recorded in the DBA_LOGSTDBY_EVENTS view and in the alert log.  
                      • DEST_EVENTS_TABLE - All events that contain information about user data will be recorded only in the DBA_LOGSTDBY_EVENTS view. This is the default value.  
                      For example, if SQL Apply receives an ORA-1403 error, the whole event is recorded in the DBA_LOGSTDBY_EVENTS view. Whereas, the alert log records only that SQL Apply stopped because of ORA-1403. No information regarding the user table or offending statement is logged in the alert log. However, if you stop the SQL Apply engine, it gets recorded in both the DBA_LOGSTDBY_EVENTS view and in the alert log.  
                      Note that this parameter affects the behavior of the following parameters: RECORD_APPLIEDDDL, RECORD_SKIPDDL, RECORD_SKIPERRORS, and RECORD_UNSUPPORTEDOPERATIONS. For example, if RECORD_APPLIEDDDL is set to TRUE, but EVENT_LOG_DEST is set to DEST_EVENTS_TABLE, then the applied DDL string will only be recorded in the DBA_LOGSTDBY_EVENTS view.  
| LOG_AUTO_DEL_RETENTION_TARGET | This parameter setting is only meaningful if LOG_AUTO_DELETE has been set to TRUE. The value you supply for this parameter controls how long (in minutes) a remote archived log that is received from the primary database will be retained at the logical standby database once all redo records contained in the log have been applied at the logical standby database. The default value is 1440 minutes. |
Table 97-3  (Cont.) APPLY_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_AUTO_DELETE</td>
<td>Automatically deletes foreign archived redo log files as soon as they have been applied on the logical standby database. By default, a foreign archived redo log file is not deleted until 24 hours (the default value of LOG_AUTO_DEL_RETENTION_TARGET parameter) after it has been applied at the logical standby database. Set to TRUE to enable automatic deletion of archived redo log files. Set to FALSE to disable automatic deletion. The default value is TRUE.</td>
</tr>
<tr>
<td>MAX_EVENTS_RECORDED</td>
<td>Number of recent events that will be visible through the DBA_LOGSTDBY_EVENTS view. To record all events encountered by SQL Apply, use the DBMS_LOGSTDBY.MAX_EVENTS constant as the number value. The default value is 10,000.</td>
</tr>
<tr>
<td>MAX_SERVERS</td>
<td>Number of processes that SQL Apply uses to read and apply redo. The default value is 9. The maximum number allowed is 2048.</td>
</tr>
<tr>
<td>MAX_SGA</td>
<td>Number of megabytes from shared pool in System Global Area (SGA) that SQL Apply will use. The default value is 30 megabytes or one quarter of the value set for SHARED_POOL_SIZE, whichever is lower. The maximum size allowed is 4095 megabytes.</td>
</tr>
<tr>
<td>PREPARE_SERVERS</td>
<td>Controls the number of PREPARER processes used to prepare changes. The maximum number allowed is 1024, provided the MAX_SERVERS parameter is set to accommodate this.</td>
</tr>
<tr>
<td>PRESERVE_COMMIT_ORDER</td>
<td>TRUE: Transactions are applied to the logical standby database in the exact order in which they were committed on the primary database. This is the default parameter setting. FALSE: Transactions containing non-overlapping sets of rows may be committed in a different order than they were committed on the primary database. Regardless of the level chosen, modifications done to the same row are always applied in the same order as they happened on the primary database. See the Usage Notes for details and recommendations. You cannot modify this parameter while SQL Apply is running.</td>
</tr>
<tr>
<td>RECORD_APPLIED_DDL</td>
<td>Controls whether DDL statements that have been applied to the logical standby database are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values: TRUE: Indicates that DDL statements applied to the logical standby database are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. FALSE: Indicates that applied DDL statements are not recorded. This is the default parameter setting.</td>
</tr>
</tbody>
</table>
Table 97-3  (Cont.) APPLY_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD_SKIP_DDL</td>
<td>Controls whether skipped DDL statements are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td><strong>TRUE</strong>: Skipped DDL statements are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting.</td>
</tr>
<tr>
<td></td>
<td><strong>FALSE</strong>: Skipped DDL statements are not recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.</td>
</tr>
<tr>
<td>RECORD_SKIP_ERRORS</td>
<td>Controls whether skipped errors (as described by the SKIP_ERROR procedure) are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td><strong>TRUE</strong>: Skipped errors are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting.</td>
</tr>
<tr>
<td></td>
<td><strong>FALSE</strong>: Skipped errors are not recorded in the DBA_LOGSTDBY Events table and the alert log.</td>
</tr>
<tr>
<td>RECORD_UNSUPPORTED_OPERATIONS</td>
<td>Captures information about transactions running on the primary database that will not be supported by a logical standby database. This procedure records its information as events in the DBA_LOGSTDBY_EVENTS table. Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td><strong>TRUE</strong>: The information is captured and recorded as events in the DBA_LOGSTDBY_EVENTS table.</td>
</tr>
<tr>
<td></td>
<td><strong>FALSE</strong>: The information is not captured. This is the default.</td>
</tr>
</tbody>
</table>

If a parameter is changed while SQL Apply is running, the change will take effect at some point in the future. In such a case, an informational row is inserted into the DBA_LOGSTDBY_EVENTS view at the time the parameter change takes effect.

Additionally, if you are modifying a parameter while SQL Apply is running on an Oracle RAC configuration, you must be connected to the same instance where SQL Apply is running.

Exceptions

Table 97-4  APPLY_SET Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>invalid Logical Standby option requested</td>
</tr>
<tr>
<td>ORA-16236</td>
<td>Logical Standby metadata operation in progress</td>
</tr>
</tbody>
</table>

Usage Notes

- Use the APPLY_UNSET procedure to restore the default settings of a parameter.
- See Oracle Data Guard Concepts and Administration for help with tuning SQL Apply and for information about setting appropriate values for different parameters.
Examples

To record DDLs in the DBA_LOGSTDBY_EVENTS view and in the alert log, issue the following statement:

SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('RECORD_APPLIED_DDL', TRUE);

97.4.2 APPLY_UNSET Procedure

Use the APPLY_UNSET procedure to restore the default values of the parameters that you changed with the APPLY_SET procedure.

In a CDB, the APPLY_UNSET procedure must be called from the root database.

Syntax

DBMS_LOGSTDBY.APPLY_UNSET (inname IN VARCHAR);

Parameters

The parameter information for the APPLY_UNSET procedure is the same as that described for the APPLY_SET procedure. See Table 97-3 for complete parameter information.

Exceptions

Table 97-5 APPLY_UNSET Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>invalid Logical Standby option requested</td>
</tr>
<tr>
<td>ORA-16236</td>
<td>Logical Standby metadata operation in progress</td>
</tr>
</tbody>
</table>

Usage Notes

- Use the APPLY_SET procedure to specify a nondefault value for a parameter.

Examples

If you previously specified that applied DDLs show up in the DBA_LOGSTDBY_EVENTS view and the alert log, you can restore the default behavior of SQL Apply regarding applied DDL statements with the following statement:

SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('RECORD_APPLIED_DDL');

97.4.3 BUILD Procedure

Use this procedure on the primary database to record relevant metadata (LogMiner dictionary) information in the redo log, which will subsequently be used by SQL Apply.
This procedure will enable database-wide primary- and unique-key supplemental logging, if necessary.

In a CDB, the `BUILD` procedure must be called from the root database on the primary. Additionally, you cannot add or remove PDBs from a CDB while this procedure is executing.

---

**Note:**

In databases created using Oracle Database 11g release 2 (11.2) or later, supplemental logging information is automatically propagated to any existing physical standby databases. However, for databases in earlier releases, or if the database was created using an earlier release and then upgraded to 11.2, you must check whether supplemental logging is enabled at the physical standby(s) if it is also enabled at the primary database. If it is not enabled at the physical standby(s), then before performing a switchover or failover, you must enable supplemental logging on all existing physical standby databases. To do so, issue the following SQL command on each physical standby:

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX) COLUMNS;
```

If you do not do this, then any logical standby that is also in the same Data Guard configuration will be unusable if a switchover or failover is performed to one of the physical standby databases. If a switchover or failover has already occurred and supplemental logging was not enabled, then you must recreate all logical standby databases.

---

**Syntax**

`DBMS_LOGSTDBY.BUILD;`

**Usage Notes**

- Supplemental log information includes extra information in the redo logs that uniquely identifies a modified row in the logical standby database, and also includes information that helps efficient application of changes to the logical standby database.
- LogMiner dictionary information allows SQL Apply to interpret data in the redo logs.
- `DBMS_LOGSTDBY.BUILD` should be run only once for each logical standby database you want to create. You do not need to use `DBMS_LOGSTDBY.BUILD` for each Oracle RAC instance.
- `DBMS_LOGSTDBY.BUILD` waits for all transactions (including distributed transactions) that are active at the time of the procedure invocation to complete before returning. See *Oracle Database Administrator's Guide* for information about how to handle in-doubt transactions.

**Examples**

To build the LogMiner dictionary in the redo stream of the primary database and to record additional information so that a logical standby database can be instantiated, issue the following SQL statement at the primary database

```
SQL> EXECUTE DBMS_LOGSTDBY.BUILD;
```
97.4.4 EDS_ADD_TABLE

This procedure adds EDS-based replication for the table. It should be invoked on the primary database first and then on the standby database.

Syntax

```sql
DBMS_LOGSTDBY.EDS_ADD_TABLE (  
    table_owner        IN VARCHAR2,  
    table_name         IN VARCHAR2,  
    p_dblink           IN VARCHAR2);
```

Parameters

**Table 97-6**  EDS_ADD_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_owner</td>
<td>Owner of the table to be created or re-created in the standby database</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table to be created or re-created in the standby database</td>
</tr>
<tr>
<td>p_dblink</td>
<td>A database link to the primary database</td>
</tr>
</tbody>
</table>

Exceptions

**Table 97-7**  EDS_ADD_TABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-942</td>
<td>Table or view does not exist</td>
</tr>
<tr>
<td>ORA-16103</td>
<td>SQL Apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16130</td>
<td>Supplemental log information is missing from log stream</td>
</tr>
<tr>
<td>ORA-16236</td>
<td>Logical standby metadata operation in progress</td>
</tr>
<tr>
<td>ORA-16276</td>
<td>Specified database link does not correspond to primary database</td>
</tr>
<tr>
<td>ORA-16278</td>
<td>Specified table has a multi-object skip rule defined</td>
</tr>
<tr>
<td>ORA-16279</td>
<td>Specified database link does not have the necessary roles for operation</td>
</tr>
<tr>
<td>ORA-16302</td>
<td>Extended datatype support is not supported for specified table</td>
</tr>
<tr>
<td>ORA-16303</td>
<td>Specified table already has EDS on this database</td>
</tr>
<tr>
<td>ORA-16304</td>
<td>Procedure must first be called at the primary database</td>
</tr>
<tr>
<td>ORA-16306</td>
<td>Specified table does not have a primary key</td>
</tr>
</tbody>
</table>

Usage Notes

- When this procedure is invoked on the primary, the definition of the local table is used to compose the shadow table and both the base and shadow table triggers.
- When this procedure is invoked on the standby, a database link to the primary database (p_dblink) must be provided so that the definition of the table on the primary can...
be used to compose the shadow table and its trigger on the standby. After the shadow table and its trigger are composed on the standby, the DBMS_LOGSTDBY.INSTANTIATE_TABLE procedure is called on the base table, using the provided database link to the primary. The p_dblink parameter is ignored when called on the primary and it defaults to NULL.

- Error checking is done up front. Generally, if the table contains a datatype that this procedure does not support, then an error is raised.

If the table is already supported by SQL Apply, then an exception is raised. To determine whether a table is a candidate for extended datatype support (EDS), query the DBA_LOGSTDBY_EDS_SUPPORTED view on the primary.

- This procedure must be executed on the primary before it is executed on the standby; otherwise an error is raised.

- For role changes to a logical standby it is essential that EDS be added to both the primary and the standby prior to the role change. If it is added only on the primary, then it must be removed after the role change because the EDS context would then be invalid.

- For rolling upgrades the procedure only needs to be called on the primary.

- This procedure is not replicated to a logical standby and therefore must be executed on both the primary and the standby databases. It is replicated to a physical standby so when it is used for rolling upgrades it should only be called on the primary.

- When this procedure is executed on the standby, it requires that SQL Apply be stopped and the database guard set to either STANDBY or NONE. When the procedure is complete, the table is protected by the database guard.

Examples

To add EDS for the table OE.CUSTOMERS, issue the following commands, first on the primary and then on the standby.

On the primary database, issue the following command:

```sql
execute dbms_logstdby.eds_add_table('OE','CUSTOMERS');
```

On the standby database, issue the following commands:

```sql
alter database stop logical standby apply;
execute dbms_logstdby.eds_add_table('OE','CUSTOMERS','dblink-to-primary');
alter database start logical standby apply immediate;
```

97.4.5 EDS_EVOLVE_AUTOMATIC

This procedure either enables or disables automatic DDL handling for EDS tables.

Syntax

```sql
DBMS_LOGSTDBY.EDS_EVOLVE_AUTOMATIC (
    options IN VARCHAR2);
```
Parameters

Table 97-8 EDS_EVOLVE_AUTOMATIC Parameter Descriptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Specify ENABLE to enable automatic DDL handling for EDS. Specify DISABLE to disable automatic DDL handling for EDS.</td>
</tr>
</tbody>
</table>

Exceptions

Table 97-9 EDS_EVOLVE_AUTOMATIC Exception Descriptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16104</td>
<td>The specified parameter is not valid</td>
</tr>
<tr>
<td>ORA-16305</td>
<td>Procedure not supported on a logical standby</td>
</tr>
</tbody>
</table>

Usage Notes

- For transparent handling of DDLs on all EDS-enabled tables, this procedure must be called at the primary database once (with the ENABLE option) prior to the first call to EDS_ADD_TABLE.
- If an EDS-specific DDL trigger is enabled, it fires after every DDL to determine whether the DDL affects the viability of EDS-replication of any of the tables presently enabled for EDS. If it does, then a separate EVOLVING trigger is generated on the table that ensures no DMLs are allowed on the affected base table until the compensating actions are taken to repair the EDS-specific metadata.
- If the SQL statements ALTER TABLE or CREATE UNIQUE INDEX are executed on an EDS-maintained table, then an evolve operation is started if automatic evolve has been enabled.
- This procedure can be called on a primary database only and is replicated via SQL Apply to a logical standby database.
- When the procedure is executed on the standby it succeeds regardless of the database guard setting and whether or not apply is running.

Examples

To enable automatic evolution of all EDS-maintained tables, issue the following statement:

```sql
execute dbms_logstdby.eds_evolve_automatic('ENABLE');
```

After automatic evolve has been enabled, there is no need to issue eds_evolve commands. They will be done automatically for you upon completion of a DDL command issued against an EDS table.

To disable automatic evolution of all EDS-maintained tables, issue the following statement:

```sql
execute dbms_logstdby.eds_evolve_automatic('DISABLE');
```
97.4.6 EDS_EVOLVE_MANUAL

This procedure allows you to evolve EDS tables manually (that is, manually take compensating actions based on DDLs on the base tables with EDS-replication).

Syntax

```
DBMS_LOGSTDBY.EDS_EVOLVE_MANUAL (
    options    IN VARCHAR2,
    table_owner IN VARCHAR2,
    table_name  IN VARCHAR2);
```

Parameters

**Table 97-10  EDS_EVOLVE_MANUAL Parameter Descriptions**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Valid options are:</td>
</tr>
<tr>
<td></td>
<td>• START - start an evolve operation, prior to performing the DDL</td>
</tr>
<tr>
<td></td>
<td>• FINISH - finish an evolve operation after performing the DDL</td>
</tr>
<tr>
<td></td>
<td>• CANCEL - cancel a started evolve operation</td>
</tr>
<tr>
<td>table_owner</td>
<td>Owner of the table to be created or re-created in the standby database</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table to be created or re-created in the standby database</td>
</tr>
</tbody>
</table>

Exceptions

**Table 97-11  EDS_EVOLVE_MANUAL Exception Descriptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16302</td>
<td>Extended datatype support is not supported for specified table</td>
</tr>
<tr>
<td>ORA-16305</td>
<td>Procedure not supported on a logical standby</td>
</tr>
<tr>
<td>ORA-16306</td>
<td>Specified table does not have a primary key</td>
</tr>
<tr>
<td>ORA-16307</td>
<td>Table not prepared for extended datatype support operation</td>
</tr>
</tbody>
</table>

Usage Notes

- To perform a manual evolve operation, you must call `EDS_EVOLVE_MANUAL` before and after any DDL that may affect the base table being replicated with EDS. The following is the prescribed method for evolving manually:
  1. Call `EDS_EVOLVE_MANUAL` with the `START` option.
  2. Perform the DDL.
  3. Call `EDS_EVOLVE_MANUAL` with the `FINISH` option.

Deviation from this order could lead to data loss. An error is returned if the procedure is called with the `FINISH` option before it is called with `START` for the specified table. When the procedure is called with the `START` option, DML on the table is blocked by an evolve trigger.
This procedure can only be invoked on a primary database.

When this procedure is executed on the standby, SQL Apply must be stopped and the database guard set to either STANDBY or NONE. When the procedure is complete, the table is protected by the database guard.

Examples

To manually evolve an EDS table, issue the following commands on the primary database:

```sql
execute dbms_logstdby.eds_evolve_manual('START','OE','CUSTOMERS');
alter table oe.customers some_DDL_construct;
execute dbms_logstdby.eds_evolve_manual('FINISH','OE','CUSTOMERS');
```

97.4.7 EDS_REMOVE_TABLE

This procedure removes EDS-based replication for the specified table. It also drops the shadow table and triggers.

Syntax

```sql
DBMS_LOGSTDBY.EDS_REMOVE_TABLE (  
    table_owner        IN VARCHAR2,
    table_name         IN VARCHAR2,
)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_owner</td>
<td>Owner of the table to be created or re-created in the standby database</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table to be created or re-created in the standby database</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
</tbody>
</table>

Usage Notes

- When this procedure is called on the standby, the shadow table and both triggers are dropped locally but not on the primary.

  When this procedure is called on the primary, the shadow table and both triggers are dropped and the procedure is re-executed on the standby by the apply engine. Therefore, the supporting shadow table and triggers are also dropped at the logical standby. Using this approach, a single call made on the primary can remove EDS from the entire configuration for a table.
In a multi-standby configuration, you can remove support on a single standby by calling this procedure on that standby.

This procedure takes out an exclusive table lock to ensure isolation in the presence of ongoing user writes on the same table. Thus the procedure will block until existing write-transactions that have outstanding row locks on the table are completed.

When this procedure is executed on the standby, SQL Apply must be stopped and the database guard set to either STANDBY or NONE. When the procedure is complete, the table is protected by the database guard.

Examples

To remove EDS for the table OE.CUSTOMERS, issue the following command on the primary database:

```sql
execute dbms_logstdby.eds_remove_table('OE','CUSTOMERS');
```

97.4.8 INSTANTIATE_TABLE Procedure

This procedure creates and populates a table in the standby database from a corresponding table in the primary database.

The table requires the name of the database link (dblink) as an input parameter. If the table already exists in the logical standby database, it will be dropped and re-created based on the table definition at the primary database. This procedure only brings over the data associated with the table, and not the associated indexes and constraints.

Use the INSTANTIATE_TABLE procedure to:

- Add a table to a standby database.
- Re-create a table in a standby database.

In a CDB, the INSTANTIATE_TABLE procedure must be called from within the container in which the table to be instantiated resides. Additionally, the database link that is provided to the primary database must point to the corresponding container on the primary.

Syntax

```
DBMS_LOGSTDBY.INSTANTIATE_TABLE (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2,
    dblink IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table to be created or re-created in the standby database</td>
</tr>
<tr>
<td>dblink</td>
<td>Name of the database link account that has privileges to read and lock the table in the primary database, as well as the SELECT_CATALOG_ROLE on the primary database</td>
</tr>
</tbody>
</table>
Exceptions

Table 97-15 INSTANTIATE_TABLE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16236</td>
<td>Logical Standby metadata operation in progress</td>
</tr>
<tr>
<td>ORA-16276</td>
<td>Specified database link does not correspond to primary database</td>
</tr>
<tr>
<td>ORA-16277</td>
<td>Specified table is not supported by logical standby database</td>
</tr>
<tr>
<td>ORA-16278</td>
<td>Specified table has a multi-object skip rule defined</td>
</tr>
</tbody>
</table>

Usage Notes

- Use this procedure to create and populate a table in a way that keeps the data on the standby database transactionally consistent with the primary database.
- This table will not be synchronized with the rest of the tables being maintained by SQL Apply and SQL Apply will not start to maintain it until SQL Apply encounters redo that occurred after the table was instantiated from the primary. The SCN at which the table was instantiated from the primary database is available in the DBA_LOGSTDBY_EVENTS view.
- The specified table must be a table that is supported by logical standby (that is, it does not appear in the DBA_LOGSTDBY_UNSUPPORTED_TABLES view on the primary database).
- If there are any skip rules that specifically name this table (without any wildcards), those skip rules will be dropped as part of INSTANTIATE_TABLE, so that the table will be properly maintained by SQL Apply in the future. If there are skip rules that indirectly reference this table (match a skip rule with a wildcard in the schema_name or table_name, and have a TABLE, DML, or SCHEMA_DDL statement type), INSTANTIATE_TABLE will fail with an ORA-16278 error. Any multi-object skip rules that pertain to the table must be dropped or changed before re-attempting the INSTANTIATE_TABLE call.

Examples

```sql
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE (-
      SCHEMA_NAME => 'HR', TABLE_NAME => 'EMPLOYEES', -
      DBLINK => 'INSTANTIATE_TBL_LINK');
```

97.4.9 IS_APPLY_SERVER Function

This function returns TRUE if it is executed from PL/SQL in the context of a logical standby apply server process.

This function is used in conjunction with triggers that have the fire_once parameter in the DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY subprogram set to FALSE (the default is TRUE). Such triggers are executed when the relevant target is updated by an apply process. This function can be used within the body of the trigger to ensure that the trigger takes different (or no) actions on the primary or on the standby.
Syntax

```sql
DBMS_LOGSTDBY.IS_APPLY_SERVER
RETURN BOOLEAN;
```

Parameters

None

97.4.10 MAP_PRIMARY_SCN Function

This function returns an SCN on the standby that predates the supplied SCN from the primary database by at least 5 minutes.

It can be used to determine a safe SCN to use in a compensating flashback database operation at the logical standby database, following a flashback database operation or a point-in-time recovery operation at the primary database.

Syntax

```sql
DBMS_LOGSTDBY.MAP_PRIMARY_SCN(primary_scn NUMBER) RETURN NUMBER;
```

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20001</td>
<td>Primary SCN is before mapped range</td>
</tr>
<tr>
<td>ORA-20002</td>
<td>SCN mapping requires PRESERVE_COMMIT_ORDER to be TRUE</td>
</tr>
</tbody>
</table>

Usage Notes

Use this function to get a conservative SCN at the logical standby database that corresponds to an SCN at the primary database. This function is useful in the context of doing compensating flashback database operations at the logical standby following a flashback database operation or a point-in-time recovery operation done at the primary database.

97.4.11 PREPARE_FOR_NEW_PRIMARY Procedure

The `PREPARE_FOR_NEW_PRIMARY` procedure must be invoked at a logical standby database following a failover, if that standby database was not the target of the failover operation.

Such a standby database must process the exact same set of redo logs processed at the new primary database. This routine ensures that the local logical standby database has not processed more redo than the new primary database and reports the set of archive logs that must be replaced to ensure consistency. The set of replacement logs will be reported in the alert.log. These logs must be copied to the logical standby and registered using the `ALTER DATABASE REGISTER LOGICAL LOGFILE` statement.

In a CDB, the `PREPARE_FOR_NEW_PRIMARY` procedure must be called from the root database.
Syntax

DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (
   FORMER_STANDBY_TYPE IN VARCHAR2,
   DBLINK IN VARCHAR2);

Parameters

Table 97-17  PREPARE_FOR_NEW_PRIMARY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMER_STANDBY_TYPE</td>
<td>The type of standby database that was the target of the failover operation to become the new primary database. Valid values are 'PHYSICAL' if the new primary was formerly a physical standby, and 'LOGICAL' if the new primary database was formerly a logical standby database.</td>
</tr>
<tr>
<td>DBLINK</td>
<td>The name of a database link to the new primary database</td>
</tr>
</tbody>
</table>

Exceptions

Table 97-18  PREPARE_FOR_NEW_PRIMARY Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16104</td>
<td>Invalid Logical Standby option.</td>
</tr>
<tr>
<td>ORA-16109</td>
<td>Failed to apply log data from previous primary.</td>
</tr>
</tbody>
</table>

Usage Notes

- This routine is intended only for logical standby systems. This routine will fail if the new primary database was formerly a logical standby database and the LogMiner dictionary build has not completed successfully. Log files displayed in the alert log will be referred to as terminal logs. Users should keep in mind that file paths are relative to the new primary database and may not resolve locally. Upon manual registration of the terminal logs, users should complete the process by calling either `START LOGICAL STANDBY APPLY` if the new primary database was formerly a physical standby database or `START LOGICAL STANDBY APPLY NEW PRIMARY` if the new primary database was formerly a logical standby database. See the alert log for more details regarding the reasons for any exception.

Examples

`SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (  - FORMER_STANDBY_TYPE => 'LOGICAL',  - DBLINK => 'dblink_to_newprimary');`

97.4.12 PURGE_SESSION Procedure

PURGE_SESSION identifies all archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply.

Once identified, you can issue operating system commands to delete some or all of the unnecessary archived redo log files.
In a CDB, the **PURGE_SESSION** procedure must be called from the root database.

**Syntax**

```
DBMS_LOGSTDBY.PURGE_SESSION;
```

**Exceptions**

**Table 97-19** **PURGE_SESSION Procedure Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01309</td>
<td>Invalid session</td>
</tr>
</tbody>
</table>

**Usage Notes**

- This procedure does not delete the archived redo log files. You must issue operating system commands to delete unneeded files.
- This procedure updates the **DBA_LOGMNR_PURGED_LOG** view that displays the archived redo log files that have been applied to the logical standby database.
- In Oracle Database 10g Release 2, metadata related to the archived redo log files (and the actual archived redo log files) are purged automatically based on the default setting of the **LOG_AUTO_DELETE** parameter described in the **DBMS_LOGSTDBY.APPLY_SET** procedure described.

**Example**

To identify and remove unnecessary files:

1. Enter the following statement on the logical standby database:

   ```
   SQL> EXECUTE DBMS_LOGSTDBY.PURGE_SESSION;
   ```

2. Query the **DBA_LOGMNR_PURGED_LOG** view to list the archived redo log files that can be removed:

   ```
   SQL> SELECT * FROM DBA_LOGMNR_PURGED_LOG;
   ```

   **FILE_NAME**
   
   `/boston/arc_dest/arc_1_40_509538672.log`
   `/boston/arc_dest/arc_1_41_509538672.log`
   `/boston/arc_dest/arc_1_42_509538672.log`
   `/boston/arc_dest/arc_1_43_509538672.log`
   `/boston/arc_dest/arc_1_44_509538672.log`
   `/boston/arc_dest/arc_1_45_509538672.log`
   `/boston/arc_dest/arc_1_46_509538672.log`
   `/boston/arc_dest/arc_1_47_509538672.log`

3. Use operating system-specific commands to delete archived redo log files from the file system.
97.4.13 REBUILD Procedure

This procedure is used if a database that has recently changed its role to a primary database following a failover operation fails to record relevant metadata (including the LogMiner dictionary) in the redo stream required for other logical standby databases.

In a CDB, the REBUILD procedure must be called from the root database.

Syntax

DBMS_LOGSTDBY.REBUILD;

Usage Notes

- LogMiner dictionary information is logged in the redo log files. The standby redo log files (if present) are archived.

Examples

SQL> EXECUTE DBMS_LOGSTDBY.REBUILD;

97.4.14 SET_TABLESPACE Procedure

This procedure moves metadata tables required by SQL Apply to the user-specified tablespace.

By default, the metadata tables are created in the SYSAUX tablespace. SQL Apply cannot be running when you invoke this procedure.

In a CDB, the SET_TABLESPACE procedure must be called from the root database.

Syntax

DBMS_LOGSTDBY.SET_TABLESPACE(
    NEW_TABLESPACE IN VARCHAR2)

Parameters

Table 97-20  SET_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_TABLESPACE</td>
<td>Name of the new tablespace where metadata tables will reside.</td>
</tr>
</tbody>
</table>

Exceptions

Table 97-21  SET_TABLESPACE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16236</td>
<td>Logical Standby metadata operation in progress</td>
</tr>
</tbody>
</table>
Examples

To move metadata tables to a new tablespace named `LOGSTDBY_TBS`, issue the following statement:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SET_TABLESPACE (new_tablespace => 'LOGSTDBY_TBS');
```

97.4.15 SKIP Procedure

The `SKIP` procedure can be used to define rules that will be used by SQL Apply to skip the application of certain changes to the logical standby database.

For example, the `SKIP` procedure can be used to skip changes to a subset of tables in the logical standby database. It can also be used to specify DDL statements that should not be applied at the logical standby database or should be modified before they are applied in the logical standby database. One reason why a DDL statement may need to be modified is to accommodate a different directory structure on the logical standby database.

Note:

For information about skipping containers, see "Skipping Containers."

Syntax

```sql
DBMS_LOGSTDBY.SKIP (  
  stmt                      IN VARCHAR2,  
  schema_name               IN VARCHAR2 DEFAULT NULL,  
  object_name               IN VARCHAR2 DEFAULT NULL,  
  proc_name                 IN VARCHAR2 DEFAULT NULL,  
  use_like                  IN BOOLEAN DEFAULT TRUE,  
  esc                       IN CHAR1 DEFAULT NULL);
```

Parameters

Table 97-22  SKIP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt</td>
<td>Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration since keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 97-23 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter. The keyword <code>PL/SQL</code> is used for the execution of Oracle-supplied packages which are supported for replication.</td>
</tr>
<tr>
<td>schema_name</td>
<td>The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the <code>stmt</code> parameter. If not applicable, this value must be set to NULL.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the <code>stmt</code>. If not applicable, this value must be set to NULL.</td>
</tr>
</tbody>
</table>
Table 97-22  (Cont.) SKIP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| proc_name | Name of a stored procedure to call when SQL Apply determines that a particular statement matches the filter defined by the stmt, schema_name, and object_name parameters. Specify the procedure in the following format: 'schema.package.procedure' This procedure returns a value that directs SQL Apply to perform one of the following: execute the statement, skip the statement, or execute a replacement statement. The procedures to be invoked in the case of DDL or PL/SQL take no arguments. You can access the various information needed inside the procedure by accessing the context associated with the namespace, LSBY_APPLYCONTEXT. For a full list of parameters that are accessible in the context of the skip procedure, see the DBMS_LOGSTDBY_CONTEXT package. The parameters of interest in the case of DDLs are: STATEMENT, STATEMENT_TYPE, SCHEMA, NAME, CURRENT_SCHEMA, XIDUSN, XIDSLT, XIDSQN, and SKIP_ACTION. The parameters of interest in the case of PL/SQL are: STATEMENT, PACKAGE_SCHEMA, PACKAGE_NAME, PROCEDURE_NAME, CURRENT_SCHEMA, XIDUSN, XIDSLT, XIDSQN, EXIT_STATUS, and SKIP_ACTION. **Note 1:** The DBMS_LOGSTDBY.SKIP_ACTION_REPLACE constant is not supported for PL/SQL. **Note 2:** SQL Apply calls the skip handler when the procedure’s exit is processed. **Note 3:** The use_like parameter must be set to FALSE for PL/SQL since wildcarding PL/SQL is not supported.
Table 97-22  (Cont.) SKIP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc_name (cont.)</td>
<td>A sample conditional skip rule on DBMS_RLS.DROP_POLICY is as follows:</td>
</tr>
<tr>
<td></td>
<td>create or replace procedure sec.mgr.skip_drop_policy is</td>
</tr>
<tr>
<td></td>
<td>l_stmt  CLOB;</td>
</tr>
<tr>
<td></td>
<td>l_pkgown varchar2(30);</td>
</tr>
<tr>
<td></td>
<td>l_pkgnam varchar2(30);</td>
</tr>
<tr>
<td></td>
<td>l_procnm varchar2(30);</td>
</tr>
<tr>
<td></td>
<td>l_cur_schema varchar2(30);</td>
</tr>
<tr>
<td></td>
<td>l_xidusn number;</td>
</tr>
<tr>
<td></td>
<td>l_xidslt number;</td>
</tr>
<tr>
<td></td>
<td>l_xidsqn number;</td>
</tr>
<tr>
<td></td>
<td>l_exit_status number;</td>
</tr>
<tr>
<td></td>
<td>l_skip_action number;</td>
</tr>
<tr>
<td></td>
<td>Begin</td>
</tr>
<tr>
<td></td>
<td>-- read all relevant info</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'STATEMENT', value =&gt; l_stmt);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'PACKAGE_SCHEMA', value =&gt; l_pkgown);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'PACKAGE_NAME', value =&gt; l_pkgnam);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'PROCEDURE_NAME', value =&gt; l_procnm);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'CURRENT_SCHEMA', value =&gt; l_cur_schema);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'XIDUSN', value =&gt; l_xidusn);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'XIDSLT', value =&gt; l_xidslt);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'XIDSQN', value =&gt; l_xidsqn);</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.get_context(name =&gt; 'EXIT_STATUS', value =&gt; l_ext_status);</td>
</tr>
<tr>
<td></td>
<td>if 0 == l_ext_status then</td>
</tr>
<tr>
<td></td>
<td>Insert Into sec_mgr.logit Values</td>
</tr>
<tr>
<td></td>
<td>('SUCCESS:'</td>
</tr>
<tr>
<td></td>
<td>If l_current_user != 'TESTSCHEMA' Then</td>
</tr>
<tr>
<td></td>
<td>_l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_APPLY;</td>
</tr>
<tr>
<td></td>
<td>Else</td>
</tr>
<tr>
<td></td>
<td>_l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_SKIP;</td>
</tr>
<tr>
<td></td>
<td>End If</td>
</tr>
<tr>
<td></td>
<td>End If</td>
</tr>
<tr>
<td></td>
<td>dbms_logstdby_context.set_context(name=&gt;'SKIP_ACTION', value =&gt; l_skip_action);</td>
</tr>
<tr>
<td></td>
<td>End skip_drop_policy;</td>
</tr>
<tr>
<td></td>
<td>EXECUTE DBMS_LOGSTDBY.SKIP{ -</td>
</tr>
<tr>
<td></td>
<td>stmt =&gt; 'PL/SQL', -</td>
</tr>
<tr>
<td></td>
<td>schema_name =&gt; 'SYS', -</td>
</tr>
<tr>
<td></td>
<td>object_name =&gt; 'DBMS_RLS.DROP_POLICY', -</td>
</tr>
<tr>
<td></td>
<td>proc_name =&gt; 'SEC_MGR.SKIP_DROP_POLICY' -</td>
</tr>
<tr>
<td></td>
<td>use_like=&gt; FALSE);</td>
</tr>
</tbody>
</table>
Table 97-22  (Cont.) SKIP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_like</td>
<td>Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The use_like parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in Oracle Database SQL Language Reference.</td>
</tr>
<tr>
<td>esc</td>
<td>Identifies an escape character (such as the character &quot;/&quot;) that you can use for pattern matching. If the escape character appears in the pattern before the character &quot;%&quot; or &quot;_&quot; then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character. See Oracle Database SQL Language Reference for more information about pattern matching.</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure requires DBA privileges to execute.
- You cannot associate a stored procedure to be invoked in the context of a DML statement. For example, the following statement returns the ORA-16104: invalid Logical Standby option requested error:

  SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
    2  stmt => 'DML', -
    3  schema_name => 'HR', -
    4  object_name => 'EMPLOYEES', -
    5  proc_name => 'DML_HANDLER');

Also, if an event matches multiple rules either because of the use of wildcards while specifying the rule or because of a specification of overlapping rules. For example, if you specify a rule for the SCHEMA_DDL event for the HR.EMPLOYEES table, and a rule for the ALTER TABLE event for the HR.EMPLOYEES table, only one of the matching procedures will be invoked (alphabetically, by procedure). In the following code example, consider the following rules:

  SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
    2  stmt => 'SCHEMA_DDL', -
    3  schema_name => 'HR', -
    4  object_name => 'EMPLOYEES', -
    5  proc_name => 'SCHEMA_DDL_HANDLER');
  SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
    2  stmt => 'ALTER TABLE', -
    3  schema_name => 'HR', -
    4  object_name => 'EMPLOYEES', -
    5  proc_name => 'TABLE_ALTER_HANDLER');

On encountering an ALTER TABLE statement, the schema_ddl_handler procedure will be invoked because its name will be at the top of an alphabetically sorted list of procedures that are relevant to the statement. Collisions on a rule set because of a specification containing wildcard entries are resolved in a similar fashion. For example, the rules in the following example will result in the empddl_handler procedure being invoked upon encountering the ALTER TABLE HR.EMPLOYEES ADD COLUMN RATING NUMBER statement:

  SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
    2  stmt => 'ALTER TABLE', -
    3  schema_name => 'HR', -
    4  object_name => 'EMPLOYEES', -
    5  proc_name => 'TABLE_ALTER_HANDLER');
• Use the `skip` procedure with caution, particularly when skipping DDL statements. If a `CREATE TABLE` statement is skipped, for example, you must also specify other DDL statements that refer to that table in the `skip` procedure. Otherwise, the statements will fail and cause an exception. When this happens, SQL Apply stops running.

• Before calling the `skip` procedure, SQL Apply must be halted. Do this by issuing an `ALTER DATABASE STOP LOGICAL STANDBY APPLY` statement. Once all desired filters have been specified, issue an `ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE` statement to start SQL Apply using the new filter settings.

• See the `unskip` procedure for information about reversing (undoing) the settings of the `skip` procedure.

• For `USER` statements, the `schema_name` parameter will be the user and specify '%' for the `object_name` parameter.

• If the `proc_name` parameter is supplied, it must already exist in `DBA_PROCEDURES` and it must execute with `DEFINER` rights. If the procedure is declared with `INVOKER` rights, the ORA-1031: insufficient privileges message will be returned.

• If the procedure returns a `replacement` statement, the `replacement` statement will be executed using the `system` and `object` privileges of the owner of the procedure.

• The PL/SQL block of a `skip` procedure cannot contain transaction control statements (for example, `commit`, `rollback`, `savepoint`, and `set constraint`) unless the block is declared to be an autonomous transaction.

Skip Statement Options

Table 97-23 lists the supported values for the `stmt` parameter of the `skip` procedure. The left column of the table lists the keywords that may be used to identify the set of SQL statements to the right of the keyword. In addition, any of the SQL statements listed in the `sys.audit_actions` table (shown in the right column of Table 97-23) are also valid values. Note that keywords are generally defined by database object.

Table 97-23  Supported Values for the `stmt` Parameter

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Associated SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is no keyword for this group of SQL statements.</td>
<td><code>GRANT</code> <code>REVOKE</code> <code>ANALYZE TABLE</code> <code>ANALYZE INDEX</code> <code>ANALYZE CLUSTER</code></td>
</tr>
</tbody>
</table>
### Table 97-23  (Cont.) Supported Values for the \textit{stmt} Parameter

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Associated SQL Statements</th>
</tr>
</thead>
</table>
| CLUSTER           | AUDIT CLUSTER  
|                   | CREATE CLUSTER     
|                   | DROP CLUSTER       
|                   | TRUNCATE CLUSTER   |
| CONTAINER         | See "Skipping Containers"                                      |
| CONTEXT           | CREATE CONTEXT     
|                   | DROP CONTEXT       |
| DATABASE LINK     | CREATE DATABASE LINK  
|                   | CREATE PUBLIC DATABASE LINK       
|                   | DROP DATABASE LINK     
|                   | DROP PUBLIC DATABASE LINK    |
| DIMENSION         | ALTER DIMENSION    
|                   | CREATE DIMENSION     
|                   | DROP DIMENSION       |
| DIRECTORY¹        | CREATE DIRECTORY   
|                   | DROP DIRECTORY       |
| DML               | \textit{Includes DML statements on a table (for example: INSERT, UPDATE, and DELETE)} |
| INDEX             | ALTER INDEX        
|                   | CREATE INDEX        
|                   | DROP INDEX          |
| NON_SCHEMA_DDL    | \textit{All DDL that does not pertain to a particular schema}  
| Note: \textit{SCHEMA\_NAME and OBJECT\_NAME must be null} |
| PL/SQL²           | \textit{Execute Oracle-supplied package.}                     |
| PROCEDURE³        | ALTER FUNCTION     
|                   | ALTER PACKAGE      
|                   | ALTER PACKAGE BODY 
|                   | ALTER PROCEDURE    
|                   | CREATE FUNCTION    
|                   | CREATE LIBRARY     
|                   | CREATE PACKAGE     
|                   | CREATE PACKAGE BODY 
|                   | CREATE PROCEDURE   
|                   | DROP FUNCTION      
|                   | DROP LIBRARY       
|                   | DROP PACKAGE       
|                   | DROP PACKAGE BODY  
|                   | DROP PROCEDURE     |
Table 97-23  (Cont.) Supported Values for the stmt Parameter

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Associated SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE</td>
<td>ALTER PROFILE</td>
</tr>
<tr>
<td></td>
<td>CREATE PROFILE</td>
</tr>
<tr>
<td></td>
<td>DROP PROFILE</td>
</tr>
<tr>
<td>ROLE</td>
<td>ALTER ROLE</td>
</tr>
<tr>
<td></td>
<td>CREATE ROLE</td>
</tr>
<tr>
<td></td>
<td>DROP ROLE</td>
</tr>
<tr>
<td></td>
<td>SET ROLE</td>
</tr>
<tr>
<td>ROLLBACK STATEMENT</td>
<td>ALTER ROLLBACK SEGMENT</td>
</tr>
<tr>
<td></td>
<td>CREATE ROLLBACK SEGMENT</td>
</tr>
<tr>
<td></td>
<td>DROP ROLLBACK SEGMENT</td>
</tr>
<tr>
<td>SCHEMA_DDL</td>
<td>All DDL statements that create, modify, or drop schema objects (for example: tables, indexes, and columns)</td>
</tr>
<tr>
<td></td>
<td>Note: SCHEMA_NAME and OBJECT_NAME must not be null</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>ALTER SEQUENCE</td>
</tr>
<tr>
<td></td>
<td>CREATE SEQUENCE</td>
</tr>
<tr>
<td></td>
<td>DROP SEQUENCE</td>
</tr>
<tr>
<td>SYNONYM</td>
<td>CREATE PUBLIC SYNONYM</td>
</tr>
<tr>
<td></td>
<td>CREATE SYNONYM</td>
</tr>
<tr>
<td></td>
<td>DROP PUBLIC SYNONYM</td>
</tr>
<tr>
<td></td>
<td>DROP SYNONYM</td>
</tr>
<tr>
<td>SYSTEM AUDIT</td>
<td>AUDIT SQL_statements</td>
</tr>
<tr>
<td></td>
<td>NOAUDIT SQL_statements</td>
</tr>
<tr>
<td>TABLE</td>
<td>CREATE TABLE</td>
</tr>
<tr>
<td></td>
<td>ALTER TABLE</td>
</tr>
<tr>
<td></td>
<td>DROP TABLE</td>
</tr>
<tr>
<td></td>
<td>TRUNCATE TABLE</td>
</tr>
<tr>
<td>TABLESPACE</td>
<td>CREATE TABLESPACE</td>
</tr>
<tr>
<td></td>
<td>DROP TABLESPACE</td>
</tr>
<tr>
<td></td>
<td>ALTER TABLESPACE</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>ALTER TRIGGER</td>
</tr>
<tr>
<td></td>
<td>CREATE TRIGGER</td>
</tr>
<tr>
<td></td>
<td>DISABLE ALL TRIGGERS</td>
</tr>
<tr>
<td></td>
<td>DISABLE TRIGGER</td>
</tr>
<tr>
<td></td>
<td>DROP TRIGGER</td>
</tr>
<tr>
<td></td>
<td>ENABLE ALL TRIGGERS</td>
</tr>
<tr>
<td></td>
<td>ENABLE TRIGGER</td>
</tr>
</tbody>
</table>
Table 97-23  (Cont.) Supported Values for the stmt Parameter

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Associated SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>ALTER TYPE, ALTER TYPE BODY, CREATE TYPE, CREATE TYPE BODY, DROP TYPE, DROP TYPE BODY</td>
</tr>
<tr>
<td>USER</td>
<td>ALTER USER, CREATE USER, DROP USER</td>
</tr>
<tr>
<td>VIEW</td>
<td>CREATE VIEW, DROP VIEW</td>
</tr>
</tbody>
</table>

1 All directory objects are owned by SYS, but for the purpose of filtering them with a skip directive the schema should be specified as '%'.
2 See Oracle Data Guard Concepts and Administration for information about supported packages.
3 Java schema objects (sources, classes, and resources) are considered the same as procedure for purposes of skipping (ignoring) SQL statements.

Exceptions

Table 97-24  DBMS_LOGSTDBY.SKIP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01031</td>
<td>Insufficient privileges:</td>
</tr>
<tr>
<td></td>
<td>- Procedure used INVOKER rights</td>
</tr>
<tr>
<td></td>
<td>- Procedure needs DBA privileges</td>
</tr>
<tr>
<td>ORA-16103</td>
<td>Logical standby apply must be stopped to allow this operation.</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>Invalid logical standby option requested.</td>
</tr>
<tr>
<td>ORA-16203</td>
<td>&quot;Unable to interpret SKIP procedure return values.&quot; Indicates that a SKIP procedure has either generated an exception or has returned ambiguous values. You can identify the offending procedure by examining the DBA_LOGSTDBY_EVENTS view.</td>
</tr>
<tr>
<td>ORA-16236</td>
<td>Logical standby metadata operation in progress.</td>
</tr>
</tbody>
</table>
Examples

**Example 1. Skipping all DML and DDL changes made to a schema**
The following example shows how to specify rules so that SQL Apply will skip both DDL and DML statements made to the HR schema.

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'SCHEMA_DDL', -
   2   schema_name => 'HR', -
   3   object_name => '%', -
   4   proc_name => null);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'DML', -
   2   schema_name => 'HR', -
   3   object_name => '%', -
   4   proc_name => null);
```

**Example 2. Creating a procedure to handle different file system organization**
For example, if the file system organization in the logical standby database is different than that in the primary database, you can write a **SKIP** procedure to handle DDL statements with file specifications transparently. The following procedure can handle DDL statements as long as you follow a specific naming convention for the file specification string.

1. **Create the **SKIP** procedure to handle tablespace DDL statements:**

```sql
CREATE OR REPLACE PROCEDURE sys.handle_tbs_ddl
IS
  l_old_stmt varchar2(4000);
  l_stmt_typ varchar2(40);
  l_schema   varchar2(30);
  l_name     varchar2(30);
  l_xidusn   number;
  l_xidslt   number;
  l_xidsqn   number;
  l_skip_action number;
  l_new_stmt varchar2(4000);

-- read all information
  dbms_logstdby_context.get_context(name=>'STATEMENT',value=>l_old_stmt);
  dbms_logstdby_context.get_context(name=>'STATEMENT_TYPE',value=>l_stmt_type);
  dbms_logstdby_context.get_context(name=>'OWNER',value=>l_schema);
  dbms_logstdby_context.get_context(name=>'NAME',value=>l_name);
  dbms_logstdby_context.get_context(name=>'XIDUSN',value=>l_xidusn);
  dbms_logstdby_context.get_context(name=>'XIDSLT',value=>l_xidslt);
  dbms_logstdby_context.get_context(name=>'XIDSQN',value=>l_xidsqn);
  dbms_logstdby_context.get_context(name=>'CONTAINER_NAME',value=>l_conname);

--
-- All primary file specification that contains a directory
-- /usr/orcl/primary/dbs
-- should go to /usr/orcl/stdby directory specification
BEGIN
  l_new_stmt := replace (l_old_stmt, '/usr/orcl/primary/dbs','/usr/orcl/stdby');
  l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_REPLACE;
EXCEPTION
  WHEN OTHERS THEN
    l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_ERROR;
    l_new_stmt := NULL;
END;
```

Chapter 97

Summary of DBMS_LOGSTDBY Subprograms

97-29
2. Register the SKIP procedure with SQL Apply:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'TABLESPACE', -
                               proc_name => 'SYS.HANDLE_TBS_DDL');
```

### Skipping Containers

To skip a container (either a PDB or the root), use the CONTAINER keyword. All SQL statements executed on the container, as well as any other actions taken on the container, are skipped.

You can skip a particular PDB within a CDB. For example, the following command skips the PDB named PDB1. The command must be executed at the root level:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'CONTAINER', object_name => 'PDB1');
```

As shown in the following example, you could also skip only the root of the CDB, but not any of the PDBs that exist under the root. The command must be executed at the root level:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'CONTAINER', object_name => 'CDB$ROOT');
```

#### Note:

To create other skip rules for a container, create the rules from within the container. The container to which the rules will apply is automatically derived from the container in which the rules are created.

### 97.4.16 SKIP_ERROR Procedure

The SKIP_ERROR procedure specifies the action to take when a logical standby database detects an error.

Upon encountering an error, the logical standby database uses the criteria contained in this procedure to determine a course of action. The default action when a match is found is to skip the error and continue with applying changes. However, if a procedure is supplied, then SKIP_ERROR can take other actions depending on the situation. It can do nothing, which causes SQL Apply to stop, or it can change the error message text and stop SQL Apply, or it can actually skip the error.

#### Syntax

```sql
DBMS_LOGSTDBY.SKIP_ERROR (  stmt                      IN VARCHAR2,  
                               schema_name               IN VARCHAR2 DEFAULT NULL,  
                               object_name               IN VARCHAR2 DEFAULT NULL,  
                               proc_name                 IN VARCHAR2 DEFAULT NULL,  
                               )
```
Parameters

Table 97-25  SKIP_ERROR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt</td>
<td>Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration because keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 97-23 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.</td>
</tr>
<tr>
<td>schema_name</td>
<td>The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the stmt parameter. If not applicable, this value must be set to NULL.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the stmt. If not applicable, this value must be set to NULL.</td>
</tr>
</tbody>
</table>
| proc_name     | Name of a stored procedure to call when SQL Apply encounters an error and determines a particular statement matches the filter defined by the stmt, schema_name, and object_name parameters. Specify the procedure in the following format: "schema"."package"."procedure"
This procedure returns an error message that directs SQL Apply to perform one of the following actions:
• Silently skip the error and continue with SQL Apply
• Replace the error message that would have been created with a custom one, and stop SQL Apply
• Do nothing, causing SQL Apply to stop and the original error message to be logged
The procedure registered with SQL Apply does not take any parameters. The context associated with LSBY_APPLY_CONTEXT can be used to retrieve all relevant information related to the error. See the DBMS_LOGSTDBY_CONTEXT package for a list of all parameters associated with LSBY_APPLY_CONTEXT.
The parameters of interest for procedures registered with SKIP_ERROR are CONTAINER_NAME, STATEMENT, STATEMENT_TYPE, SCHEMA, NAME, XIDUSN, XIDSLT, XIDSQN, ERROR and NEW_ERROR. |
| use_like      | Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The use_like parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in Oracle Database SQL Language Reference. |
| esc           | Identifies an escape character (such as the characters "%" or "_") that you can use for pattern matching. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character. |
Usage Notes

• A stored procedure provided to the SKIP_ERROR procedure is called when SQL Apply encounters an error that could shut down the application of redo logs to the standby database.

• Running this stored procedure affects the error being written in the STATUS column of the DBA_LOGSTDBY_EVENTS table. The STATUS_CODE column remains unchanged. If the stored procedure is to have no effect, that is, apply will be stopped, then the NEW_ERROR is written to the events table. To truly have no effect, set NEW_ERROR to ERROR in the procedure.

• If the stored procedure requires that a shutdown be avoided, then you must set NEW_ERROR to NULL.

• This procedure requires DBA privileges to execute.

• For USER statements, the SCHEMA_NAME parameter will be the user and you should specify '%' for the OBJECT_NAME parameter.

• If the PROC_NAME parameter is specified, it must already exist in DBA_PROCEDURES and it must execute with DEFINERS rights. If the procedure is declared with INVOKERS rights, the ORA-1031: insufficient privileges message will be returned.

• The PL/SQL block of a SKIP_ERROR procedure cannot contain transaction control statements (for example: COMMIT, ROLLBACK, SAVEPOINT, and SET CONSTRAINT) unless the block is declared to be an autonomous transaction using the following syntax:

```plsql
PRAGMA AUTONOMOUS_TRANSACTION
```

Exceptions

Table 97-26  SKIP_ERROR Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01031</td>
<td>Insufficient privileges:</td>
</tr>
<tr>
<td></td>
<td>• Procedure used INVOKER rights</td>
</tr>
<tr>
<td></td>
<td>• Procedure needs DBA privileges</td>
</tr>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>invalid Logical Standby option requested</td>
</tr>
<tr>
<td>ORA-16236</td>
<td>Logical Standby metadata operation in progress</td>
</tr>
</tbody>
</table>

Example 1

The following example shows how to specify rules so that SQL Apply will skip any error raised from any GRANT DDL command.

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR('GRANT')
```
Example 2

To skip errors on GRANT statements on SYS or HR schemas, define a procedure handle_error_ddl and register it. In the following example, assume that handle_error_ddl is a free-standing procedure in the SYS schema.

1. Create the error-handler procedure:

   ```sql
   CREATE OR REPLACE PROCEDURE sys.handle_error_ddl
   is
     l_stmt    VARCHAR2(4000);
     l_stmt_type VARCHAR2(40);
     l_schema    VARCHAR2(30);
     l_name      VARCHAR2(30);
     l_xidusn      NUMBER;
     l_xidslt      NUMBER;
     l_xidsqn      NUMBER;
     l_error     VARCHAR2(4000);
     l_conname    VARCHAR2(30);
     l_newerr   VARCHAR2(4000);
   BEGIN
     dbms_logstdby_context.get_context(name=>'STATEMENT',value=>l_stmt);
     dbms_logstdby_context.get_context(name=>'STATEMENT_TYPE',value=>l_stmt_type);
     dbms_logstdby_context.get_context(name=>'SCHEMA',value=>l_schema);
     dbms_logstdby_context.get_context(name=>'NAME',value=>l_name);
     dbms_logstdby_context.get_context(name=>'XIDUSN',value=>l_xidusn);
     dbms_logstdby_context.get_context(name=>'XIDSLT',value=>l_xidslt);
     dbms_logstdby_context.get_context(name=>'XIDSQN',value=>l_xidsqn);
     dbms_logstdby_context.get_context(name=>'ERROR',value=>l_error);
     dbms_logstdby_context.get_context(name=>'CONTAINER_NAME',value=>l_conname);

     -- default error to what we already have
     l_new_error := l_error;

     -- Ignore any GRANT errors on SYS or HR schemas
     IF INSTR(UPPER(l_stmt), 'GRANT') > 0
     THEN
       IF l_schema is NULL
         OR (l_schema is NOT NULL AND
             (UPPER(l_schema) = 'SYS' OR
             UPPER(l_schema) = 'HR'))
       THEN
         l_new_error := NULL;
       END IF;
     END IF;

     dbms_logstdby_context.set_context(name => 'NEW_ERROR', value => l_new_error);
   END handle_error_ddl;
   /
   ``

2. Register the error handler with SQL Apply:

   ```sql
   SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR ( -
     1  statement => 'NON_SCHEMA_DDL', -
     2  schema_name => NULL, -
   ```
97.4.17 SKIP_TRANSACTION Procedure

This procedure provides a way to skip (ignore) applying transactions to the logical standby database. You can skip specific transactions by specifying transaction identification information.

Syntax

```sql
DBMS_LOGSTDBY.SKIP_TRANSACTION (
    xidusn          IN NUMBER,
    xidslt          IN NUMBER,
    xidsqn          IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIDUSN</td>
<td>NUMBER</td>
</tr>
<tr>
<td></td>
<td>Transaction ID undo segment number of the transaction being skipped</td>
</tr>
<tr>
<td>XIDSLT</td>
<td>NUMBER</td>
</tr>
<tr>
<td></td>
<td>Transaction ID slot number of the transaction being skipped</td>
</tr>
<tr>
<td>XIDSQN</td>
<td>NUMBER</td>
</tr>
<tr>
<td></td>
<td>Transaction ID sequence number of the transaction being skipped</td>
</tr>
</tbody>
</table>

Usage Notes

If SQL Apply stops due to a particular transaction (for example, a DDL transaction), you can specify that transaction ID and then continue to apply. You can call this procedure multiple times for as many transactions as you want SQL Apply to ignore.

**WARNING:**

SKIP_TRANSACTION is an inherently dangerous operation. Do not invoke this procedure unless you have examined the transaction in question through the V$LOGMNRR_CONTENTS view and have taken compensating actions at the logical standby database. SKIP_TRANSACTION is not the appropriate procedure to invoke to skip DML changes to a table.

To skip a DML failure, use a SKIP procedure, such as SKIP('DML','MySchema','MyFailed Table'). Using the SKIP_TRANSACTION procedure for DML transactions may skip changes for other tables, thus logically corrupting them.

- This procedure requires DBA privileges to execute.
- Use the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by SQL Apply.
Exceptions

Table 97-28   SKIP_TRANSACTION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01031</td>
<td>Need DBA privileges</td>
</tr>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>invalid Logical Standby option requested</td>
</tr>
</tbody>
</table>

Examples

To skip a DDL transaction with \((\text{XIDUSN}, \text{XIDSLT}, \text{XIDSQN})\) of \((1.13.1726)\) you can register a rule as shown in the following example:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_TRANSACTION (-
   2   XIDUSN => 1, XIDSLT => 13, XIDSQN => 1726);
```

97.4.18 UNSKIP Procedure

Use the UNSKIP procedure to delete rules specified earlier with the SKIP procedure.

The parameters specified in the UNSKIP procedure must match exactly for it to delete an already-specified rule.

The `container_name` argument is valid only in a CDB.

Syntax

```
DBMS_LOGSTDBY.UNSKIP (
   stmt                      IN VARCHAR2,
   schema_name               IN VARCHAR2 DEFAULT NULL,
   object_name               IN VARCHAR2 DEFAULT NULL,
   container_name            IN VARCHAR2 DEFAULT NULL);
```

Parameters

The parameter information for the UNSKIP procedure is the same as that described for the SKIP procedure. See Table 97-22 for complete parameter information.

Exceptions

Table 97-29   UNSKIP Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01031</td>
<td>need DBA privileges to execute this procedure</td>
</tr>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>invalid Logical Standby option requested</td>
</tr>
</tbody>
</table>
Usage Notes

**WARNING:**
If DML changes for a table have been skipped and not compensated for, you must follow the call to the UNSKIP procedure with a call to the INSTANTIATE_TABLE procedure to synchronize this table with those maintained by SQL Apply.

- This procedure requires DBA privileges to execute.
- Wildcards passed in the schema_name or the object_name parameter are not expanded. The wildcard character is matched at the character level. Thus, you can delete only one specified rule by invoking the UNSKIP procedure, and you will need a distinct UNSKIP procedure call to delete each rule that was previously specified.

For example, assume you have specified the following two rules to skip applying DML statements to the HR.EMPLOYEE and HR.EMPTEMP tables:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML',-
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMPLOYEE', -
    PROC_NAME => null);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML',-
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMPTEMP', -
    PROC_NAME => null);
```

In the following example, the wildcard in the TABLE_NAME parameter cannot be used to delete the rules that were specified:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP (STMT => 'DML',-
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMP%');
```

In fact, this UNSKIP procedure matches neither of the rules, because the wildcard character in the TABLE_NAME parameter is not expanded. Instead, the wildcard character will be used in an exact match to find the corresponding SKIP rule.

### 97.4.19 UNSKIP_ERROR Procedure

Use the UNSKIP_ERROR procedure to delete rules specified earlier with the SKIP_ERROR procedure.

The parameters specified in the UNSKIP_ERROR procedure must match exactly for the procedure to delete an already-specified rule.

The container_name argument is valid only in a CDB.

**Syntax**

```sql
DBMS_LOGSTDBY.UNSKIP_ERROR (stmt                      IN VARCHAR2,
                             schema_name               IN VARCHAR2 DEFAULT NULL,
                             object_name               IN VARCHAR2 DEFAULT NULL,
                             container_name            IN VARCHAR2 DEFAULT NULL)
```
object_name IN VARCHAR2 DEFAULT NULL,
container_name IN VARCHAR2 DEFAULT NULL);

Parameters

The parameter information for the UNSKIP_ERROR procedure is the same as that described for the SKIP_ERROR procedure. See Table 97-25 for complete parameter information.

Exceptions

Table 97-30  UNSKIP_ERROR Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01031</td>
<td>Need DBA privileges</td>
</tr>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>invalid Logical Standby option requested</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure requires DBA privileges to execute.
- Wildcards passed in the schema_name or the object_name parameters are not expanded. Instead, the wildcard character is treated as any other character and an exact match is made. Thus, you can delete only one specified rule by invoking the UNSKIP_ERROR procedure, and you need a distinct UNSKIP_ERROR procedure call to delete each rule that you previously specified.

For example, assume you have specified the following two rules to handle the HR.EMPLOYEE and HR.EMPTEMP tables:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML',-
   2   SCHEMA_NAME => 'HR', -
   3   OBJECT_NAME => 'EMPLOYEE', -
   4   PROC_NAME => 'hr_employee_handler');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML',-
   2   SCHEMA_NAME => 'HR', -
   3   OBJECT_NAME => 'EMPTEMP', -
   4   PROC_NAME => 'hr_tempemp_handler');
```

In this case, the following UNSKIP procedure cannot be used to delete the rules that you have specified:

```sql
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP_ERROR (STMT => 'DML',-
   2   SCHEMA_NAME => 'HR', -
   3   OBJECT_NAME => 'EMP%');
```

In fact, the UNSKIP procedure will match neither of the rules, because the wildcard character in the OBJECT_NAME parameter will not be expanded.

Example

To remove a handler that was previously registered with SQL Apply from getting called on encountering an error, you can issue the following statement:

```sql
DBMS_LOGSTDBY.UNSKIP_ERROR ( -
   statement => 'NON_SCHEMA_DDL', -
```
97.4.20 UNSKIP_TRANSACTION Procedure

Use the UNSKIP_TRANSACTION procedure to delete rules specified earlier with the SKIP_TRANSACTION procedure.

The parameters specified in the UNSKIP_TRANSACTION procedure must match exactly for the procedure to delete an already-specified rule.

Syntax

```sql
DBMS_LOGSTDBY.UNSKIP_TRANSACTION (    xidusn_p IN NUMBER,    xidslt_p IN NUMBER,    xidsqn_p IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIDUSN</td>
<td>Transaction ID undo segment number of the transaction being skipped</td>
</tr>
<tr>
<td>XIDSLT</td>
<td>Transaction ID slot number of the transaction being skipped</td>
</tr>
<tr>
<td>XIDSQN</td>
<td>Transaction ID sequence number of the transaction being skipped</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-01031</td>
<td>need DBA privileges to execute this procedure</td>
</tr>
<tr>
<td>ORA-16103</td>
<td>Logical Standby apply must be stopped to allow this operation</td>
</tr>
<tr>
<td>ORA-16104</td>
<td>invalid Logical Standby option requested</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure requires DBA privileges to execute.
- Query the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by SQL Apply.

Examples

To remove a rule that was originally specified to skip the application of a transaction with (XIDUSN, XIDSLT, XIDSQN) of (1.13.1726) issue the following statement:

```sql
SQL> DBMS_LOGSTDBY.UNSKIP_TRANSACTION (XIDUSN => 1, XIDSLT => 13, XIDSQN => 1726);
```
As of Oracle Database 12c release 1 (12.1), SQL Apply processes have access to a context namespace called LSBY_APPLY_CONTEXT. You can use the procedures provided in the DBMS_LOGSTDBY_CONTEXT package to set and retrieve various parameters associated with LSBY_APPLY_CONTEXT. This is useful when writing skip procedures that are registered with SQL Apply using the DBMS_LOGSTBDY.SKIP and DBMS_LOGSTBDY.SKIP_ERROR procedures.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_LOGSTDBY_CONTEXT Subprograms

See Also:
Oracle Data Guard Concepts and Administration for more information about SQL Apply and logical standby databases

98.1 DBMS_LOGSTDBY_CONTEXT Overview

SQL Apply processes have access to a context namespace called LSBY_APPLY_CONTEXT.

The following table lists the predefined parameters associated with LSBY_APPLY_CONTEXT that you can set and retrieve by using the procedures provided in the DBMS_LOGSTDBY_CONTEXT package. The ability to set and retrieve the parameters in this way is useful when writing skip procedures that are registered with SQL Apply using the DBMS_LOGSTBDY.SKIP and DBMS_LOGSTBDY.SKIP_ERROR procedures.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT</td>
<td>First 4000 bytes of the statement that the Apply process is processing.</td>
</tr>
<tr>
<td>STATEMENT_TYPE</td>
<td>See Table 97-23 for a list of various statement types.</td>
</tr>
<tr>
<td>PACKAGE_SCHEMA</td>
<td>Schema that owns the PL/SQL package being processed (if applicable).</td>
</tr>
<tr>
<td>PACKAGE_NAME</td>
<td>Name of the PL/SQL package being processed (if applicable).</td>
</tr>
<tr>
<td>PROCEDURE_NAME</td>
<td>Name of the PL/SQL procedure being processed (if applicable).</td>
</tr>
<tr>
<td>CURRENT_SCHEMA</td>
<td>Current schema in effect when the DDL or PL/SQL procedure was executed at the primary.</td>
</tr>
<tr>
<td>XIDUSN</td>
<td>XIDUSN of the transaction at the primary database.</td>
</tr>
</tbody>
</table>
Table 98-1  (Cont.) Predefined Parameters of Namespace LSBY_APPLY_CONTEXT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIDSLT</td>
<td>XIDSLT of the transaction at the primary database.</td>
</tr>
<tr>
<td>XIDSQN</td>
<td>XIDSQN of the transaction at the primary database.</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>Schema that owns the object being processed by SQL Apply (in the case of DML or DDL operations).</td>
</tr>
<tr>
<td>NAME</td>
<td>Object name being processed by SQL Apply (in case of DML or DDL operations).</td>
</tr>
<tr>
<td>CONTAINER_NAME</td>
<td>Container where the target object or the PL/SQL procedure resides (in the case of a multitenant container database (CDB)).</td>
</tr>
<tr>
<td>ERROR</td>
<td>Text of the original error encountered by the SQL Apply process (if applicable).</td>
</tr>
<tr>
<td>NEW_ERROR</td>
<td>Text of the new error to be raised by the SQL Apply process (if applicable, See the DBMS_LOGSTDBY SKIP_ERROR Procedure ).</td>
</tr>
<tr>
<td>NEW_STMT</td>
<td>Text of the replacement statement that SQL Apply must execute (If applicable, See the DBMS_LOGSTDBY SKIP Procedure).</td>
</tr>
<tr>
<td>SKIP_ACTION</td>
<td>The skip action to be performed by SQL Apply (See the DBMS_LOGSTDBY SKIP Procedure).</td>
</tr>
</tbody>
</table>

98.2 DBMS_LOGSTDBY_CONTEXT Security Model

The security model for the DBMS_LOGSTDBY_CONTEXT package is the same as for the DBMS_LOGSTDBY package.

See Also:
DBMS_LOGSTDBY

98.3 Summary of DBMS_LOGSTDBY_CONTEXT Subprograms

This table lists and briefly describes the DBMS_LOGSTDBY_CONTEXT package subprograms.

Table 98-2  DBMS_LOGSTDBY_CONTEXT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR_ALL_CONTEXT</td>
<td>Clears all parameters contained within namespace LSBY_APPLY_CONTEXT.</td>
</tr>
<tr>
<td>CLEAR_CONTEXT</td>
<td>Clears the specific parameter.</td>
</tr>
<tr>
<td>GET_CONTEXT</td>
<td>Retrieves the value for the specified parameter.</td>
</tr>
</tbody>
</table>
Table 98-2  (Cont.) DBMS_LOGSTDBY_CONTEXT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_CONTEXT</td>
<td>Sets the named parameter with the specified value.</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
</tbody>
</table>

98.3.1 CLEAR_ALL_CONTEXT Procedure

This procedure clears all parameters contained within namespace LSBY_APPLY_CONTEXT.

Syntax

DBMS_LOGSTDBY_CONTEXT.CLEAR_ALL_CONTEXT;

Parameters

None

98.3.2 CLEAR_CONTEXT Procedure

This procedure clears the specific parameter.

Syntax

DBMS_LOGSTDBY_CONTEXT.CLEAR_CONTEXT (name IN VARCHAR2);

Parameters

Table 98-3  CLEAR_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the parameter to be cleared.</td>
</tr>
</tbody>
</table>

98.3.3 GET_CONTEXT Procedure

This procedure retrieves the value for the specified parameter.

Syntax

DBMS_LOGSTDBY_CONTEXT.GET_CONTEXT (name IN VARCHAR2, value OUT VARCHAR2);

Parameters

Table 98-4  GET_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the parameter.</td>
</tr>
<tr>
<td>value</td>
<td>The value retrieved for the parameter.</td>
</tr>
</tbody>
</table>
98.3.4 SET_CONTEXT Procedure

This procedure sets the named parameter with the specified value.

Syntax

```
DBMS_LOGSTDBY_CONTEXT.SET_CONTEXT (    
   name       IN VARCHAR2    
   value      IN VARCHAR2);    
```

Parameters

Table 98-5  SET_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the parameter to be set.</td>
</tr>
<tr>
<td>value</td>
<td>Value to be assigned to the parameter being set.</td>
</tr>
</tbody>
</table>
The DBMS_MEMOPTIMIZE package provides an interface for managing data in the memoptimize pool.

This chapter contains the following topics:

- DBMS_MEMOPTIMIZE Overview
- Summary of DBMS_MEMOPTIMIZE Subprograms

### 99.1 DBMS_MEMOPTIMIZE Overview

The DBMS_MEMOPTIMIZE package provides an interface for managing data in the memoptimize pool. The memoptimize pool is a memory area in the system global area (SGA) that stores table data and hash index related to Memoptimized Rowstore.

The DBMS_MEMOPTIMIZE package provides the following functionality:

- Populate the Memoptimized Rowstore hash index with the data related to a specific table.
- Remove the data from the Memoptimized Rowstore hash index related to a specific table.

### 99.2 Summary of DBMS_MEMOPTIMIZE Subprograms

This table lists the DBMS_MEMOPTIMIZE subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP_OBJECT</td>
<td>Removes the in-memory hash index related to a table.</td>
</tr>
<tr>
<td>POPULATE</td>
<td>Populates the in-memory hash index with a table data.</td>
</tr>
</tbody>
</table>

### 99.2.1 DROP_OBJECT Procedure

This procedure removes the in-memory hash index for a table.

**Syntax**

```sql
DBMS_MEMOPTIMIZE.DROP_OBJECT (
    schema_name       IN VARCHAR2,
    table_name        IN VARCHAR2,
    partition_name    IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 99-2 DROP_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table whose data is to be removed from the in-memory hash index.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the table partition whose data is to be removed from the in-memory hash index. This is an optional parameter and its default value is NULL.</td>
</tr>
</tbody>
</table>

99.2.2 POPULATE Procedure

This procedure populates the in-memory hash index for a table.

Syntax

```sql
DBMS_MEMOPTIMIZE.POPULATE (
    schema_name    IN VARCHAR2,
    table_name     IN VARCHAR2,
    partition_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 99-3 POPULATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table from which the data is to be stored into the in-memory hash index.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the table partition from which the data is to be stored into the in-memory hash index. This is an optional parameter and its default value is NULL.</td>
</tr>
</tbody>
</table>
The **DBMS_METADATA** package provides a way for you to retrieve metadata from the database dictionary as XML or creation DDL and to submit the XML to re-create the object.

### See Also:

*Oracle Database Utilities* for more information and for examples of using the `DBMS_METADATA` package.

This chapter contains the following topics:

- Overview
- Security Model
- Rules and Limits
- Data Structures - Object and Table Types
- Subprogram Groupings
  - Subprograms for Retrieving Multiple Objects From the Database
  - Subprograms for Submitting XML to the Database
- Summary of All DBMS_METADATA Subprograms

### 100.1 DBMS_METADATA Overview

You can use the `DBMS_METADATA` package to retrieve metadata and to submit XML.

**Retrieving Metadata**

If you are retrieving metadata, you can specify:

- The kind of object to be retrieved. This can be either a particular object type (such as a table, index, or procedure) or a heterogeneous collection of object types that form a logical unit (such as a database export or schema export).
- Optional selection criteria, such as owner or name.
- Parse items (attributes of the returned objects to be parsed and returned separately).
- Optional transformations on the output, implemented by XSLT (Extensible Stylesheet Language Transformation) scripts. By default the output is represented in XML, but you can specify transformations (into SQL DDL, for example), which are implemented by XSLT stylesheets stored in the database or externally.

`DBMS_METADATA` provides the following retrieval interfaces:
• For programmatic use: OPEN, SET_FILTER, SET_COUNT, GET_QUERY, SET_PARSE_ITEM, ADD_TRANSFORM, SET_TRANSFORM_PARAM, SET_REMAP_PARAM, FETCH_xxx, and CLOSE retrieve multiple objects.

• For use in SQL queries and for browsing: GET_XML, GET_DDL and GET_SXML return metadata for a single named object. The GET_DEPENDENT_XML, GET_DEPENDENT_DDL, GET_GRANTED_XML, and GET_GRANTED_DDL interfaces return metadata for one or more dependent or granted objects. These procedures do not support heterogeneous object types.

 Submitting XML

If you are submitting XML, you specify:

• The type of object
• Optional transform parameters to modify the object (for example, changing the object's owner)
• Parse items (attributes of the submitted objects to be parsed and submitted separately)
• Whether to execute the operation or simply return the generated DDL

DBMS_METADATA provides a programmatic interface for submission of XML. It is comprised of the following procedures: OPEN, ADD_TRANSFORM, SET_TRANSFORM_PARAM, SET_REMAP_PARAM, SET_PARSE_ITEM, CONVERT, PUT, and CLOSE.

100.2 DBMS_METADATA Security Model

The DBMS_METADATA package considers a privileged user to be one who is connected as user SYS or who has the SELECT_CATALOG_ROLE role.

The object views of the Oracle metadata model implement security as follows:

• Nonprivileged users can see the metadata of only their own objects.
• Nonprivileged users can also retrieve public synonyms, system privileges granted to them, and object privileges granted to them or by them to others. This also includes privileges granted to PUBLIC.
• If callers request objects they are not privileged to retrieve, no exception is raised; the object is simply not retrieved.
• If nonprivileged users are granted some form of access to an object in someone else's schema, they will be able to retrieve the grant specification through the Metadata API, but not the object's actual metadata.
• In stored procedures, functions, and definers-rights packages, roles (such as SELECT_CATALOG_ROLE) are disabled. Therefore, such a PL/SQL program can only fetch metadata for objects in its own schema. If you want to write a PL/SQL program that fetches metadata for objects in a different schema (based on the invoker's possession of SELECT_CATALOG_ROLE), you must make the program invokers-rights.
• For all objects that have passwords, except database links (for example, users and roles), the following rules apply:
  – A user who has the SELECT_CATALOG_ROLE can see all metadata for an object except the passwords for that object.
The SYS user, users who have the EXP_FULL_DATABASE role, and users who own an object can see all metadata for that object, including passwords.

- For database links the password is never displayed. For security reasons Oracle restricts visibility of the password value to SYS users who query the link$.passwordx column directly. Instead of the password, DBMS_METADATA returns the following invalid syntax:

  IDENTIFIED BY VALUES ':1'

A user who knows the password of the database link can manually replace the :1 with the password.

100.3 Rules and Limits

In an Oracle Shared Server (OSS) environment, the DBMS_METADATA package must disable session migration and connection pooling.

This results in any shared server process that is serving a session running the package to effectively become a default, dedicated server for the life of the session. You should ensure that sufficient shared servers are configured when the package is used and that the number of servers is not artificially limited by too small a value for the MAX_SHARED_SERVERS initialization parameter.

100.4 DBMS_METADATA Data Structures - Object and Table Types

The DBMS_METADATA package defines, in the SYS schema, the OBJECT and TABLE types shown in this code.

```sql
CREATE TYPE sys.ku$_parsed_item AS OBJECT (
  item VARCHAR2(30),
  value VARCHAR2(4000),
  object_row NUMBER ) /
CREATE PUBLIC SYNONYM ku$_parsed_item FOR sys.ku$_parsed_item;
CREATE TYPE sys.ku$_parsed_items IS TABLE OF sys.ku$_parsed_item /
CREATE PUBLIC SYNONYM ku$_parsed_items FOR sys.ku$_parsed_items;
CREATE TYPE sys.ku$_ddl AS OBJECT (
  ddlText CLOB,
  parsedItems sys.ku$_parsed_items ) /
CREATE PUBLIC SYNONYM ku$_ddl FOR sys.ku$_ddl;
CREATE TYPE sys.ku$_ddls IS TABLE OF sys.ku$_ddl /
CREATE PUBLIC SYNONYM ku$_ddls FOR sys.ku$_ddls;
CREATE TYPE sys.ku$_multi_ddl AS OBJECT (
  object_row NUMBER,
  ddls sys.ku$_ddls )
```
CREATE OR REPLACE PUBLIC SYNONYM ku$ multi ddl FOR sys.ku$ multi ddl;

CREATE TYPE sys.ku$ multi ddl s IS TABLE OF sys.ku$ multi ddl;

CREATE OR REPLACE PUBLIC SYNONYM ku$ multi ddls FOR sys.ku$ multi ddls;

CREATE TYPE sys.ku$ ErrorLine IS OBJECT (
    errorNumber NUMBER,
    errorText VARCHAR2(2000)
);

CREATE PUBLIC SYNONYM ku$ ErrorLine FOR sys.ku$ ErrorLine;

CREATE TYPE sys.ku$ ErrorLines IS TABLE OF sys.ku$ ErrorLine;

CREATE PUBLIC SYNONYM ku$ErrorLines FOR sys.ku$ ErrorLines;

CREATE TYPE sys.ku$ SubmitResult AS OBJECT (
    ddl sys.ku$ ddl,
    errorLines sys.ku$ ErrorLines);

CREATE TYPE sys.ku$ SubmitResults IS TABLE OF sys.ku$ SubmitResult;

CREATE PUBLIC SYNONYM ku$ SubmitResults FOR sys.ku$ SubmitResults;

Note:

The maximum size of the VARCHAR2, NVARCHAR2, and RAW datatypes has been increased to 32 KB when the COMPATIBLE initialization parameter is set to 12.0 and the MAX_STRING_SIZE initialization parameter is set to EXTENDED. The DBMS_METADATA package supports this increased size unless the version of the metadata is earlier than Oracle Database 12c Release 1 (12.1).

100.5 DBMS_METADATA Subprogram Groupings

The DBMS_METADATA subprograms retrieve objects from, and submit XML to, a database. Some subprograms are used for both activities, while others are used only for retrieval or only for submission.

- **Table 100-1** provides a summary, in alphabetical order, of DBMS_METADATA subprograms used to retrieve multiple objects from a database.
- **Table 100-2** provides a summary, in alphabetical order, of DBMS_METADATA subprograms used to submit XML metadata to a database.
100.5.1 DBMS_METADATA Subprograms for Retrieving Multiple Objects From the Database

DBMS_METADATA uses these subprograms used for retrieving multiple objects from the database.

Table 100-1  DBMS_METADATA Subprograms for Retrieving Multiple Objects

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TRANSFORM Function</td>
<td>Specifies a transform that FETCH_xxx applies to the XML representation of the retrieved objects</td>
</tr>
<tr>
<td>CLOSE Procedure</td>
<td>Invalidates the handle returned by OPEN and cleans up the associated state</td>
</tr>
<tr>
<td>FETCH_xxx Functions and Procedures</td>
<td>Returns metadata for objects meeting the criteria established by OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on</td>
</tr>
<tr>
<td>GET_QUERY Function</td>
<td>Returns the text of the queries that are used by FETCH_xxx</td>
</tr>
<tr>
<td>GET_xxx Functions</td>
<td>Fetches the metadata for a specified object as XML, SXML, or DDL, using only a single call</td>
</tr>
<tr>
<td>OPEN Function</td>
<td>Specifies the type of object to be retrieved, the version of its metadata, and the object model</td>
</tr>
<tr>
<td>SET_COUNT Procedure</td>
<td>Specifies the maximum number of objects to be retrieved in a single FETCH_xxx call</td>
</tr>
<tr>
<td>SET_FILTER Procedure</td>
<td>Specifies restrictions on the objects to be retrieved, for example, the object name or schema</td>
</tr>
<tr>
<td>SET_PARSE_ITEM Procedure</td>
<td>Enables output parsing by specifying an object attribute to be parsed and returned</td>
</tr>
<tr>
<td>SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures</td>
<td>Specifies parameters to the XSLT stylesheets identified by transform_handle</td>
</tr>
</tbody>
</table>

100.5.2 DBMS_METADATA Subprograms for Submitting XML to the Database

DBMS_METADATA uses these subprograms for submitting XML to the database.

Table 100-2  DBMS_METADATA Subprograms for Submitting XML

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TRANSFORM Function</td>
<td>Specifies a transform for the XML documents</td>
</tr>
<tr>
<td>CLOSE Procedure</td>
<td>Closes the context opened with OPENNW</td>
</tr>
<tr>
<td>CONVERT Functions and Procedures</td>
<td>Converts an XML document to DDL</td>
</tr>
<tr>
<td>OPENNW Function</td>
<td>Opens a write context</td>
</tr>
<tr>
<td>PUT Function</td>
<td>Submits an XML document to the database</td>
</tr>
<tr>
<td>SET_PARSE_ITEM Procedure</td>
<td>Specifies an object attribute to be parsed</td>
</tr>
</tbody>
</table>
### 100.6 Summary of All DBMS_METADATA Subprograms

This table lists the DBMS_METADATA subprograms and briefly describes them.

#### Table 100-3 DBMS_METADATA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TRANSFORM Function</td>
<td>Specifies a transform that <code>FETCH_xxx</code> applies to the XML representation of retrieved objects</td>
</tr>
<tr>
<td>CLOSE Procedure</td>
<td>Invalidates the handle returned by <code>OPEN</code> and cleans up the associated state</td>
</tr>
<tr>
<td>CONVERT Functions and Procedures</td>
<td>Converts an XML document to DDL</td>
</tr>
<tr>
<td>FETCH_xxx Functions and Procedures</td>
<td>Returns metadata for objects meeting the criteria established by <code>OPEN</code>, <code>SET_FILTER</code>, <code>SET_COUNT</code>, <code>ADD_TRANSFORM</code>, and so on</td>
</tr>
<tr>
<td>GET_xxx Functions</td>
<td>Fetches the metadata for a specified object as XML, SXML, or DDL, using only a single call</td>
</tr>
<tr>
<td>GET_QUERY Function</td>
<td>Returns the text of the queries that are used by <code>FETCH_xxx</code></td>
</tr>
<tr>
<td>OPEN Function</td>
<td>Specifies the type of object to be retrieved, the version of its metadata, and the object model</td>
</tr>
<tr>
<td>OPENW Function</td>
<td>Opens a write context</td>
</tr>
<tr>
<td>PUT Function</td>
<td>Submits an XML document to the database</td>
</tr>
<tr>
<td>SET_COUNT Procedure</td>
<td>Specifies the maximum number of objects to be retrieved in a single <code>FETCH_xxx</code> call</td>
</tr>
<tr>
<td>SET_FILTER Procedure</td>
<td>Specifies restrictions on the objects to be retrieved, for example, the object name or schema</td>
</tr>
<tr>
<td>SET_PARSE_ITEM Procedure</td>
<td>Enables output parsing by specifying an object attribute to be parsed and returned</td>
</tr>
<tr>
<td>SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures</td>
<td>Specifies parameters to the XSLT stylesheets identified by <code>transform_handle</code></td>
</tr>
</tbody>
</table>

#### 100.6.1 ADD_TRANSFORM Function

The DBMS_METADATA.ADD_TRANSFORM function is used for both retrieval and submission.

- When this procedure is used to retrieve objects, it specifies a transform that `FETCH_xxx` applies to the XML representation of the retrieved objects.
• When used to submit objects, it specifies a transform that CONVERT or PUT applies to the XML representation of the submitted objects. It is possible to add more than one transform.

**See Also:**
- Subprograms for Retrieving Multiple Objects From the Database
- Subprograms for Submitting XML to the Database
- “SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures” for information about how to modify and customize transform output

**Syntax**

```sql
DBMS_METADATA.ADD_TRANSFORM (
    handle       IN NUMBER,
    name         IN VARCHAR2,
    encoding     IN VARCHAR2 DEFAULT NULL,
    object_type  IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

**Parameters**

**Table 100-4  ADD_TRANSFORM Function Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN when this transform is used to retrieve objects. Or the handle returned from OPENW when this transform is used in the submission of XML metadata.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the transform. The name can be an internal keyword like DDL to use internally stored stylesheets. If the name contains a colon, it is interpreted as directory_object_name:file_name of a user-supplied Extensible Stylesheet Language Transformation (XSLT) script. Otherwise, name designates a transform implemented by DBMS_METADATA. See Table 100-5 for descriptions of available transforms.</td>
</tr>
<tr>
<td>encoding</td>
<td>The name of the Globalization Support character set in which the stylesheet pointed to by name is encoded. This is only valid if name is a URL. If left NULL and the URL is external to the database, UTF-8 encoding is assumed. If left NULL and the URL is internal to the database (that is, it begins with /oradb/), then the encoding is assumed to be the database character set.</td>
</tr>
</tbody>
</table>
Table 100-4  (Cont.) ADD_TRANSFORM Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The definition of this parameter depends upon whether you are retrieving objects or submitting XML metadata.</td>
</tr>
</tbody>
</table>

1. When you use ADD_TRANSFORM to retrieve objects, the following definition of object_type applies:

   Designates the object type to which the transform applies. (Note that this is an object type name, not a path name.) By default the transform applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, the following behavior can occur:
   - if object_type is omitted, the transform applies to all object types within the heterogeneous collection
   - if object_type is specified, the transform only applies to that specific object type within the collection
   
   If you omit this parameter you can add the DDL transform to all objects in a heterogeneous collection with a single call. If you supply this parameter, you can add a transform for a specific object type.

2. When you use ADD_TRANSFORM in the submission of XML metadata, this parameter is the object type to which the transform applies. By default, it is the object type of the OPENW handle. Because the OPENW handle cannot designate a heterogeneous object type, the caller would normally leave this parameter NULL in the ADD_TRANSFORM calls.

The following table describes the transforms available on the ADD_TRANSFORM function.

Because new transforms are occasionally added, you might want to query the DBMS_METADATA_TRANSFORMS view to see all valid Oracle-supplied transforms for specific object types.

Table 100-5  Transforms Available on ADD_TRANSFORM Function

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Transform Name</th>
<th>Input Doc Type</th>
<th>Output Doc Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>DDL</td>
<td>XML</td>
<td>DDL</td>
<td>Convert XML to SQL to create the object</td>
</tr>
<tr>
<td>All</td>
<td>MODIFY</td>
<td>XML</td>
<td>XML</td>
<td>Modify XML document according to transform parameters</td>
</tr>
<tr>
<td>Subset</td>
<td>SXML</td>
<td>XML</td>
<td>SXML</td>
<td>Convert XML to SXML</td>
</tr>
<tr>
<td>Subset</td>
<td>MODIFYSXML</td>
<td>SXML</td>
<td>SXML</td>
<td>Modify SXML document according to transform parameters</td>
</tr>
<tr>
<td>Subset</td>
<td>SXMLDDL</td>
<td>SXML</td>
<td>DDL</td>
<td>Convert SXML to DDL</td>
</tr>
</tbody>
</table>
### Table 100-5  (Cont.) Transforms Available on ADD_TRANSFORM Function

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Transform Name</th>
<th>Input Doc Type</th>
<th>Output Doc Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset</td>
<td>ALTERXML</td>
<td>SXML difference</td>
<td>ALTER_XML</td>
<td>Generate ALTER_XML from SXML difference document. (See the DBMS_METADATA_DIFF PL/SQL package for more information about SXML difference format.) The following parameters are valid for the ALTERXML transform:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>document</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALTERDDL</td>
<td>ALTER_XML</td>
<td>ALTER_DDL</td>
<td>Convert ALTER_XML to ALTER_DDL</td>
</tr>
</tbody>
</table>

### Return Values

The opaque handle that is returned is used as input to SET_TRANSFORM_PARAM and SET_REMAP_PARAM. Note that this handle is different from the handle returned by OPEN or OPENW; it refers to the transform, not the set of objects to be retrieved.

### Usage Notes

- With no transforms added, objects are returned by default as XML documents. You call ADD_TRANSFORM to specify the XSLT stylesheets to be used to transform the returned XML documents.
- You can call ADD_TRANSFORM more than once to apply multiple transforms to XML documents. The transforms are applied in the order in which they were specified, the output of the first transform being used as input to the second, and so on.
- The output of a DDL transform is not an XML document. Therefore, no transform should be added after the DDL transform.
- Each transform expects a certain format XML document as input. If the input document is unspecified, metadata XML format is assumed.
• When the ALTERXML transform is used, parse items are returned in a PARSE_LIST element of the ALTER_XML document. Each PARSE_LIST_ITEM element contains an ITEM and a VALUE. For example:

```xml
<PARSE_LIST>
  <PARSE_LIST_ITEM>
    <ITEM>XPATH</ITEM>
  </PARSE_LIST_ITEM>
  <PARSE_LIST_ITEM>
    <ITEM>NAME</ITEM>
    <VALUE>Z1</VALUE>
  </PARSE_LIST_ITEM>
  <PARSE_LIST_ITEM>
    <ITEM>CLAUSE_TYPE</ITEM>
    <VALUE>ADD_COLUMN</VALUE>
  </PARSE_LIST_ITEM>
  <PARSE_LIST_ITEM>
    <ITEM>COLUMN_ATTRIBUTE</ITEM>
    <VALUE>NOT_NULL</VALUE>
  </PARSE_LIST_ITEM>
</PARSE_LIST>
```

Exceptions

• INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.

• INVALID_OPERATION. ADD_TRANSFORM was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to ADD_TRANSFORM for the current OPEN context are permitted.

• INCONSISTENT_ARGS. The arguments are inconsistent. Possible inconsistencies include the following:
  – encoding is specified even though name is not a URL.
  – object_type is not part of the collection designated by handle.

100.6.2 CLOSE Procedure

This procedure is used for both retrieval and submission. This procedure invalidates the handle returned by OPEN (or OPENW) and cleans up the associated state.

See Also:

For more information about related subprograms:

• Subprograms for Retrieving Multiple Objects From the Database
• Subprograms for Submitting XML to the Database

Syntax

```sql
DBMS_METADATA.CLOSE ( handle IN NUMBER);
```
Parameters

Table 100-6  CLOSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN (or OPENW).</td>
</tr>
</tbody>
</table>

Usage Notes

Note:

The following notes apply only to object retrieval

You can prematurely terminate the stream of objects established by OPEN or (OPENW).

- If a call to FETCH_xxx returns NULL, indicating no more objects, a call to CLOSE is made transparently. In this case, you can still call CLOSE on the handle and not get an exception. (The call to CLOSE is not required.)
- If you know that only one specific object will be returned, you should explicitly call CLOSE after the single FETCH_xxx call to free resources held by the handle.

Exceptions

- INVALID_ARGVAL. The value for the handle parameter is NULL or invalid.

100.6.3 CONVERT Functions and Procedures

The DBMS_METADATA.CONVERT functions and procedures transform input XML documents.

The CONVERT functions return creation DDL. The CONVERT procedures return either XML or DDL, depending on the specified transforms.

See Also:

- Subprograms for Submitting XML to the Database

Syntax

The CONVERT functions are as follows:

```sql
DBMS_METADATA.CONVERT (  
    handle   IN NUMBER,  
    document IN sys.XMLType)  
RETURN sys.kr$_multi_ddls;

DBMS_METADATA.CONVERT (  
    handle   IN NUMBER,
```
The `CONVERT` procedures are as follows:

```sql
DBMS_METADATA.CONVERT (  
  handle   IN NUMBER,  
  document IN sys.XMLType,  
  result   IN OUT NOCOPY CLOB);

DBMS_METADATA.CONVERT (  
  handle   IN NUMBER,  
  document IN CLOB,  
  result   IN OUT NOCOPY CLOB);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from <code>OPENW</code></td>
</tr>
<tr>
<td>document</td>
<td>The XML document containing object metadata of the type of the <code>OPENW</code> handle</td>
</tr>
<tr>
<td>result</td>
<td>The converted document</td>
</tr>
</tbody>
</table>

### Return Values

Either XML or DDL, depending on the specified transforms.

### Usage Notes

You can think of `CONVERT` as the second half of `FETCH_xxx`, either `FETCH_DDL` (for the function variants) or `FETCH_CLOB` (for the procedure variants). There are two differences:

- `FETCH_xxx` gets its XML document from the database, but `CONVERT` gets its XML document from the caller
- `FETCH_DDL` returns its results in a `sys.ku$_ddls` nested table, but `CONVERT` returns a `sys.ku$_multi_ddls` nested table

The transforms specified with `ADD_TRANSFORM` are applied in turn, and the result is returned to the caller. For the function variants, the DDL transform must be specified. If parse items were specified, they are returned in the `parsedItems` column. Parse items are ignored by the procedure variants.

The encoding of the XML document is embedded in its CLOB or XMLType representation. The version of the metadata is embedded in the XML. The generated DDL is valid for the database version specified in `OPENW`.

### Exceptions

- `INVALID_ARGVAL`. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INCONSISTENT_OPERATION`. No transform was specified. The DDL transform was not specified (function variants only).
100.6.4 FETCH_xxx Functions and Procedures

These functions and procedures return metadata for objects meeting the criteria established by OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on.

See "Usage Notes" for the variants.

**See Also:**

For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database

### Syntax

The **FETCH** functions are as follows:

```sql
DBMS_METADATA.FETCH_XML (handle IN NUMBER) RETURN sys.XMLType;

DBMS_METADATA.FETCH_DDL (handle IN NUMBER) RETURN sys.ku$_ddls;

DBMS_METADATA.FETCH_CLOB (handle IN NUMBER, cache_lob IN BOOLEAN DEFAULT TRUE, lob_duration IN PLS INTEGER DEFAULT DBMS_LOB.SESSION) RETURN CLOB;
```

The **FETCH** procedures are as follows:

```sql
DBMS_METADATA.FETCH_CLOB (handle IN NUMBER, doc IN OUT NOCOPY CLOB);

DBMS_METADATA.FETCH_XML_CLOB (handle IN NUMBER, doc IN OUT NOCOPY CLOB, parsed_items OUT sys.ku$_parsed_items, object_type_path OUT VARCHAR2);
```

### Parameters

**Table 100-8**  **FETCH_xxx Function Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>cache_lob</td>
<td>TRUE=read LOB into buffer cache</td>
</tr>
</tbody>
</table>
Table 100-8  (Cont.) FETCH_xxx Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_duration</td>
<td>The duration for the temporary LOB created by FETCH_CLOB, either DBMS_LOB.SESSION (the default) or DBMS_LOB.CALL.</td>
</tr>
<tr>
<td>doc</td>
<td>The metadata for the objects, or NULL if all objects have been returned.</td>
</tr>
<tr>
<td>parsed_items</td>
<td>A nested table containing the items specified by SET_PARSE_ITEM. If SET_PARSE_ITEM was not called, a NULL is returned.</td>
</tr>
<tr>
<td>object_type_path</td>
<td>For heterogeneous object types, this is the full path name of the object type for the objects returned by the call to FETCH_XXX. If handle designates a homogeneous object type, a NULL is returned.</td>
</tr>
</tbody>
</table>

Return Values

The metadata for the objects or NULL if all objects have been returned.

Usage Notes

These functions and procedures return metadata for objects meeting the criteria established by the call to OPEN that returned the handle, and subsequent calls to SET_FILTER, SET_COUNT, ADD_TRANSFORM, and so on. Each call to FETCH_XXX returns the number of objects specified by SET_COUNT (or less, if fewer objects remain in the underlying cursor) until all objects have been returned. After the last object is returned, subsequent calls to FETCH_XXX return NULL and cause the stream created by OPEN to be transparently closed.

There are several different FETCH_xxx functions and procedures:

- The FETCH_XML function returns the XML metadata for an object as an XMLType. It assumes that if any transform has been specified, that transform will produce an XML document. In particular, it assumes that the DDL transform has not been specified.
- The FETCH_DDL function returns the DDL (to create the object) in a sys.ku$_ddls nested table. It assumes that the DDL transform has been specified. Each row of the sys.ku$_ddls nested table contains a single DDL statement in the ddlText column; if requested, parsed items for the DDL statement will be returned in the parsedItems column. Multiple DDL statements may be returned under the following circumstances:
  - When you call SET_COUNT to specify a count greater than 1
  - When an object is transformed into multiple DDL statements. For example, A TYPE object that has a DDL transform applied to it can be transformed into both CREATE TYPE and CREATE TYPE BODY statements. A TABLE object can be transformed into a CREATE TABLE, and one or more ALTER TABLE statements
- The FETCH_CLOB function simply returns the object, transformed or not, as a CLOB. By default, the CLOB is read into the buffer cache and has session duration, but these defaults can be overridden with the cache_lob and lob_duration parameters.
The `FETCH_CLOB` procedure returns the objects by reference in an `IN OUT NOCOPY` parameter. This is faster than the function variant, which returns LOBs by value, a practice that involves an expensive LOB copy.

The `FETCH_XML_CLOB` procedure returns the XML metadata for the objects as a CLOB in an `IN OUT NOCOPY` parameter. This helps to avoid LOB copies, which can consume a lot of resources. It also returns a nested table of parse items and the full path name of the object type of the returned objects.

All LOBs returned by `FETCH_xxx` are temporary LOBs. You must free the LOB. If the LOB is supplied as an `IN OUT NOCOPY` parameter, you must also create the LOB.

If `SET_PARSE_ITEM` was called, `FETCH_DDL` and `FETCH_XML_CLOB` return attributes of the object's metadata (or the DDL statement) in a `sys.ku$_parsed_items` nested table. For `FETCH_XML_CLOB`, the nested table is an `OUT` parameter. For `FETCH_DDL`, it is a column in the returned `sys.ku$_dlds` nested table. Each row of the nested table corresponds to an item specified by `SET_PARSE_ITEM` and contains the following columns:
- `item`—the name of the attribute as specified in the `name` parameter to `SET_PARSE_ITEM`.
- `value`—the attribute value, or `NULL` if the attribute is not present in the DDL statement.
- `object-row`—a positive integer indicating the object to which the parse item applies. If multiple objects are returned by `FETCH_xxx`, (because `SET_COUNT` specified a count greater than 1) then `object_row=1` for all items for the first object, 2 for the second, and so on.

The rows of the `sys.ku$_parsed_items` nested table are ordered by ascending `object_row`, but otherwise the row order is undetermined. To find a particular parse item within an object row the caller must search the table for a match on `item`.

In general there is no guarantee that a requested parse item will be returned. For example, the parse item may not apply to the object type or to the particular line of DDL, or the item's value may be `NULL`.

If `SET_PARSE_ITEM` was not called, `NULL` is returned as the value of the parsed items nested table.

It is expected that the same variant of `FETCH_xxx` will be called for all objects selected by `OPEN`. That is, programs will not intermix calls to `FETCH_XML`, `FETCH_DDL`, `FETCH_CLOB`, and so on using the same `OPEN` handle. The effect of calling different variants is undefined; it might do what you expect, but there are no guarantees.

Every object fetched will be internally consistent with respect to on-going DDL (and the subsequent recursive DML) operations against the dictionary. In some cases, multiple queries may be issued, either because the object type is heterogeneous or for performance reasons (for example, one query for heap tables, one for index-organized tables). Consequently the `FETCH_xxx` calls may in fact be fetches from different underlying cursors (meaning that read consistency is not guaranteed).

**Exceptions**

Most exceptions raised during execution of the query are propagated to the caller. Also, the following exceptions may be raised:

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.
• **INCONSISTENT_OPERATION.** Either `FETCH_XML` was called when the DDL transform had been specified, or `FETCH_DDL` was called when the DDL transform had *not* been specified.

### 100.6.5 GET_xxx Functions

GET_xxx functions let you fetch metadata for objects with a single call.

These GET_xxx functions are:

- GET_XML
- GET_DDL
- GET_SXML
- GET_DEPENDENT_XML
- GET_DEPENDENT_DDL
- GET_GRANTED_XML
- GET_GRANTED_DDL

#### See Also:

For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database

#### Syntax

```sql
DBMS_METADATA.GET_XML (object_type IN VARCHAR2, 
                        name IN VARCHAR2, 
                        schema IN VARCHAR2 DEFAULT NULL, 
                        version IN VARCHAR2 DEFAULT 'COMPATIBLE', 
                        model IN VARCHAR2 DEFAULT 'ORACLE', 
                        transform IN VARCHAR2 DEFAULT NULL) 
RETURN CLOB;
```

```sql
DBMS_METADATA.GET_DDL (object_type IN VARCHAR2, 
                        name IN VARCHAR2, 
                        schema IN VARCHAR2 DEFAULT NULL, 
                        version IN VARCHAR2 DEFAULT 'COMPATIBLE', 
                        model IN VARCHAR2 DEFAULT 'ORACLE', 
                        transform IN VARCHAR2 DEFAULT 'DDL') 
RETURN CLOB;
```

```sql
DBMS_METADATA.GET_SXML (object_type IN VARCHAR2, 
                        name IN VARCHAR2 DEFAULT NULL, 
                        schema IN VARCHAR2 DEFAULT NULL, 
                        version IN VARCHAR2 DEFAULT 'COMPATIBLE', 
                        model IN VARCHAR2 DEFAULT 'ORACLE', 
                        transform IN VARCHAR2 DEFAULT 'SXML') 
RETURN CLOB;
```
DBMS_METADATA.GET_DEPENDENT_XML (object_type IN VARCHAR2, base_object_name IN VARCHAR2, base_object_schema IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT NULL, object_count IN NUMBER DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_DEPENDENT_DDL (object_type IN VARCHAR2, base_object_name IN VARCHAR2, base_object_schema IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT 'DDL', object_count IN NUMBER DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_GRANTED_XML (object_type IN VARCHAR2, grantee IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT NULL, object_count IN NUMBER DEFAULT 10000)
RETURN CLOB;

DBMS_METADATA.GET_GRANTED_DDL (object_type IN VARCHAR2, grantee IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT 'DDL', object_count IN NUMBER DEFAULT 10000)
RETURN CLOB;

Parameters

Table 100-9  GET_xxx Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The type of object to be retrieved. This parameter takes the same values as the OPEN object_type parameter, except that it cannot be a heterogeneous object type. The attributes of the object type must be appropriate to the function. That is, for GET_xxx it must be a named object.</td>
</tr>
<tr>
<td>name</td>
<td>The object name. It is used internally in a NAME filter. (If the name is longer than 30 characters, it will be used in a LONGNAME filter.) If this parameter is NULL, then no NAME or LONGNAME filter is specified. See Table 100-18 for a list of filters.</td>
</tr>
<tr>
<td>schema</td>
<td>The object schema. It is used internally in a SCHEMA filter. The default is the current user.</td>
</tr>
<tr>
<td>version</td>
<td>The version of metadata to be extracted. This parameter takes the same values as the OPEN version parameter.</td>
</tr>
</tbody>
</table>
Table 100-9  (Cont.) GET_xxx Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>The object model to use. This parameter takes the same values as the OPEN model parameter.</td>
</tr>
<tr>
<td>transform</td>
<td>The name of a transformation on the output. This parameter takes the same values as the ADD_TRANSFORM name parameter. For GET_XML this must not be DDL.</td>
</tr>
<tr>
<td>base_object_name</td>
<td>The base object name. It is used internally in a BASE_OBJECT_NAME filter.</td>
</tr>
<tr>
<td>base_object_schema</td>
<td>The base object schema. It is used internally in a BASE_OBJECT_SCHEMA filter. The default is the current user.</td>
</tr>
<tr>
<td>grantee</td>
<td>The grantee. It is used internally in a GRANTEE filter. The default is the current user.</td>
</tr>
<tr>
<td>object_count</td>
<td>The maximum number of objects to return. See SET_COUNT Procedure.</td>
</tr>
</tbody>
</table>

Return Values

The metadata for the specified object as XML or DDL.

Usage Notes

- These functions allow you to fetch metadata for objects with a single call. They encapsulate calls to OPEN, SET_FILTER, and so on. The function you use depends on the characteristics of the object type and on whether you want XML, SXML, or DDL.
  - GET_xxx is used to fetch named objects, especially schema objects (tables, views).
  - GET_DEPENDENT_xxx is used to fetch dependent objects (audits, object grants).
  - GET_GRANTED_xxx is used to fetch granted objects (system grants, role grants).
- For some object types you can use more than one function. For example, you can use GET_xxx to fetch an index by name, or GET_DEPENDENT_xxx to fetch the same index by specifying the table on which it is defined.
- GET_xxx only returns a single named object.
- For GET_DEPENDENT_xxx and GET_GRANTED_xxx, an arbitrary number of dependent or granted objects can match the input criteria. You can specify an object count when fetching these objects. (The default count of 10000 should be adequate in most cases.)
- If the DDL transform is specified, session-level transform parameters are inherited.
- If you invoke these functions from SQL*Plus, you should set the PAGESIZE to 0 and set LONG to some large number to get complete, uninterrupted output.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
• OBJECT_NOT_FOUND. The specified object was not found in the database.

Examples

Example: Fetch the XML Representation of SCOTT.EMP

To generate complete, uninterrupted output, set the \texttt{PAGESIZE} to 0 and set \texttt{LONG} to some large number, as shown, before executing your query.

\begin{verbatim}
SET LONG 2000000
SET PAGESIZE 0
SELECT DBMS_METADATA.GET_XML('TABLE','EMP','SCOTT')
FROM DUAL;
\end{verbatim}

Example: Fetch the DDL for all Complete Tables in the Current Schema, Filter Out Nested Tables and Overflow Segments

This example fetches the DDL for all "complete" tables in the current schema, filtering out nested tables and overflow segments. The example uses \texttt{SET_TRANSFORM_PARAM} (with the handle value = \texttt{DBMS_METADATA.SESSION_TRANSFORM} meaning "for the current session") to specify that storage clauses are not to be returned in the SQL DDL. Afterwards, the example resets the session-level parameters to their defaults.

To generate complete, uninterrupted output, set the \texttt{PAGESIZE} to 0 and set \texttt{LONG} to some large number, as shown, before executing your query.

\begin{verbatim}
SET LONG 2000000
SET PAGESIZE 0
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(DBMS_METADATA.SESSION_TRANSFORM,'STORAGE',false);
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
FROM USER_ALL_TABLES u
WHERE u.nested='NO'
    AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(DBMS_METADATA.SESSION_TRANSFORM,'DEFAULT');
\end{verbatim}

Example: Fetch the DDL For All Object Grants On HR.EMPLOYEES

\begin{verbatim}
SELECT DBMS_METADATA.GET_DEPENDENT_DDL('OBJECT_GRANT','EMPLOYEES','HR') FROM DUAL;
\end{verbatim}

Example: Fetch the DDL For All System Grants Granted To SCOTT

\begin{verbatim}
SELECT DBMS_METADATA.GET_GRANTED_DDL('SYSTEM_GRANT','SCOTT') FROM DUAL;
\end{verbatim}

100.6.6 \texttt{GET_QUERY} Function

This function returns the text of the queries that are used by \texttt{FETCH\_xxx}. This function assists in debugging.

\begin{itemize}
  \item See Also:
    \begin{itemize}
    \item Subprograms for Retrieving Multiple Objects From the Database
    \end{itemize}
\end{itemize}
Syntax

DBMS_METADATA.GET_QUERY (
    handle  IN NUMBER)
RETURN VARCHAR2;

Parameters

Table 100-10  GET_QUERY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN. It cannot be the handle for a heterogeneous object type.</td>
</tr>
</tbody>
</table>

Return Values

The text of the queries that will be used by FETCH_xxx.

Exceptions

• INVALID_ARGVAL. A NULL or invalid value was supplied for the handle parameter.

100.6.7 OPEN Function

The DBMS_METADATA.OPEN function specifies the type of object to be retrieved, the version of its metadata, and the object model.

The return value is an opaque context handle for the set of objects to be used in subsequent calls.

See Also:

For more information about related subprograms:

• Subprograms for Retrieving Multiple Objects From the Database

Syntax

DBMS_METADATA.OPEN (
    object_type  IN VARCHAR2,
    version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
    model        IN VARCHAR2 DEFAULT 'ORACLE',
    network_link IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
Parameters

Table 100-11  Open Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| object_type   | The type of object to be retrieved. Table 100-12 lists the valid type names and their meanings. These object types will be supported for the ORACLE model of metadata (see model in this table). The Attributes column in Table 100-12 specifies some object type attributes:  
  • Schema objects, such as tables, belong to schemas.  
  • Named objects have unique names (if they are schema objects, the name is unique to the schema).  
  • Dependent objects, such as indexes, are defined with reference to a base schema object.  
  • Granted objects are granted or assigned to a user or role and therefore have a named grantee.  
  • Heterogeneous object types denote a collection of related objects of different types. See Table 100-13 for a listing of object types returned for the heterogeneous object type. These attributes are relevant when choosing object selection criteria. See "SET_FILTER Procedure" for more information. |
| version       | The version of metadata to be extracted. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are as follows: COMPATIBLE (default)—the version of the metadata corresponds to the database compatibility level. LATEST—the version of the metadata corresponds to the database version. A specific database version. The value cannot be lower than 9.2.0. |
| model         | Specifies which view to use, because the API can support multiple views on the metadata. Only the ORACLE model is supported. |
| network_link  | The name of a database link to the database whose metadata is to be retrieved. If NULL (the default), metadata is retrieved from the database on which the caller is running |

Table 100-12 provides the name, meaning, attributes, and notes for the DBMS_METADATA package object types. In the attributes column, S represents a schema object, N represents a named object, D represents a dependent object, G represents a granted object, and H represents a heterogeneous object.

Table 100-12  DBMS_METADATA: Object Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Meaning</th>
<th>Attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AQ_QUEUE</td>
<td>queues</td>
<td>SND</td>
<td>Dependent on table</td>
</tr>
<tr>
<td>AQ_QUEUE_TABLE</td>
<td>additional metadata for queue tables</td>
<td>ND</td>
<td>Dependent on table</td>
</tr>
<tr>
<td>AQ_TRANSFORM</td>
<td>transforms</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>ASSOCIATION</td>
<td>associate statistics</td>
<td>D</td>
<td>None</td>
</tr>
<tr>
<td>Type Name</td>
<td>Meaning</td>
<td>Attributes</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>AUDIT</td>
<td>audits of SQL statements</td>
<td>DG</td>
<td>Modeled as dependent, granted object. The base object name is the statement audit option name (for example, ALTER SYSTEM). There is no base object schema. The grantee is the user or proxy whose statements are audited.</td>
</tr>
<tr>
<td>AUDIT_OBJ</td>
<td>audits of schema objects</td>
<td>D</td>
<td>None</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>clusters</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>COMMENT</td>
<td>comments</td>
<td>D</td>
<td>None</td>
</tr>
</tbody>
</table>
| CONSTRAINT       | constraints                      | SND        | Does not include:  
  * primary key constraint for IOT  
  * column NOT NULL constraints  
  * certain REF SCOPE and WITH ROWID constraints for tables with REF columns |
<p>| CONTEXT          | application contexts             | N          | None                                                                  |
| DATABASE_EXPORT  | all metadata objects in a database | H         | Corresponds to a full database export                                |
| DB_LINK          | database links                   | SN         | Modeled as schema objects because they have owners. For public links, the owner is PUBLIC. For private links, the creator is the owner. |
| DEFAULT_ROLE     | default roles                    | G          | Granted to a user by ALTER USER                                      |
| DIMENSION        | dimensions                       | SN         | None                                                                  |
| DIRECTORY        | directories                      | N          | None                                                                  |
| FGA_POLICY       | fine-grained audit policies      | D          | Not modeled as named object because policy names are not unique.     |
| FUNCTION         | stored functions                 | SN         | None                                                                  |
| INDEX_STATISTICS | precomputed statistics on indexes| D          | The base object is the index's table.                                |
| INDEX            | indexes                          | SND        | None                                                                  |
| INDEXTYPE        | indextypes                       | SN         | None                                                                  |
| JAVA_SOURCE      | Java sources                     | SN         | None                                                                  |
| JOB              | jobs                             | S          | None                                                                  |
| LIBRARY          | external procedure libraries     | SN         | None                                                                  |
| MATERIALIZED_VIEW| materialized views               | SN         | None                                                                  |
| MATERIALIZED_VIEW_LOG | materialized view logs   | D          | None                                                                  |
| OBJECT_GRANT     | object grants                    | DG         | None                                                                  |
| OPERATOR         | operators                        | SN         | None                                                                  |
| PACKAGE          | stored packages                  | SN         | By default, both package specification and package body are retrieved. See &quot;SET_FILTER Procedure&quot;. |</p>
<table>
<thead>
<tr>
<th>Type Name</th>
<th>Meaning</th>
<th>Attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACKAGE_SPEC</td>
<td>package specifications</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>PACKAGE_BODY</td>
<td>package bodies</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>stored procedures</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>PROFILE</td>
<td>profiles</td>
<td>N</td>
<td>None</td>
</tr>
<tr>
<td>PROXY</td>
<td>proxy authentications</td>
<td>G</td>
<td>Granted to a user by ALTER USER</td>
</tr>
<tr>
<td>REP_CONSTRAINT</td>
<td>referential constraint</td>
<td>SND</td>
<td>None</td>
</tr>
<tr>
<td>REFRESH_GROUP</td>
<td>refresh groups</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>RESOURCE_COST</td>
<td>resource cost info</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>RLS_CONTEXT</td>
<td>driving contexts for enforcement of fine-grained access-control policies</td>
<td>D</td>
<td>Corresponds to the DBMS_RLS.ADD_POLICY_CONTENT procedure</td>
</tr>
<tr>
<td>RLS_GROUP</td>
<td>fine-grained access-control policy groups</td>
<td>D</td>
<td>Corresponds to the DBMS_RLS.CREATE_GROUP procedure</td>
</tr>
<tr>
<td>RLS_POLICY</td>
<td>fine-grained access-control policies</td>
<td>D</td>
<td>Corresponds to DBMS_RLS.ADD_GROUPED_POLICY. Not modeled as named objects because policy names are not unique.</td>
</tr>
<tr>
<td>RMGR_CONSUMER_GROUP</td>
<td>resource consumer groups</td>
<td>SN</td>
<td>Data Pump does not use these object types. Instead, it exports resource manager objects as procedural objects.</td>
</tr>
<tr>
<td>RMGRInicial_CONSUMER_GROUP</td>
<td>assign initial consumer groups to users</td>
<td>G</td>
<td>None</td>
</tr>
<tr>
<td>RMGR_PLAN</td>
<td>resource plans</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>RMGR_PLAN_DIRECTIVE</td>
<td>resource plan directives</td>
<td>D</td>
<td>Dependent on resource plan</td>
</tr>
<tr>
<td>ROLE</td>
<td>roles</td>
<td>N</td>
<td>None</td>
</tr>
<tr>
<td>ROLE_GRANT</td>
<td>role grants</td>
<td>G</td>
<td>None</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENT</td>
<td>rollback segments</td>
<td>N</td>
<td>None</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>all metadata objects in a schema</td>
<td>H</td>
<td>Corresponds to user-mode export.</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>sequences</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td>SYNONYM</td>
<td>synonyms</td>
<td></td>
<td>Private synonyms are schema objects. Public synonyms are not, but for the purposes of this API, their schema name is PUBLIC. The name of a synonym is considered to be the synonym itself. For example, in CREATE PUBLIC SYNONYM FOO FOR BAR, the resultant object is considered to have name FOO and schema PUBLIC.</td>
</tr>
<tr>
<td>SYSTEM_GRANT</td>
<td>system privilege grants</td>
<td>G</td>
<td>None</td>
</tr>
<tr>
<td>TABLE</td>
<td>tables</td>
<td>SN</td>
<td>None</td>
</tr>
</tbody>
</table>
### Table 100-12 (Cont.) DBMS_METADATA: Object Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Meaning</th>
<th>Attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TABLE_DATA</strong></td>
<td>metadata describing row data for a table, nested table, or partition</td>
<td>SND</td>
<td>For partitions, the object name is the partition name. For nested tables, the object name is the storage table name. The base object is the top-level table to which the table data belongs. For nested tables and partitioning, this is the top-level table (not the parent table or partition). For nonpartitioned tables and non-nested tables this is the table itself.</td>
</tr>
<tr>
<td><strong>TABLE_EXPORT</strong></td>
<td>metadata for a table and its associated objects</td>
<td>H</td>
<td>Corresponds to table-mode export</td>
</tr>
<tr>
<td><strong>TABLE_STATISTICS</strong></td>
<td>precomputed statistics on tables</td>
<td>D</td>
<td>None</td>
</tr>
<tr>
<td><strong>TABLESPACE</strong></td>
<td>tablespaces</td>
<td>N</td>
<td>None</td>
</tr>
<tr>
<td><strong>TABLESPACE_QUOTA</strong></td>
<td>tablespace quotas</td>
<td>G</td>
<td>Granted with ALTER USER</td>
</tr>
<tr>
<td><strong>TRANSPORTABLE_EXPORT</strong></td>
<td>metadata for objects in a transportable tablespace set</td>
<td>H</td>
<td>Corresponds to transportable tablespace export</td>
</tr>
<tr>
<td><strong>TRIGGER</strong></td>
<td>triggers</td>
<td>SND</td>
<td>None</td>
</tr>
<tr>
<td><strong>TRUSTED_DB_LINK</strong></td>
<td>trusted links</td>
<td>N</td>
<td>None</td>
</tr>
<tr>
<td><strong>TYPE</strong></td>
<td>user-defined types</td>
<td>SN</td>
<td>By default, both type and type body are retrieved. See &quot;SET_FILTER Procedure&quot;.</td>
</tr>
<tr>
<td><strong>TYPE_SPEC</strong></td>
<td>type specifications</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td><strong>TYPE_BODY</strong></td>
<td>type bodies</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td><strong>USER</strong></td>
<td>users</td>
<td>N</td>
<td>None</td>
</tr>
<tr>
<td><strong>VIEW</strong></td>
<td>views</td>
<td>SN</td>
<td>None</td>
</tr>
<tr>
<td><strong>XMLSCHEMA</strong></td>
<td>XML schema</td>
<td>SN</td>
<td>The object's name is its URL (which may be longer than 30 characters). Its schema is the user who registered it.</td>
</tr>
<tr>
<td><strong>XS_USER</strong></td>
<td>Real Application Security (RAS) user</td>
<td>N</td>
<td>Corresponds to RAS users</td>
</tr>
<tr>
<td><strong>XS_ROLE</strong></td>
<td>Real Application Security (RAS) role</td>
<td>N</td>
<td>Corresponds to RAS roles</td>
</tr>
<tr>
<td><strong>XS_ROLESET</strong></td>
<td>Real Application Security (RAS) rolesets</td>
<td>N</td>
<td>Corresponds to RAS rolesets</td>
</tr>
<tr>
<td><strong>XS_ROLE_GRANT</strong></td>
<td>Real Application Security (RAS) role grants</td>
<td>N</td>
<td>Corresponds to RAS role grants</td>
</tr>
<tr>
<td><strong>XS_SECURITY_CLASS</strong></td>
<td>Real Application Security (RAS) security class</td>
<td>SN</td>
<td>Corresponds to RAS security classes</td>
</tr>
<tr>
<td><strong>XS_DATA_SECURITY</strong></td>
<td>Real Application Security (RAS) data security policy</td>
<td>SN</td>
<td>Corresponds to RAS data security policies</td>
</tr>
<tr>
<td><strong>XS_ACL</strong></td>
<td>Real Application Security (RAS) ACL</td>
<td>SN</td>
<td>Corresponds to RAS access control lists (ACLs) and associated access control entries (ACEs)</td>
</tr>
</tbody>
</table>
Table 100-12  (Cont.) DBMS_METADATA: Object Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Meaning</th>
<th>Attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>XS_ACL_PARAM</td>
<td>Real Application Security (RAS) ACL parameter</td>
<td>N</td>
<td>Corresponds to RAS access control lists (ACL) parameters</td>
</tr>
<tr>
<td>XS_NAMESPACE</td>
<td>Real Application Security (RAS) namespace</td>
<td>N</td>
<td>Corresponds to RAS namespaces.</td>
</tr>
</tbody>
</table>

Table 100-13 lists the types of objects returned for the major heterogeneous object types. For SCHEMA_EXPORT, certain object types are only returned if the INCLUDE_USER filter is specified at TRUE. In the table, such object types are marked INCLUDE_USER.

Table 100-13  Object Types Returned for the Heterogeneous Object Type

<table>
<thead>
<tr>
<th>Object Type</th>
<th>DATABASE_EXPORT</th>
<th>SCHEMA_EXPORT</th>
<th>TABLE_EXPORT</th>
<th>TRANSPORTABLE_EXPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSOCIATION</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AUDIT</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AUDIT_OBJ</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>COMMENT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CONTEXT</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DB_LINK</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DEFAULT_ROLE</td>
<td>Yes</td>
<td>INCLUDE_USER</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DIMENSION</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DIRECTORY</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>FGA_POLICY</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>INDEX_STATISTICS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>INDEX</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>INDEXTYPE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>JAVA_SOURCE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>JOB</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LIBRARY</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MATERIALIZED_VIEW</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MATERIALIZED_VIEW_LOG</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OBJECT_GRANT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OPERATOR</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PACKAGE_SPEC</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PACKAGE_BODY</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 100-13  (Cont.) Object Types Returned for the Heterogeneous Object Type

<table>
<thead>
<tr>
<th>Object Type</th>
<th>DATABASE_EXPORT</th>
<th>SCHEMAEXPORT</th>
<th>TABLEEXPORT</th>
<th>TRANSPORTABLEEXPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD_HISTORY</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PASSWORD_VERIFY_FUNCTION</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PROFILE</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PROXY</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>REF_CONSTRAINT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>REFRESH_GROUP</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RESOURCE_COST</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RLS_CONTEXT</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RLS_GROUP</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RLS_POLICY</td>
<td>Yes</td>
<td>Table data is retrieved according to policy</td>
<td>Table data is retrieved according to policy</td>
<td>Yes</td>
</tr>
<tr>
<td>ROLE</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ROLE_GRANT</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENT</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYNONYM</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYSTEM_GRANT</td>
<td>Yes</td>
<td>INCLUDE_USER</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TABLE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TABLE_DATA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TABLE_STATISTICS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TABLESPACE</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TABLESPACE_QUOTA</td>
<td>Yes</td>
<td>INCLUDE_USER</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TRUSTED_DB_LINK</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TYPE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if the types are used by tables in the transportable set</td>
</tr>
<tr>
<td>TYPE_SPEC</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if the types are used by tables in the transportable set</td>
</tr>
<tr>
<td>TYPE_BODY</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if the types are used by tables in the transportable set</td>
</tr>
<tr>
<td>USER</td>
<td>Yes</td>
<td>INCLUDE_USER</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VIEW</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>XMLSCHEMA</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Return Values

An opaque handle to the class of objects. This handle is used as input to SET_FILTER, SET_COUNT, ADD_TRANSFORM, GET_QUERY, SET_PARSE_ITEM, FETCH.xxx, and CLOSE.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OBJECT_PARAM. The version or model parameter was not valid for the object_type.

100.6.8 OPENW Function

This function specifies the type of object to be submitted and the object model. The return value is an opaque context handle.

See Also:

For more information about related subprograms:

- Subprograms for Submitting XML to the Database

Syntax

```sql
DBMS_METADATA.OPENW
  (object_type  IN VARCHAR2,
   version       IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model         IN VARCHAR2 DEFAULT 'ORACLE')
RETURN NUMBER;
```

Parameters

**Table 100-14 OPENW Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The type of object to be submitted. Valid types names and their meanings are listed in Table 100-12. The type cannot be a heterogeneous object type.</td>
</tr>
<tr>
<td>version</td>
<td>The version of DDL to be generated by the CONVERT function. DDL clauses that are incompatible with the version will not be generated. The legal values for this parameter are as follows:</td>
</tr>
<tr>
<td></td>
<td>- COMPATIBLE - This is the default. The version of the DDL corresponds to the database compatibility level. Database compatibility must be set to 9.2.0 or higher.</td>
</tr>
<tr>
<td></td>
<td>- LATEST - The version of the DDL corresponds to the database version.</td>
</tr>
<tr>
<td></td>
<td>- A specific database version. The value cannot be lower than 9.2.0.</td>
</tr>
<tr>
<td>model</td>
<td>Specifies which view to use. Only the Oracle proprietary (ORACLE) view is supported by DBMS_METADATA.</td>
</tr>
</tbody>
</table>
Return Values
An opaque handle to write context. This handle is used as input to the ADD_TRANSFORM, CONVERT, PUT, and CLOSE procedures.

Exceptions
- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID_OBJECT_PARAM. The model parameter was not valid for the object_type.

100.6.9 PUT Function
This function submits an XML document containing object metadata to the database to create the object.

See Also:
For more information about related subprograms:
- Subprograms for Submitting XML to the Database

Syntax

```sql
DBMS_METADATA.PUT (
    handle IN NUMBER,
    document IN sys.XMLType,
    flags IN NUMBER,
    results IN OUT NOCOPY sys.ku$_SubmitResults)
RETURN BOOLEAN;
```

```
DBMS_METADATA.PUT (
    handle IN NUMBER,
    document IN CLOB,
    flags IN NUMBER,
    results IN OUT NOCOPY sys.ku$_SubmitResults)
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPENW.</td>
</tr>
<tr>
<td>document</td>
<td>The XML document containing object metadata for the type of the OPENW handle.</td>
</tr>
<tr>
<td>flags</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>results</td>
<td>Detailed results of the operation.</td>
</tr>
</tbody>
</table>
Return Values

TRUE if all SQL operations succeeded; FALSE if there were any errors.

Usage Notes

The PUT function converts the XML document to DDL just as CONVERT does (applying the specified transforms in turn) and then submits each resultant DDL statement to the database. As with CONVERT, the DDL transform must be specified. The DDL statements and associated parse items are returned in the sys.ku$_SubmitResults nested table. Each DDL statement is a nested table of error lines containing any errors or exceptions raised by the statement.

The encoding of the XML document is embedded in its CLOB or XMLType representation. The version of the metadata is embedded in the XML. The generated DDL is valid for the database version specified in OPENW.

Exceptions

- INVALID_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INCONSISTENT_OPERATION. The DDL transform was not specified.
- INCOMPATIBLE_DOCUMENT. The version of the XML document is not compatible with this version of the software.

100.6.10 SET_COUNT Procedure

This procedure specifies the maximum number of objects to be retrieved in a single FETCH_xxx call.

By default, each call to FETCH_xxx returns one object. You can use the SET_COUNT procedure to override this default. If FETCH_xxx is called from a client, specifying a count value greater than 1 can result in fewer server round trips and, therefore, improved performance.

For heterogeneous object types, a single FETCH_xxx operation only returns objects of a single object type.

See Also:

For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database

Syntax

```sql
DBMS_METADATA.SET_COUNT ( handle IN NUMBER, value IN NUMBER, object_type_path IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 100-16  SET_COUNT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>value</td>
<td>The maximum number of objects to retrieve.</td>
</tr>
<tr>
<td>object_type_path</td>
<td>A path name designating the object types to which the count value applies.</td>
</tr>
<tr>
<td></td>
<td>By default, the count value applies to the object type of the OPEN handle.</td>
</tr>
<tr>
<td></td>
<td>When the OPEN handle designates a heterogeneous object type, behavior can</td>
</tr>
<tr>
<td></td>
<td>be either of the following:</td>
</tr>
<tr>
<td></td>
<td>· If object_type_path is omitted, then the count applies to all object</td>
</tr>
<tr>
<td></td>
<td>types within the heterogeneous collection.</td>
</tr>
<tr>
<td></td>
<td>· If object_type_path is specified, then the count only applies to the</td>
</tr>
<tr>
<td></td>
<td>specific node (or set of nodes) within the tree of object types forming</td>
</tr>
<tr>
<td></td>
<td>the heterogeneous collection.</td>
</tr>
</tbody>
</table>

Exceptions

- **INVALID_ARGVAL.** A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- **INVALID_OPERATION.** SET_COUNT was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_COUNT for the current OPEN context are permitted.
- **INCONSISTENT_ARGS.** object_type parameter is not consistent with handle.

100.6.11 SET_FILTER Procedure

This procedure specifies restrictions on the objects to be retrieved, for example, the object name or schema.

**See Also:**

For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database

Syntax

```sql
DBMS_METADATA.SET_FILTER (  
    handle           IN NUMBER,  
    name             IN VARCHAR2,  
    value            IN VARCHAR2,  
    object_type_path IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_FILTER (  
    handle           IN NUMBER,  
    name             IN VARCHAR2,  
    value            IN BOOLEAN DEFAULT TRUE,
```
object_type_path IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_FILTER (
    handle IN NUMBER,
    name IN VARCHAR2,
    value IN NUMBER,
    object_type_path IN VARCHAR2 DEFAULT NULL);

**Parameters**

**Table 100-17 SET_FILTER Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the filter. For each filter, Table 100-18 lists the object_type it applies to, its name, its datatype (text or Boolean), and its meaning or effect (including its default value, if any). The Datatype column of Table 100-18 also indicates whether a text filter is an expression filter. An expression filter is the right-hand side of a SQL comparison (that is, a SQL comparison operator (=, !=, and so on,) and the value compared against. The value must contain parentheses and quotation marks where appropriate. Note that in PL/SQL and SQL*Plus, two single quotes (not a double quote) are needed to represent an apostrophe. For example, an example of a NAME_EXPR filter in PL/SQL is as follows: 'IN (''DEPT'',''EMP'')' The filter value is combined with a particular object attribute to produce a WHERE condition in the query that fetches the objects. By default, all named objects of object_type are selected.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the filter. Text, Boolean, and Numeric filters are supported.</td>
</tr>
<tr>
<td>object_type_path</td>
<td>A path name designating the object types to which the filter applies. By default, the filter applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, you can use this parameter to specify a filter for a specific node or set of nodes within the tree of object types that form the heterogeneous collection. See Table 100-19 for a listing of some of the values for this parameter.</td>
</tr>
</tbody>
</table>

Table 100-18 describes the object type, name, datatype, and meaning of the filters available with the SET_FILTER procedure.

**Table 100-18 SET_FILTER: Filters**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Named objects</td>
<td>NAME</td>
<td>Text</td>
<td>Objects with this exact name are selected.</td>
</tr>
<tr>
<td>Named objects</td>
<td>NAME_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the object attribute corresponding to the object name to produce a WHERE condition in the query that fetches the objects. By default, all named objects of object_type are selected.</td>
</tr>
</tbody>
</table>
### Table 100-18 (Cont.) SET_FILTER: Filters

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Named objects</td>
<td>EXCLUDE_NAME_EXP</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the object name to specify objects that are to be excluded from the set of objects fetched. By default, all named objects of the object type are selected.</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>EXCLUDE_TABLES</td>
<td>Boolean</td>
<td>If TRUE, all paths associated with tables are excluded from the set of objects fetched. If FALSE (the default), all paths associated with tables are fetched.</td>
</tr>
<tr>
<td>Schema objects</td>
<td>SCHEMA</td>
<td>Text</td>
<td>Objects in this schema are selected. If the object type is SYNONYM, then specify PUBLIC to select public synonyms.</td>
</tr>
<tr>
<td>Schema objects</td>
<td>SCHEMA_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the object's schema. The default is determined as follows: - if BASE_OBJECT_SCHEMA is specified, then objects in that schema are selected; - otherwise, objects in the current schema are selected.</td>
</tr>
<tr>
<td>PACKAGE, TYPE</td>
<td>SPECIFICATION</td>
<td>Boolean</td>
<td>If TRUE, retrieve the package or type specification. Defaults to TRUE.</td>
</tr>
<tr>
<td>PACKAGE, TYPE</td>
<td>BODY</td>
<td>Boolean</td>
<td>If TRUE, retrieve the package or type body. Defaults to TRUE.</td>
</tr>
<tr>
<td>TABLE, CLUSTER, INDEX,</td>
<td>TABLESPACE</td>
<td>Text</td>
<td>Objects in this tablespace (or having a partition in this tablespace) are selected.</td>
</tr>
<tr>
<td>TABLE_DATA, TABLE_EXPORT,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSPORTABLE_EXPORT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE, CLUSTER, INDEX,</td>
<td>TABLESPACE_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the object's tablespace (or in the case of a partitioned table or index, the partition's tablespaces). By default, objects in all tablespaces are selected.</td>
</tr>
<tr>
<td>TABLE_DATA, TABLE_EXPORT,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSPORTABLE_EXPORT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE, objects dependent on tables</td>
<td>PRIMARY</td>
<td>Boolean</td>
<td>If TRUE, retrieve primary tables (that is, tables for which the secondary object bit in obj$ is clear. Defaults to TRUE.</td>
</tr>
<tr>
<td>TABLE, objects dependent on tables</td>
<td>SECONDARY</td>
<td>Boolean</td>
<td>If TRUE, retrieve secondary tables (that is, tables for which the secondary object bit in obj$ is set). Defaults to TRUE.</td>
</tr>
<tr>
<td>Object Type</td>
<td>Name</td>
<td>Datatype</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_NAME</td>
<td>Text</td>
<td>Objects are selected that are defined or granted on objects with this name. Specify SCHEMA for triggers on schemas. Specify DATABASE for database triggers. Column-level comments cannot be selected by column name; the base object name must be the name of the table, view, or materialized view containing the column.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_SCHEMA</td>
<td>Text</td>
<td>Objects are selected that are defined or granted on objects in this schema. If BASE_OBJECT_NAME is specified with a value other than SCHEMA or DATABASE, this defaults to the current schema.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_NAME_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the name of the base object. Not valid for schema and database triggers.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>EXCLUDE_BASE_OBJECT_NAME_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the name of the base object to specify objects that are to be excluded from the set of objects fetched. Not valid for schema and database triggers.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_SCHEMA_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the schema of the base object.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_TYPE</td>
<td>Text</td>
<td>The object type of the base object.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_TYPE_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the object type of the base object. By default no filtering is done on object type.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_TABLESPACE</td>
<td>Text</td>
<td>The tablespace of the base object.</td>
</tr>
<tr>
<td>Dependent Objects</td>
<td>BASE_OBJECT_TABLESPACE_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the tablespaces of the base object. By default, no filtering is done on the tablespace.</td>
</tr>
<tr>
<td></td>
<td>INDEX, TRIGGER</td>
<td>Boolean</td>
<td>If TRUE, select indexes or triggers even if they are system-generated. If FALSE, omit system-generated indexes or triggers. Defaults to TRUE.</td>
</tr>
<tr>
<td>Granted Objects</td>
<td>GRANTEE</td>
<td>Text</td>
<td>Objects are selected that are granted to this user or role. Specify PUBLIC for grants to PUBLIC.</td>
</tr>
<tr>
<td>Granted Objects</td>
<td>PRIVNAME</td>
<td>Text</td>
<td>The name of the privilege or role to be granted. For TABLESPACE QUOTA, only UNLIMITED can be specified.</td>
</tr>
<tr>
<td>Granted Objects</td>
<td>GRANTEE_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the grantee name.</td>
</tr>
</tbody>
</table>
#### Table 100-18 (Cont.) SET_FILTER: Filters

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granted Objects</td>
<td>EXCLUDE_GRANTEEEXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to the grantee name to specify objects that are to be excluded from the set of objects fetched.</td>
</tr>
<tr>
<td>OBJECT_GRANT</td>
<td>GRANTOR</td>
<td>Text</td>
<td>Object grants are selected that are granted by this user.</td>
</tr>
<tr>
<td>SYNONYM, JAVA_SOURCE,XMLSCHEMA</td>
<td>LONGNAME</td>
<td>Text</td>
<td>A name longer than 30 characters. Objects with this exact name are selected. If the object name is 30 characters or less, the NAME filter must be used.</td>
</tr>
<tr>
<td>SYNONYM, JAVA_SOURCE,XMLSCHEMA</td>
<td>LONGNAME_EXPR</td>
<td>Text</td>
<td>The filter value is combined with the attribute corresponding to the object's long name. By default, no filtering is done on the long name of an object.</td>
</tr>
<tr>
<td>All objects</td>
<td>CUSTOM_FILTER</td>
<td>Text</td>
<td>The text of a WHERE condition. The condition is appended to the query that fetches the objects. By default, no custom filter is used. The other filters are intended to meet the needs of the majority of users. Use CUSTOM_FILTER when no defined filters exists for your purpose. Of necessity such a filter depends on the detailed structure of the UDTs and views used in the query. Because filters may change from version to version, upward compatibility is not guaranteed.</td>
</tr>
<tr>
<td>All objects</td>
<td>EDITION</td>
<td>Text</td>
<td>The edition filter is accepted for any object type, but affects only objects that support editions. The filter is only accepted for local objects (that is, the network_link parameter is not specified in the OPEN call). The edition name must be a valid edition name. If an edition is not specified, the edition of the active session is used.</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>SCHEMA</td>
<td>Text</td>
<td>The schema whose objects are selected.</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>SCHEMA_EXPR</td>
<td>Text expression</td>
<td>The filter value is either: combined with the attribute corresponding to a schema name to produce a WHERE condition in the query that fetches schema objects, combined with the attribute corresponding to a base schema name to produce a WHERE condition in the query that fetches dependent objects. By default the current user's objects are selected.</td>
</tr>
</tbody>
</table>
### Table 100-18  (Cont.) SET_FILTER: Filters

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_EXPORT</td>
<td>INCLUDE_USER</td>
<td>Boolean</td>
<td>If TRUE, retrieve objects containing privileged information about the user. For example, USER, PASSWORD_HISTORY, TABLESPACE_QUOTA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Defaults to FALSE.</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>SCHEMA</td>
<td>Text</td>
<td>Objects (tables and their dependent objects) in this schema are selected.</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>SCHEMA_EXPR</td>
<td>Text expression</td>
<td>The filter value is either: combined with the attribute corresponding to a schema name to produce a WHERE condition in the query that fetches the tables, combined with the attribute corresponding to a base schema name to produce a WHERE condition in the query that fetches the tables' dependent objects. By default the current user's objects are selected.</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>NAME</td>
<td>Text</td>
<td>The table with this exact name is selected along with its dependent objects.</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>NAME_EXPR</td>
<td>Text expression</td>
<td>The filter value is combined with the attribute corresponding to a table name in the queries that fetch tables and their dependent objects. By default all tables in the selected schemas are selected, along with their dependent objects.</td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>BEGIN_WITH</td>
<td>Text</td>
<td>The fully qualified path name of the first object type in the heterogeneous collection to be retrieved. Objects normally fetched prior to this object type will not be retrieved.</td>
</tr>
<tr>
<td>objects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>BEGIN_AFTER</td>
<td>Text</td>
<td>The fully qualified path name of an object type after which the heterogeneous retrieval should begin. Objects of this type will not be retrieved, nor will objects normally fetched prior to this object type.</td>
</tr>
<tr>
<td>objects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>END_BEFORE</td>
<td>Text</td>
<td>The fully qualified path name of an object type where the heterogeneous retrieval should end. Objects of this type will not be retrieved, nor will objects normally fetched after this object type.</td>
</tr>
<tr>
<td>objects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>END_WITH</td>
<td>Text</td>
<td>The fully qualified path name of the last object type in the heterogeneous collection to be retrieved. Objects normally fetched after this object type will not be retrieved.</td>
</tr>
<tr>
<td>objects</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 100-18  (Cont.) SET_FILTER: Filters

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterogeneous</td>
<td>INCLUDE_PATH_EXPR</td>
<td>Text expression</td>
<td>For these two filters, the filter value is combined with the attribute corresponding to an object type path name to produce a WHERE condition in the query that fetches the object types belonging to the heterogeneous collection. Objects of types satisfying this condition are included (INCLUDE_PATH_EXPR) or excluded (EXCLUDE_PATH_EXPR) from the set of object types fetched. Path names in the filter value do not have to be fully qualified. See Table 100-19 for valid path names that can be used with these filters. BEGIN_WITH, BEGIN_AFTER, END_BEFORE, END_WITH, INCLUDE_PATH_EXPR, and EXCLUDE_PATH_EXPR all restrict the set of object types in the heterogeneous collection. By default, objects of all object types in the heterogeneous collection are retrieved.</td>
</tr>
</tbody>
</table>

Usage Notes

- Each call to SET_FILTER causes a WHERE condition to be added to the underlying query that fetches the set of objects. The WHERE conditions are concatenated with the AND keyword so that you can use multiple SET_FILTER calls to refine the set of objects to be returned. For example to specify that you want the object named EMP in schema SCOTT, do the following:

```
SET_FILTER(handle,'SCHEMA','SCOTT');
SET_FILTER(handle,'NAME','EMP');
```

- You can use the same text expression filter multiple times with different values. All the filter conditions will be applied to the query. For example, to get objects with names between Felix and Oscar, do the following:

```
SET_FILTER(handle,'NAME_EXPR','>=''FELIX''');
SET_FILTER(handle,'NAME_EXPR','<=''OSCAR''');
```

- With SET_FILTER, you can specify the schema of objects to be retrieved, but security considerations may override this specification. If the caller is SYS or has the SELECT_CATALOG_ROLE role, then any object can be retrieved; otherwise, only the following can be retrieved:

  - Schema objects owned by the current user
  - Public synonyms
  - System privileges granted to the current user or to PUBLIC
  - Grants on objects for which the current user is owner, grantor, or grantee (either explicitly or as PUBLIC).
  - SCHEMA_EXPORT where the name is the current user
  - TABLE_EXPORT where SCHEMA is the current user
If you request objects that you are not privileged to retrieve, no exception is raised; the object is not retrieved, as if it did not exist.

In stored procedures, functions, and definers-rights packages, roles (such as SELECT_CATALOG_ROLE) are disabled. Therefore, such a PL/SQL program can only fetch metadata for objects in its own schema. If you want to write a PL/SQL program that fetches metadata for objects in a different schema (based on the invoker's possession of SELECT_CATALOG_ROLE), you must make the program invokers-rights.

- For heterogeneous object types, the BEGIN_WITH and BEGIN_AFTER filters allow restart on an object type boundary. Appropriate filter values are returned by the FETCH_XML_CLOB procedure.

Filters on heterogeneous objects provide default values for filters on object types within the collection. You can override this default for a particular object type by specifying the appropriate filter for the specific object type path. For example, for SCHEMA_EXPORT the NAME filter specifies the schema to be fetched including all the tables in the schema, but you can further restrict this set of tables by supplying a NAME_EXPR filter explicitly for the TABLE object type path. Table 100-19 lists valid object type path names for the major heterogeneous object types along with an explanation of the scope of each path name. (The same information is available in the following catalog views: DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, and TABLE_EXPORT_OBJECTS.) See Table 100-18 for filters defined for each path name. These path names are valid in the INCLUDE_PATH_EXPR and EXCLUDE_PATH_EXPR filters. Path names marked with an asterisk (*) are only valid in those filters; they cannot be used as values of the SET_FILTER object_type_path parameter.

### Table 100-19  Object Type Path Names for Heterogeneous Object Types

<table>
<thead>
<tr>
<th>Heterogeneous Type</th>
<th>Path Name (*=valid only in xxx_PATH_EXPR)</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_EXPORT</td>
<td>AUDIT_OBJ</td>
<td>Object audits on the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>COMMENT</td>
<td>Table and column comments for the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>CONSTRAINT</td>
<td>Constraints (including referential constraints) on the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>*GRANT</td>
<td>Object grants on the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>INDEX</td>
<td>Indexes (including domain indexes) on the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>OBJECT_GRANT</td>
<td>Object grants on the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>REF_CONSTRAINT</td>
<td>Referential (foreign key) constraints on the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>STATISTICS</td>
<td>Statistics on the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>TABLE_DATA</td>
<td>Row data for the selected tables</td>
</tr>
<tr>
<td>TABLE_EXPORT</td>
<td>TRIGGER</td>
<td>Triggers on the selected tables</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>ASSOCIATION</td>
<td>Statistics type associations for objects in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>AUDIT_OBJ</td>
<td>Audits on all objects in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>CLUSTER</td>
<td>Clusters in the selected schemas and their indexes</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>COMMENT</td>
<td>Comments on all objects in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>CONSTRAINT</td>
<td>Constraints (including referential constraints) on all objects in the selected schemas</td>
</tr>
</tbody>
</table>
### Table 100-19  (Cont.) Object Type Path Names for Heterogeneous Object Types

<table>
<thead>
<tr>
<th>Heterogeneous Type</th>
<th>Path Name (*=valid only in xxx_PATH_EXPR)</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_EXPORT</td>
<td>DB_LINK</td>
<td>Private database links in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>DEFAULT_ROLE</td>
<td>Default roles granted to users associated with the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>DIMENSION</td>
<td>Dimensions in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>FUNCTION</td>
<td>Functions in the selected schemas and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>*GRANT</td>
<td>Grants on objects in the selected schemas</td>
</tr>
<tr>
<td>SCHEMAEXPORT</td>
<td>INDEX</td>
<td>Indexes (including domain indexes) on tables and clusters in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>INDEXTYPE</td>
<td>Indextypes in the selected schemas and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>JAVA_SOURCE</td>
<td>Java sources in the selected schemas and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>JOB</td>
<td>Jobs in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>LIBRARY</td>
<td>External procedure libraries in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>MATERIALIZED_VIEW</td>
<td>Materialized views in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>MATERIALIZED_VIEW_LOG</td>
<td>Materialized view logs on tables in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>OBJECT_GRANT</td>
<td>Grants on objects in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>OPERATOR</td>
<td>Operators in the selected schemas and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>PACKAGE</td>
<td>Packages (both specification and body) in the selected schemas, and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>PACKAGE_BODY</td>
<td>Package bodies in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>PACKAGE_SPEC</td>
<td>Package specifications in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>PASSWORD_HISTORY</td>
<td>The password history for users associated with the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>PROCEDURE</td>
<td>Procedures in the selected schemas and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>REF_CONSTRAINT</td>
<td>Referential (foreign key) constraints on tables in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>REFRESH_GROUP</td>
<td>Refresh groups in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>SEQUENCE</td>
<td>Sequences in the selected schemas and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>STATISTICS</td>
<td>Statistics on tables and indexes in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>SYNONYM</td>
<td>Private synonyms in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>TABLE</td>
<td>Tables in the selected schemas and their dependent objects (indexes, constraints, triggers, grants, audits, comments, table data, and so on)</td>
</tr>
<tr>
<td>Heterogeneous Type</td>
<td>Path Name (*=valid only in xxx_PATH_EXPR)</td>
<td>Scope</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>TABLE_DATA</td>
<td>Row data for tables in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>TABLESPACE_QUOTA</td>
<td>Tablespace quota granted to users associated with the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>TRIGGER</td>
<td>Triggers on tables in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>XS_SECURITY_CLASS</td>
<td>Oracle Real Application Security (RAS) security classes</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>XS_DATA_SECURITY</td>
<td>Oracle Real Application Security (RAS) data security policies</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>XS_ACL</td>
<td>Oracle Real Application Security (RAS) access control lists (ACLS)</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>TYPE</td>
<td>Types (both specification and body) in the selected schemas, and their dependent grants and audits</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>TYPE_BODY</td>
<td>Type bodies in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>TYPE_SPEC</td>
<td>Type specifications in the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>USER</td>
<td>User definitions for users associated with the selected schemas</td>
</tr>
<tr>
<td>SCHEMA_EXPORT</td>
<td>VIEW</td>
<td>Views in the selected schemas and their dependent objects (grants, constraints, comments, audits)</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>ASSOCIATION</td>
<td>Statistics type associations for objects in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>AUDIT</td>
<td>Audits of SQL statements</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>AUDIT_OBJ</td>
<td>Audits on all objects in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>CLUSTER</td>
<td>Clusters and their indexes</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>COMMENT</td>
<td>Comments on all objects</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>CONSTRAINT</td>
<td>Constraints (including referential constraints)</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>CONTEXT</td>
<td>Application contexts</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>DB_LINK</td>
<td>Private and public database links</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>DEFAULT_ROLE</td>
<td>Default roles granted to users in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>DIMENSION</td>
<td>Dimensions in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>DIRECTORY</td>
<td>Directory objects in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>FGA_POLICY</td>
<td>Fine-grained audit policies</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>FUNCTION</td>
<td>Functions</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>* GRANT</td>
<td>Object and system grants</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>INDEX</td>
<td>Indexes (including domain indexes) on tables and clusters</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>INDEXTYPE</td>
<td>Indextypes and their dependent grants and audits</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>JAVA_SOURCE</td>
<td>Java sources and their dependent grants and audits</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>JOB</td>
<td>Jobs</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>LIBRARY</td>
<td>External procedure libraries</td>
</tr>
</tbody>
</table>
Table 100-19  (Cont.) Object Type Path Names for Heterogeneous Object Types

<table>
<thead>
<tr>
<th>Heterogeneous Type</th>
<th>Path Name (*=valid only in xxx_PATH_EXPR)</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE_EXPORT</td>
<td>MATERIALIZED_VIEW W</td>
<td>Materialized views</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>MATERIALIZED_VIEW W_LOG</td>
<td>Materialized view logs</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>OBJECT_GRANT</td>
<td>All object grants in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>OPERATOR</td>
<td>Operators and their dependent grants and audits</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>PACKAGE</td>
<td>Packages (both specification and body) and their dependent grants and audits</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>PACKAGE_BODY</td>
<td>Package bodies</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>PACKAGE_SPEC</td>
<td>Package specifications</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>PASSWORD_HISTORY</td>
<td>Password histories for database users</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>*PASSWORD_VERIFY_FUNCTION</td>
<td>The password complexity verification function</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>PROCEDURE</td>
<td>Procedures and their dependent grants and objects</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>PROFILE</td>
<td>Profiles</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>PROXY</td>
<td>Proxy authentications</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>REF_CONSTRAINT</td>
<td>Referential (foreign key) constraints on tables in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>REFRESH_GROUP</td>
<td>Refresh groups</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>*RESOURCE_COST</td>
<td>Resource cost information</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>RLS_CONTEXT</td>
<td>Fine-grained access-control driving contexts</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>RLS_GROUP</td>
<td>Fine-grained access-control policy groups</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>RLS_POLICY</td>
<td>Fine-grained access-control policies</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>ROLE</td>
<td>Roles</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>ROLE_GRANT</td>
<td>Role grants to users in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>ROLLBACK_SEGMENT</td>
<td>Rollback segments</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>*SCHEMA (named object)</td>
<td>Database schemas including for each schema all related and dependent objects: user definitions and their attributes (default roles, role grants, tablespace quotas, and so on), objects in the schema (tables, view, packages, types, and so on), and their dependent objects (grants, audits, indexes, constraints, and so on). The NAME and NAME_EXPR filters can be used with this object type path name to designate the database schemas to be fetched.</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>SEQUENCE</td>
<td>Sequences</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>STATISTICS</td>
<td>Statistics on tables and indexes</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>SYNONYM</td>
<td>Public and private synonyms</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>SYSTEM_GRANT</td>
<td>System privilege grants</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>TABLE</td>
<td>Tables and their dependent objects (indexes, constraints, triggers, grants, audits, comments, table data, and so on)</td>
</tr>
</tbody>
</table>
Table 100-19 (Cont.) Object Type Path Names for Heterogeneous Object Types

<table>
<thead>
<tr>
<th>Heterogeneous Type</th>
<th>Path Name (*=valid only in xxx_PATH_EXPR)</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE_EXPORT</td>
<td>TABLE_DATA</td>
<td>Row data for all tables</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>TABLESPACE</td>
<td>Tablespace definitions</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>TABLESPACE_QUOTA</td>
<td>Tablespace quota granted to users in the database</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>TRIGGER</td>
<td>Triggers on the database, on schemas, and on schema objects</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>XS_USER</td>
<td>Oracle Real Application Security (RAS) users</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>XS_ROLE</td>
<td>Oracle Real Application Security (RAS) roles</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>XS_SECURITY_CLAS</td>
<td>Oracle Real Application Security (RAS) security classes</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>XS_DATA_SECURITY</td>
<td>Oracle Real Application Security (RAS) data security policies</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>XS_ACL</td>
<td>Oracle Real Application Security (RAS) access control lists (ACLs)</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>XS_NAMESPACE</td>
<td>Oracle Real Application Security (RAS) namespaces</td>
</tr>
<tr>
<td>DATABASE_EXPORT</td>
<td>TRUSTED_DB_LINK</td>
<td>Trusted links</td>
</tr>
<tr>
<td>DATABASEEXPORT</td>
<td>TYPE</td>
<td>Types (both specification and body) and their dependent grants and audits</td>
</tr>
<tr>
<td>DATABASEEXPORT</td>
<td>TYPE_BODY</td>
<td>Type bodies</td>
</tr>
<tr>
<td>DATABASEEXPORT</td>
<td>TYPE_SPEC</td>
<td>Type specifications</td>
</tr>
<tr>
<td>DATABASEEXPORT</td>
<td>USER</td>
<td>User definitions</td>
</tr>
<tr>
<td>DATABASEEXPORT</td>
<td>VIEW</td>
<td>Views</td>
</tr>
</tbody>
</table>

Exceptions

- **INVALID_ARGVAL.** A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- **INVALID_OPERATION.** SET_FILTER was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_FILTER are permitted.
- **INCONSISTENT_ARGS.** The arguments are inconsistent. Possible inconsistencies include the following:
  - The filter name is not valid for the object type associated with the OPEN context.
  - The filter name is not valid for the object_type_path.
  - The object_type_path is not part of the collection designated by handle.
  - The filter value is the wrong datatype.
100.6.12 SET_PARSE_ITEM Procedure

This procedure is used for both retrieval and submission. This procedure enables output parsing and specifies an object attribute to be parsed and returned.

See Also:

For more information about related subprograms:

- Subprograms for Retrieving Multiple Objects From the Database
- Subprograms for Submitting XML to the Database

Syntax

The following syntax applies when SET_PARSE_ITEM is used for object retrieval:

```sql
DBMS_METADATA.SET_PARSE_ITEM (
    handle       IN  NUMBER,
    name         IN  VARCHAR2,
    object_type  IN  VARCHAR2 DEFAULT NULL);
```

The following syntax applies when SET_PARSE_ITEM is used for XML submission:

```sql
DBMS_METADATA.SET_PARSE_ITEM (
    handle     IN NUMBER,
    name        IN VARCHAR2);
```

Parameters

**Table 100-20  SET_PARSE_ITEM Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPEN (or OPENW).</td>
</tr>
<tr>
<td>name</td>
<td>The name of the object attribute to be parsed and returned. See Table 100-21</td>
</tr>
</tbody>
</table>
| object_type  | Designates the object type to which the parse item applies (this is an object type name, not a path name). By default, the parse item applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, behavior can be either of the following:  
  - If `object_type` is omitted, then the parse item applies to all object types within the heterogeneous collection  
  - If `object_type` is specified, then the parse item only applies to that specific object type within the collection  

This parameter only applies when SET_PARSE_ITEM is used for object retrieval.

Table 100-21 describes the object type, name, and meaning of the items available in the SET_PARSE_ITEM procedure.
Because new items are occasionally added, you can query the `DBMS_METADATA_PARSE_ITEMS` view to see a complete list of valid parse items or to find valid parse items for a specific object type.

Table 100-21  SET_PARSE_ITEM: Parse Items

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>All objects</td>
<td>VERB</td>
<td>If <code>FETCH_XML_CLOB</code> is called, no value is returned. If <code>FETCH_DDL</code> is called, then for every row in the <code>sys.ku$_ddls</code> nested table returned by <code>FETCH_DDL</code> the verb in the corresponding <code>ddiText</code> is returned. If the <code>ddiText</code> is a SQL DDL statement, then the SQL verb (for example, CREATE, GRANT, AUDIT) is returned. If the <code>ddiText</code> is a procedure call (for example, DBMS_AQADM.CREATE_QUEUE_TABLE()) then the package.procedure-name is returned.</td>
</tr>
<tr>
<td>All objects</td>
<td>OBJECT_TYPE</td>
<td>If <code>FETCH_XML_CLOB</code> is called, an object type name from Table 100-12 is returned. If <code>FETCH_DDL</code> is called and the <code>ddiText</code> is a SQL DDL statement whose verb is CREATE or ALTER, the object type as used in the DDL statement is returned (for example, TABLE, PACKAGE_BODY, and so on). Otherwise, an object type name from Table 100-12 is returned.</td>
</tr>
<tr>
<td>Schema objects</td>
<td>SCHEMA</td>
<td>The object schema is returned. If the object is not a schema object, no value is returned.</td>
</tr>
<tr>
<td>Named objects</td>
<td>NAME</td>
<td>The object name is returned. If the object is not a named object, no value is returned.</td>
</tr>
<tr>
<td>TABLE, TABLE_DATA, INDEX</td>
<td>TABLESPACE</td>
<td>The name of the object's tablespace or, if the object is a partitioned table, the default tablespace is returned. For a TABLE_DATA object, this is always the tablespace where the rows are stored.</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>ENABLE</td>
<td>If the trigger is enabled, ENABLE is returned. If the trigger is disabled, DISABLE is returned.</td>
</tr>
<tr>
<td>OBJECT_GRANT, TABLESPACE_QUOTA</td>
<td>GRANTOR</td>
<td>The grantor is returned.</td>
</tr>
<tr>
<td>Dependent objects (including domain index secondary tables)</td>
<td>BASE_OBJECT_NAME</td>
<td>The name of the base object is returned. If the object is not a dependent object, no value is returned.</td>
</tr>
<tr>
<td>Dependent objects (including domain index secondary tables)</td>
<td>BASE_OBJECT_SCHEMA</td>
<td>The schema of the base object is returned. If the object is not a dependent object, no value is returned.</td>
</tr>
<tr>
<td>Dependent objects (including domain index secondary tables)</td>
<td>BASE_OBJECT_TYPE</td>
<td>The object type of the base object is returned. If the object is not a dependent object, no value is returned.</td>
</tr>
<tr>
<td>Granted objects</td>
<td>GRANTEE</td>
<td>The grantee is returned. If the object is not a granted object, no value is returned.</td>
</tr>
</tbody>
</table>
Usage Notes

These notes apply when using **SET_PARSE_ITEM** to retrieve objects.

By default, the **FETCH_xxx** routines return an object's metadata as XML or creation DDL. By calling **SET_PARSE_ITEM** you can request that individual attributes of the object be returned as well.

You can call **SET_PARSE_ITEM** multiple times to ask for multiple items to be parsed and returned. Parsed items are returned in the `sys.ku$_parsed_items` nested table.

For **TABLE_DATA** objects, the following parse item return values are of interest:

<table>
<thead>
<tr>
<th>If Object Is</th>
<th>NAME, SCHEMA</th>
<th>BASE_OBJECT_NAME, BASE_OBJECT_SCHEMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonpartitioned table</td>
<td>table name, schema</td>
<td>table name, schema</td>
</tr>
<tr>
<td>table partition</td>
<td>partition name, schema</td>
<td>table name, schema</td>
</tr>
<tr>
<td>nested table</td>
<td>storage table name, schema</td>
<td>name and schema of top-level table (\textit{not} ) the parent nested table (\textit{not} ) the parent nested table</td>
</tr>
</tbody>
</table>

Tables are not usually thought of as dependent objects. However, secondary tables for domain indexes are dependent on the domain indexes. Consequently, the **BASE_OBJECT_NAME**, **BASE_OBJECT_SCHEMA** and **BASE_OBJECT_TYPE** parse items for secondary **TABLE** objects return the name, schema, and type of the domain index.

![See Also:](image)

- "**FETCH_xxx Functions and Procedures**"
- *Oracle Database Utilities* for more information about using the metadata APIs.

By default, the **CONVERT** and **PUT** procedures simply transform an object's XML metadata to DDL. By calling **SET_PARSE_ITEM** you can request that individual attributes of the object be returned as well.

Exceptions

- **INVALID_ARGVAL**. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- **INVALID_OPERATION**. **SET_PARSE_ITEM** was called after the first call to **FETCH_xxx** for the **OPEN** context. After the first call to **FETCH_xxx** is made, no further calls to **SET_PARSE_ITEM** are permitted.
- **INCONSISTENT_ARGS**. The attribute name is not valid for the object type associated with the **OPEN** context.
100.6.13 SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures

These procedures are used for both retrieval and submission. SET_TRANSFORM_PARAM and SET_REMAP_PARAM specify parameters to the XSLT stylesheet identified by transform_handle.

Use them to modify or customize the output of the transform.

See Also:
For more information about related subprograms:
- Subprograms for Retrieving Multiple Objects From the Database
- Subprograms for Submitting XML to the Database

Syntax

DBMS_METADATA.SET_TRANSFORM_PARAM (transform_handle   IN NUMBER, name               IN VARCHAR2, value              IN VARCHAR2, object_type        IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_TRANSFORM_PARAM (transform_handle   IN NUMBER, name               IN VARCHAR2, value              IN BOOLEAN DEFAULT TRUE, object_type        IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_TRANSFORM_PARAM (transform_handle   IN NUMBER, name               IN VARCHAR2, value              IN NUMBER, object_type        IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_REMAP_PARAM (transform_handle   IN NUMBER, name               IN VARCHAR2, old_value          IN VARCHAR2, new_value          IN VARCHAR2, object_type        IN VARCHAR2 DEFAULT NULL);

Parameters
Table 100-22 describes the parameters for the SET_TRANSFORM_PARAM and SET_REMAP_PARAM procedures.

Because new parameters are occasionally added, you might want to query the DBMS_METADATA_TRANSFORM_PARAMS view to see all the valid transform parameters for each transform or to find valid transform parameters for specific object types.
Table 100-22  SET_TRANSFORM_PARAM and SET_REMAP_PARAM Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transform_handle</td>
<td>Either (1) the handle returned from ADD_TRANSFORM, or (2) the enumerated constant SESSION_TRANSFORM that designates the DDL transform for the whole session. Note that the handle returned by OPEN is not a valid transform handle. For SET_REMAP_PARAM, the transform handle must designate the MODIFY transform.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the transform parameter. For descriptions of the parameters available for each transform on the SET_TRANSFORM_PARAM procedure, see the following: Table 100-23 - DDL transform  Table 100-24 - MODIFY transform  Table 100-26 - SXML transform  Table 100-27 - MODIFYXML transform  Table 100-28 - SXMLDDL transform For descriptions of the parameters available for the MODIFY transform on the SET_REMAP_PARAM procedure, see Table 100-25. For descriptions of the parameters available for the ALTERXML transform, see Table 100-4.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the transform. This parameter is valid only for SET_TRANSFORM_PARAM.</td>
</tr>
<tr>
<td>old_value</td>
<td>The old value for the remapping. This parameter is valid only for SET_REMAP_PARAM.</td>
</tr>
<tr>
<td>new_value</td>
<td>The new value for the remapping. This parameter is valid only for SET_REMAP_PARAM.</td>
</tr>
<tr>
<td>object_type</td>
<td>Designates the object type to which the transform or remap parameter applies. By default, it applies to the same object type as the transform. In cases where the transform applies to all object types within a heterogeneous collection, the following apply: • If object_type is omitted, the parameter applies to all applicable object types within the heterogeneous collection. • If object_type is specified, the parameter only applies to that object type. This allows a caller who has added a transform to a heterogeneous collection to specify different transform parameters for different object types within the collection.</td>
</tr>
</tbody>
</table>

Table 100-23 describes the object type, name, datatype, and meaning of the parameters for the DDL transform in the `SET_TRANSFORM_PARAM` procedure.
### Table 100-23  SET_TRANSFORM_PARAM: Transform Parameters for the DDL Transform

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER, TABLE,</td>
<td>COLLATION_CLAUSE</td>
<td>Text</td>
<td>There are three possible values:</td>
</tr>
<tr>
<td>CLUSTER, VIEW,</td>
<td></td>
<td></td>
<td>• NEVER — Collation clauses are never generated.</td>
</tr>
<tr>
<td>MATERIALIZED_VI</td>
<td></td>
<td></td>
<td>• ALWAYS — Collation clauses are always generated.</td>
</tr>
<tr>
<td>EW PROCEDURE,</td>
<td></td>
<td></td>
<td>• NON_DEFAULT — Collation clauses other than USING_NLS_COMP are generated.</td>
</tr>
<tr>
<td>FUNCTION, PACKAGE,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE, TRIGGER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All objects</td>
<td>PRETTY</td>
<td>BOOLEAN</td>
<td>If TRUE, format the output with indentation and line feeds. Defaults to TRUE.</td>
</tr>
<tr>
<td>All objects</td>
<td>SQLTERMINATOR</td>
<td>BOOLEAN</td>
<td>If TRUE, append a SQL terminator (; or /) to each DDL statement. Defaults to FALSE.</td>
</tr>
<tr>
<td>TABLE</td>
<td>CONSTRAINTS</td>
<td>BOOLEAN</td>
<td>If TRUE, include all non-referential table constraints in the DDL. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>TABLE</td>
<td>REF_CONSTRAINTS</td>
<td>BOOLEAN</td>
<td>If TRUE, include all referential constraints (foreign keys) in the DDL. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>TABLE</td>
<td>CONSTRAINTS_AS_ALTER</td>
<td>BOOLEAN</td>
<td>If TRUE, include table constraints as separate ALTER TABLE (and, if necessary, CREATE INDEX) statements. If FALSE, specify table constraints as part of the CREATE TABLE statement. Defaults to FALSE. Requires that CONSTRAINTS be TRUE.</td>
</tr>
<tr>
<td>TABLE, TYPE</td>
<td>OID</td>
<td>BOOLEAN</td>
<td>If TRUE, include the Object ID (OID) clause in the DDL. If FALSE, omit it. Defaults to FALSE.</td>
</tr>
<tr>
<td>TABLE</td>
<td>SIZE_BYTE_KEYWORD</td>
<td>BOOLEAN</td>
<td>If TRUE, include the BYTE keyword as part of the size specification of CHAR and VARCHAR2 columns that use byte semantics. If FALSE, omit the keyword. Defaults to FALSE.</td>
</tr>
<tr>
<td>TABLE, INDEX</td>
<td>PARTITIONING</td>
<td>BOOLEAN</td>
<td>If TRUE, include partitioning clauses in the DDL. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>INDEX, CONSTRAINT,</td>
<td>SEGMENT_ATTRIBUTES</td>
<td>BOOLEAN</td>
<td>If TRUE, include segment attributes clauses (physical attributes, storage attributes, tablespace, logging) in the DDL. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENT,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE, CLUSTER,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLESPACE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX, CONSTRAINT,</td>
<td>STORAGE</td>
<td>BOOLEAN</td>
<td>If TRUE, include storage clauses in the DDL. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENT,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T, CLUSTER, TABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Type</td>
<td>Name</td>
<td>Datatype</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INDEX, CONSTRAINT,</td>
<td>TABLESPACE</td>
<td>BOOLEAN</td>
<td>If TRUE, include tablespace clauses in the DDL. If FALSE, omit them.</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENT, CLUSTER, TABLE</td>
<td></td>
<td></td>
<td>(Ignored if SEGMENT_ATTRIBUTES is FALSE.)</td>
</tr>
<tr>
<td>TYPE, PACKAGE</td>
<td>SPECIFICATION</td>
<td>BOOLEAN</td>
<td>If TRUE, include the type or package specification in the DDL. If FALSE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>omit it. Defaults to TRUE.</td>
</tr>
<tr>
<td>TYPE, PACKAGE</td>
<td>BODY</td>
<td>BOOLEAN</td>
<td>If TRUE, include the type body or package body in the DDL. If FALSE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>omit it. Defaults to TRUE.</td>
</tr>
<tr>
<td>VIEW</td>
<td>FORCE</td>
<td>BOOLEAN</td>
<td>If TRUE, use the FORCE keyword in the CREATE VIEW statement. If FALSE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>do not use the FORCE keyword in the CREATE VIEW statement. Defaults to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TRUE.</td>
</tr>
<tr>
<td>OUTLINE</td>
<td>INSERT</td>
<td>BOOLEAN</td>
<td>If TRUE, include the INSERT statements into the OL$ dictionary tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>that will create the outline and its hints. If FALSE, omit a CREATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OUTLINE statement. Defaults to FALSE.</td>
</tr>
<tr>
<td>All objects</td>
<td>DEFAULT</td>
<td>BOOLEAN</td>
<td>Calling SET_TRANSFORM_PARAM with this parameter set to TRUE has the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>effect of resetting all parameters for the transform to their default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>values. Setting this FALSE has no effect. There is no default.</td>
</tr>
<tr>
<td>All objects</td>
<td>INHERIT</td>
<td>BOOLEAN</td>
<td>If TRUE, inherits session-level parameters. Defaults to FALSE. If an</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>application calls ADD_TRANSFORM to add the DDL transform, then by</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>default the only transform parameters that apply are those explicitly</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set for that transform handle. This has no effect if the transform</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>handle is the session transform handle.</td>
</tr>
<tr>
<td>ROLE</td>
<td>REVOKE_FROM</td>
<td>Text</td>
<td>The name of a user from whom the role must be revoked. If this is a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>non-null string and if the CREATE ROLE statement grants you the role,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a REVOKE statement is included in the DDL after the CREATE ROLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: When you issue a CREATE ROLE statement, Oracle may grant you the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>role. You can use this transform parameter to undo the grant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Defaults to null string.</td>
</tr>
</tbody>
</table>

Note: This object type is being deprecated.
<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLESPACE</td>
<td>REUSE</td>
<td>BOOLEAN</td>
<td>If TRUE, include the REUSE parameter for datafiles in a tablespace to indicate that existing files can be reused. If FALSE, omit the REUSE parameter. Defaults to FALSE.</td>
</tr>
<tr>
<td>CLUSTER, INDEX, ROLLBACKSEGMENT, TABLE, TABLESPACE</td>
<td>PCTSPACE</td>
<td>NUMBER</td>
<td>A number representing the percentage by which space allocation for the object type is to be modified. The value is the number of one-hundredths of the current allocation. For example, 100 means 100%. If the object type is TABLESPACE, the following size values are affected: - in file specifications, the value of SIZE - MINIMUM EXTENT - EXTENT MANAGEMENT LOCAL UNIFORM SIZE For other object types, INITIAL and NEXT are affected.</td>
</tr>
</tbody>
</table>
| TABLE             | LOB_STORAGE        | Text      | Specifies the storage type to use for LOB segments. The options are as follows:  
- SECUREFILE - LOB storage is returned as SECUREFILE  
- BASICFILE - LOB storage is returned as BASICFILE  
- DEFAULT - The keyword (SECUREFILE or BASICFILE) is omitted in the LOB STORE AS clause.  
- NO_CHANGE - LOB segments are created with the same storage they had in the source database. This is the default. Specifying this transform changes the LOB storage for all tables in the job, including tables that provide storage for materialized views. |
| TABLE             | TABLE_COMPRESSION_CLAUSE | Text      | Specifies a table compression clause (for example, COMPRESS BASIC) to use when the table is created.  
Specify NONE to omit the table compression clause. The table will have the default compression for the tablespace.  
Specifying this transform changes the compression type for all tables in the job, including tables that provide storage for materialized views. |
Table 100-24 describes the object type, name, datatype, and meaning of the parameters for the MODIFY transform in the `SET_TRANSFORM_PARAM` procedure.

### Table 100-24  SET_TRANSFORM_PARAM: Transform Parameters for the MODIFY Transform

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>All objects</td>
<td>OBJECT_ROW</td>
<td>NUMBER</td>
<td>A number designating the object row for an object. The object in the document that corresponds to this number will be copied to the output document. This parameter is additive. By default, all objects are copied to the output document.</td>
</tr>
</tbody>
</table>

Table 100-25 describes the object type, name, datatype, and meaning of the parameters for the MODIFY transform in the `SET_REMAP_PARAM` procedure.

### Table 100-25  SET_REMAP_PARAM: Transform Parameters for the MODIFY Transform

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRARY, TABLESPACE, DIRECTORY</td>
<td>REMAP_DATAFILE</td>
<td>Text</td>
<td>Objects in the document will have their filespecs renamed as follows: any filespec matching old_value will be changed to new_value. Filespecs should not be enclosed in quotes. This parameter is additive. By default, filespecs are not renamed.</td>
</tr>
<tr>
<td>Named objects and all objects dependent on named objects</td>
<td>REMAP_NAME</td>
<td>Text</td>
<td>Any named object in the document whose name matches old_value will have its name changed to new_value. Any dependent object whose base object name matches old_value will have its base schema name changed to new_value. This parameter is additive. By default, names are not remapped. (Use REMAP_TABLESPACE to remap the name of a TABLESPACE object.)</td>
</tr>
</tbody>
</table>
### Table 100-25  (Cont.) SET_REMAP_PARAM: Transform Parameters for the MODIFY Transform

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>REMAP_SCHEMA</td>
<td>Text</td>
<td>Any schema object in the document whose name matches old_value will have its schema name changed to new_value.</td>
</tr>
<tr>
<td>Objects,</td>
<td></td>
<td></td>
<td>Any dependent object whose base object schema name matches old_value will have its base object schema name changed to new_value.</td>
</tr>
<tr>
<td>Dependent</td>
<td></td>
<td></td>
<td>Any granted object whose grantee name matches old_value will have its grantee name changed to new_value.</td>
</tr>
<tr>
<td>Objects,</td>
<td></td>
<td></td>
<td>Any user whose name matches old_value will have its name changed to new_value.</td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td>This parameter is additive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By default, schemas are not remapped.</td>
</tr>
</tbody>
</table>

NOTE: The mapping may not be 100 percent complete because there are certain schema references that Import is not capable of finding. For example, Import will not find schema references embedded within the body of definitions of triggers, types, views, procedures, and packages.

**TABLE, CLUSTER, CONSTRAINT, INDEX, ROLLBACK_SEGMENT, MATERIALIZED VIEW, MATERIALIZED VIEW_LOG, TABLESPACE_QUOTA**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>REMAP_TABLESPACE</td>
<td>Text</td>
<td>Objects in the document will have their tablespaces renamed as follows: any tablespace name matching old_value will be changed to new_value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This parameter is additive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By default, tablespaces are not remapped.</td>
</tr>
</tbody>
</table>

### Table 100-26  SET_TRANSFORM_PARAM: Transform Parameters for the SXML Transform

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER, TABLE, CLUSTER, VIEW, MATERIALIZED VIEW PROCEDURE, FUNCTION, PACKAGE, TYPE, TRIGGER</td>
<td>COLLATION_CLAUSE</td>
<td>Text</td>
<td>There are three possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• NEVER — Collation clauses are never generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ALWAYS — Collation clauses are always generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• NON_DEFAULT — Collation clauses other than USING_NLS_COMP are generated.</td>
</tr>
<tr>
<td>TABLE, TYPE</td>
<td>OID</td>
<td>Boolean</td>
<td>If TRUE, include the Oracle Internet Directory (OID) clause in the SXML. If FALSE, omit it. Defaults to FALSE.</td>
</tr>
</tbody>
</table>
### Table 100-26  (Cont.) SET_TRANSFORM_PARAM: Transform Parameters for the SXML Transform

<table>
<thead>
<tr>
<th>Object type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE, INDEX, CLUSTER,</td>
<td>STORAGE</td>
<td>Boolean</td>
<td>If TRUE, include storage clauses in the SXML. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
<tr>
<td>MATERIALIZED_VIEW,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATERIALIZED_VIEW_LOG.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE, INDEX, CLUSTER,</td>
<td>TABLESPACE</td>
<td>Boolean</td>
<td>If TRUE, include tablespace clauses in the SXML. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
<tr>
<td>MATERIALIZED_VIEW,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATERIALIZED_VIEW_LOG.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td>REF_CONSTRAINTS</td>
<td>Boolean</td>
<td>If TRUE, include all referential constraints (foreign keys) in the SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>INDEX, CONSTRAINT,</td>
<td>PHYSICAL_PROPERTIES</td>
<td>Boolean</td>
<td>If TRUE, include segment attributes clauses (physical attributes, storage attributes, tablespace, logging) in the SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENT, CLUSTER, TABLE,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLESPACE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX, CONSTRAINT,</td>
<td>SEGMENT_ATTRIBUTES</td>
<td>Boolean</td>
<td>If TRUE, include segment attributes clauses (physical attributes, storage attributes, tablespace, logging) in the SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENT, CLUSTER, TABLE,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLESPACE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td>PARTITIONING</td>
<td>Boolean</td>
<td>If TRUE, include partitioning clauses in the SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>INDEX</td>
<td>CONSTRAINTS</td>
<td>Boolean</td>
<td>If TRUE, include all non-referential table constraints in the SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
</tbody>
</table>

### Table 100-27  SET_TRANSFORM_PARAM: Transform Parameters for the MODIFYSXML Transform

<table>
<thead>
<tr>
<th>Object type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE, INDEX, CLUSTER,</td>
<td>STORAGE</td>
<td>Boolean</td>
<td>If TRUE, include storage clauses in the output SXML. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
<tr>
<td>MATERIALIZED_VIEW,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATERIALIZED_VIEW_LOG.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE, INDEX, CLUSTER,</td>
<td>TABLESPACE</td>
<td>Boolean</td>
<td>If TRUE, include tablespace clauses in the output SXML. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
<tr>
<td>MATERIALIZED_VIEW,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATERIALIZED_VIEW_LOG.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 100-27  (Cont.) SET_TRANSFORM_PARAM: Transform Parameters for the MODIFYSXML Transform

<table>
<thead>
<tr>
<th>Object type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>REF_CONSTRAINTS</td>
<td>Boolean</td>
<td>If TRUE, include all referential constraints (foreign keys) in the output SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>TABLE, INDEX, VIEW, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG</td>
<td>REMAP_NAME</td>
<td>Text</td>
<td>Any NAME element in the document that matches old_value will be changed to new_value. This does not apply to column names. (See REMAP_COLUMN_NAME).</td>
</tr>
<tr>
<td>TABLE, INDEX, VIEW, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG</td>
<td>REMAP_SCHEMA</td>
<td>Text</td>
<td>Any SCHEMA element in the document matching old_value will be changed to new_value.</td>
</tr>
<tr>
<td>TABLE, INDEX, VIEW, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG</td>
<td>REMAP_COLUMN_NAME</td>
<td>Text</td>
<td>Any column in the document whose name matches old_value will have its name changed to new_value.</td>
</tr>
<tr>
<td>TABLE, INDEX, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG</td>
<td>SEGMENT_ATTRIBUTES</td>
<td>Boolean</td>
<td>If TRUE, include segment attributes clauses (physical attributes, storage attributes, tablespace, logging) in the output SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>TABLE</td>
<td>CONSTRAINTS</td>
<td>Boolean</td>
<td>If TRUE, include all non-referential table constraints in the output SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
</tbody>
</table>

### Table 100-28  SET_TRANSFORM_PARAM: Transform Parameters for the SXMLDDL Transform

<table>
<thead>
<tr>
<th>Object type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER, TABLE, CLUSTER, VIEW, MATERIALIZED_VIEW, PROCEDURE, FUNCTION, PACKAGE, TYPE, TRIGGER</td>
<td>COLLATION_CLAUSE</td>
<td>Text</td>
<td>There are three possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• NEVER — Collation clauses are never generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ALWAYS — Collation clauses are always generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• NON_DEFAULT — Collation clauses other than USING_NLS_COMP are generated.</td>
</tr>
<tr>
<td>TABLE</td>
<td>OID</td>
<td>Boolean</td>
<td>If TRUE, include OIDs in the DDL. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
</tbody>
</table>
Table 100-28  (Cont.) SET_TRANSFORM_PARAM: Transform Parameters for the SXMLDDL Transform

<table>
<thead>
<tr>
<th>Object type</th>
<th>Name</th>
<th>Datatype</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE, INDEX,</td>
<td>TABLESPACE</td>
<td>Boolean</td>
<td>If TRUE, include tablespace clauses in the DDL. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
<tr>
<td>CLUSTER,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATERIALIZED_VIEW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>, MATERIALIZED_VIEW_LOG.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE, INDEX,</td>
<td>STORAGE</td>
<td>Boolean</td>
<td>If TRUE, include storage clauses in the DDL. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.</td>
</tr>
<tr>
<td>CLUSTER,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATERIALIZED_VIEW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>, MATERIALIZED_VIEW_LOG.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td>REF_CONSTRAINTS</td>
<td>Boolean</td>
<td>If TRUE, include all referential constraints (foreign keys) in the DDL. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>INDEX</td>
<td>PRESERVE_LOCAL</td>
<td>Boolean</td>
<td>If PARTITIONING is FALSE and PRESERVE_LOCAL is TRUE and the index is a locally partitioned index, include the LOCAL keyword in the DDL. Defaults to FALSE.</td>
</tr>
<tr>
<td>TABLE, INDEX,</td>
<td>SEGMENT_ATTRIBUTES</td>
<td>Boolean</td>
<td>If TRUE, include segment attributes clauses (physical attributes, storage attributes, tablespace, logging) in the DDL. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>CLUSTER,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATERIALIZED_VIEW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>, MATERIALIZED_VIEW_LOG.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLESPACE</td>
<td>REUSE</td>
<td>Boolean</td>
<td>If TRUE, include the REUSE parameter for datafiles in a tablespace to indicate that existing files can be reused. If FALSE, omit the REUSE parameter. Defaults to FALSE.</td>
</tr>
<tr>
<td>TABLE, INDEX</td>
<td>PARTITIONING</td>
<td>Boolean</td>
<td>If TRUE, include partitioning clauses in the DDL. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
<tr>
<td>TABLE</td>
<td>CONSTRAINTS</td>
<td>Boolean</td>
<td>If TRUE, include all non-referential table constraints in the output SXML. If FALSE, omit them. Defaults to TRUE.</td>
</tr>
</tbody>
</table>

Exceptions

- **INVALID_ARGVAL.** A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- **INVALID_OPERATION.** Either SET_TRANSFORM_PARAM or SET_REMAP_PARAM was called after the first call to FETCH_xxx for the OPEN context. After the first call to FETCH_xxx is made, no further calls to SET_TRANSFORM_PARAM or SET_REMAP_PARAM are permitted.
- **INCONSISTENT_ARGS.** The arguments are inconsistent. This can mean the following:
– The transform parameter name is not valid for the object type associated with the OPEN context or for the transform associated with the transform handle.

– The transform applies to all object types in a heterogeneous collection, but object_type is not part of the collection.

Usage Notes

XSLT allows parameters to be passed to stylesheets. You call SET_TRANSFORM_PARAM or SET_REMAP_PARAM to specify the value of a parameter to be passed to the stylesheet identified by transform_handle.

Normally, if you call SET_TRANSFORM_PARAMETER multiple times for the same parameter name, each call overrides the prior call. For example, the following sequence simply sets the STORAGE transform parameter to TRUE.

```sql
SET_TRANSFORM_PARAM(tr_handle,'STORAGE',false);
SET_TRANSFORM_PARAM(tr_handle,'STORAGE',true);
```

However, some transform parameters are additive which means that all specified parameter values are applied to the document, not just the last one. For example, the OBJECT_ROW parameter to the MODIFY transform is additive. If you specify the following, then both specified rows are copied to the output document.

```sql
SET_TRANSFORM_PARAM(tr_handle,'OBJECT_ROW',5);
SET_TRANSFORM_PARAM(tr_handle,'OBJECT_ROW',8);
```

The REMAP_TABLESPACE parameter is also additive. If you specify the following, then tablespaces TBS1 and TBS3 are changed to TBS2 and TBS4, respectively.

```sql
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS1','TBS2');
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS3','TBS4');
```

The order in which the transformations are performed is undefined. For example, if you specify the following, the result is undefined.

```sql
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS1','TBS2');
SET_REMAP_PARAM(tr_handle,'REMAP_TABLESPACE','TBS2','TBS3');
```

Note:

The number of remap parameters that can be specified for a MODIFY transform is limited to ten. That is, you can specify up to ten REMAP_DATAFILE parameters, up to ten REMAP_SCHEMA parameters and so on. Additional instances are ignored. To work around this, you can perform another DBMS_METADATA.ADD_TRANSFORM and specify additional remap parameters.

The GET_DDL, GET_DEPENDENT_DDL, and GET_GRANTED_DDL functions allow the casual browser to extract the creation DDL for an object. So that you can specify transform parameters, this package defines an enumerated constant SESSION_TRANSFORM as the handle of the DDL transform at the session level. You can call SET_TRANSFORM_PARAM using DBMS_METADATA.SESSION_TRANSFORM as the transform handle to set transform parameters for the whole session. GET_DDL, GET_DEPENDENT_DDL, and GET_GRANTED_DDL inherit these parameters when they invoke the DDL transform.
Note:

The enumerated constant must be prefixed with the package name DBMS_METADATA.SESSION_TRANSFORM.
The `DBMS_METADATA_DIFF` package contains the interfaces for comparing two metadata documents in SXML format.

This chapter contains the following topics:

- Overview
- Security Model
- Browsing APIs for Fetching and Comparing Objects
- Summary of DBMS_METADATA_DIFF Subprograms

**See Also:**

Oracle Database Utilities for more information and for examples of using the `DBMS_METADATA_DIFF` package.

### 101.1 DBMS_METADATA_DIFF Overview

You can use the interfaces contained in the `DBMS_METADATA_DIFF` package to compare two metadata documents in SXML format.

The result of the comparison is an SXML difference document. This document can be converted to other formats using the `DBMS_METADATA` submit interface and the `CONVERT` API.

### 101.2 DBMS_METADATA_DIFF Security Model

The browsing interface of the `DBMS_METADATA_DIFF` package actually uses the `DBMS_METADATA` package to fetch the metadata to be compared. Therefore, the security model used for `DBMS_METADATA` also applies to `DBMS_METADATA_DIFF`. (Note, however, that `DBMS_METADATA_DIFF` does not support all object types.)

**See Also:**

`DBMS_METADATA` for information about the `DBMS_METADATA` security model.
101.3 DBMS_METADATA_DIFF — Browsing APIs for Fetching and Comparing Objects

These functions allow you to compare the metadata for two objects with a single call.

Syntax

DBMS_METADATA_DIFF.COMPARE_SXML(
object_type   IN VARCHAR2,
name1         IN VARCHAR2,
name2         IN VARCHAR2,
schema1       IN VARCHAR2 DEFAULT NULL,
schema2       IN VARCHAR2 DEFAULT NULL,
network_link1 IN VARCHAR2 DEFAULT NULL,
network_link2 IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;

DBMS_METADATA_DIFF.COMPARE_ALTER(
object_type   IN VARCHAR2,
name1         IN VARCHAR2,
name2         IN VARCHAR2,
schema1       IN VARCHAR2 DEFAULT NULL,
schema2       IN VARCHAR2 DEFAULT NULL,
network_link1 IN VARCHAR2 DEFAULT NULL,
network_link2 IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;

DBMS_METADATA_DIFF.COMPARE_ALTER_XML(
object_type   IN VARCHAR2,
name1         IN VARCHAR2,
name2         IN VARCHAR2,
schema1       IN VARCHAR2 DEFAULT NULL,
schema2       IN VARCHAR2 DEFAULT NULL,
network_link1 IN VARCHAR2 DEFAULT NULL,
network_link2 IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;

Parameters

Table 101-1  COMPARE_xxx Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The type of object to be compared. Valid type names are CLUSTER, CONTEXT, DB_LINK, FGA_POLICY, INDEX, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG, QUEUE, QUEUE_TABLE, RLS_CONTEXT, RLS_GROUP, RLS_POLICY, ROLE, SEQUENCE, SYNONYM, TABLE, TABLESPACE, TRIGGER, TYPE, TYPE_SPEC, TYPE_BODY, USER, and VIEW.</td>
</tr>
<tr>
<td>name1</td>
<td>The name of the first object in the comparison.</td>
</tr>
<tr>
<td>name2</td>
<td>The name of the second object in the comparison.</td>
</tr>
<tr>
<td>schema1</td>
<td>The schema of the first object in the comparison. The default is the current user.</td>
</tr>
</tbody>
</table>
Table 101-1 (Cont.) COMPARE_xxx Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema2</td>
<td>The schema of the second object in the comparison. The default is the value of schema1.</td>
</tr>
<tr>
<td>network_link1</td>
<td>The name of a database link to the database on which the first object resides. If NULL (the default), then the object is assumed to be in the database on which the caller is running.</td>
</tr>
<tr>
<td>network_link2</td>
<td>The name of a database link to the database on which the second object resides. The default is the value of network_link1.</td>
</tr>
</tbody>
</table>

Return Values

DBMS_METADATA_DIFF.COMPARE_xxx returns the differences between two objects.

Exceptions

- INVALID_ARGVAL
  A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- OBJECT_NOT_FOUND
  The specified object was not found in the database.

Usage Notes

These functions encapsulate calls to both DBMS_METADATA and DBMS_METADATA_DIFF functions and procedures to fetch the metadata for each of the two objects and compare them.

Which function you use depends on the comparison format you want:

- COMPARE_SXML returns an SXML difference document.
- COMPARE_ALTER returns a set of ALTER statements for making the first object like the second object.
- COMPARE_ALTER_XML returns an ALTER_XML document.

101.4 Summary of DBMS_METADATA_DIFF Subprograms

The DBMS_METADATA_DIFF subprograms provide comparison functionality for different object types.

These subprograms are used to:

- Specify the type of objects to be compared
- Specify the SXML documents to be compared
- Show the differences between the compared documents
- Clean up after the comparison

Table 101-2 provides a summary of DBMS_METADATA_DIFF subprograms.
### Table 101-2  DBMS_METADATA_DIFF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPENC Function</td>
<td>Specifies the type of objects to be compared</td>
</tr>
<tr>
<td>ADD_DOCUMENT Procedure</td>
<td>Specifies an SXML document to be compared</td>
</tr>
<tr>
<td>FETCH_CLOB Functions and Procedures</td>
<td>Returns a CLOB showing the differences between the two documents specified by ADD_DOCUMENT</td>
</tr>
<tr>
<td>CLOSE Procedure</td>
<td>Invalidates the handle returned by OPENC and cleans up associated state</td>
</tr>
</tbody>
</table>

#### 101.4.1 OPENC Function

This function specifies the type of objects to be compared. The return value is an opaque context handle.

**Syntax**

```sql
DBMS_METADATA_DIFF.OPENC (object_type  IN VARCHAR2)
RETURN NUMBER;
```

**Parameters**

**Table 101-3  OPENC Function Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The type of object to be compared. Valid type names are CLUSTER, CONTEXT, DB_LINK, FGA_POLICY, INDEX, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG, QUEUE, QUEUE_TABLE, RLS_CONTEXT, RLS_GROUP, RLS_POLICY, ROLE, SEQUENCE, SYNONYM, TABLE, TABLESPACE, TRIGGER, TYPE, TYPE_SPEC, TYPE_BODY, USER, and VIEW.</td>
</tr>
</tbody>
</table>

**Return Values**

The opaque handle that is returned is used as input to ADD_DOCUMENT, FETCH_xxx and CLOSE.

**Exceptions**

- **INVALID_ARGVAL**
  
  A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
101.4.2 ADD_DOCUMENT Procedure

This procedure specifies an SXML document that is to be compared.

Syntax

DBMS_METADATA_DIFF.ADD_DOCUMENT(
   handle IN NUMBER, document IN sys.XMLType);

DBMS_METADATA_DIFF.ADD_DOCUMENT(
   handle IN NUMBER, document IN CLOB);

Parameters

Table 101-4   ADD_DOCUMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from OPENC</td>
</tr>
<tr>
<td>document</td>
<td>A document to be compared. The document must be of the type specified in OPENC.</td>
</tr>
</tbody>
</table>

Usage Notes

Because the comparison interface allows you to compare exactly two SXML documents, a program must call ADD_DOCUMENT exactly twice for each OPENC handle. In the comparison result, the document specified by the first call is document 1, and the document specified by the second call is document 2.

Exceptions

• INVALID_ARGVAL
  
  A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.

101.4.3 FETCH_CLOB Functions and Procedures

The FETCH_CLOB functions and procedures return a CLOB showing the differences between the two documents specified by ADD_DOCUMENT.

Syntax

DBMS_METADATA_DIFF.FETCH_CLOB(
   handle IN NUMBER)
RETURN CLOB;

DBMS_METADATA_DIFF.FETCH_CLOB(
   handle IN NUMBER,
   doc    IN OUT NOCOPY CLOB);

DBMS_METADATA_DIFF.FETCH_CLOB(
   handle IN NUMBER,
   doc    IN OUT NOCOPY CLOB,
   diffs  OUT BOOLEAN);
Parameters

Table 101-5  FETCH_CLOB Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from <code>OPENC</code>.</td>
</tr>
<tr>
<td>doc</td>
<td>A CLOB containing the differences between documents 1 and 2.</td>
</tr>
<tr>
<td>diffs</td>
<td><code>TRUE</code> if the documents are different or <code>FALSE</code> if they are identical.</td>
</tr>
</tbody>
</table>

Return Values

The differences between documents 1 and 2.

Exceptions

- `INVALID_ARGVAL`
  
  A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.

101.4.4 CLOSE Procedure

This procedure invalidates the handle returned by `OPENC` and cleans up associated state.

Syntax

```sql
DBMS_METADATA_DIFF.CLOSE(
handle IN NUMBER);
```

Parameters

Table 101-6  CLOSE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle returned from <code>OPENC</code>.</td>
</tr>
</tbody>
</table>

Exceptions

- `INVALID_ARGVAL`
  
  A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
The `DBMS_MGD_ID_UTL` package contains various utility functions and procedures. These consist of the following utility subprograms:

- A logging utility that sets and gets Java and PL/SQL logging levels.
- A proxy utility consisting of two procedures used to set and unset the host and port of the proxy server.
- A metadata utility consisting of functions and procedures used for managing metadata.

### See Also:

*Oracle Database Development Guide* for more information.

This chapter describes each of these utility subprograms and contains the following topics:

- Security Model
- Constants
- Exceptions
- Summary of `DBMS_MGD_ID_UTL` Subprograms

The examples in this chapter assume that the user has run the following set of commands before running the contents of each script:

```
SQL> connect / as sysdba;
Connected.
SQL> create user mgduser identified by password
SQL> grant connect, resource to mgduser;
SQL> connect mgduser
Enter password: mgduserpassword
Connected.
SQL> set serveroutput on;
```

### 102.1 DBMS_MGD_ID_UTL Security Model

You must run the `catmgd.sql` script to load the `DBMS_MGD_ID_UTL` package and Identity Code Package schema objects in the `MGDSYS` schema.

`DBMS_MGD_ID_UTL` is a `MGDSYS`-owned package. Any `DBMS_MGD_ID_UTL` subprogram called from an anonymous PL/SQL block is run using the privileges of the current user.

A user must be granted connect and resource roles to use the `DBMS_MGD_ID_UTL` package and its subprograms.

**EXECUTE privilege** is granted to `PUBLIC` for these ADTs: `MGD_ID`, `MGD_ID_COMPONENT`, `MGD_ID_COMPONENT_VARRAY`, and for this package `DBMS_MGD_ID_UTL`. 

---

102-1
SELECT or READ privilege is granted to PUBLIC for these read-only views:
MGD_ID_CATEGORY and MGD_ID_SCHEMA and for these metadata views:
USER_MGD_ID_CATEGORY and USER_MGD_ID_SCHEMA, and for table
MGD_ID_XML_VALIDATOR, and for sequence MGD$SEQUENCE_CATEGORY.

INSERT, UPDATE and DELETE privilege is granted to PUBLIC for these metadata views:
USER_MGD_ID_CATEGORY and USER_MGD_ID_SCHEMA.

Public synonyms, by the same name, are created for these ADTs: MGD_ID,
MGD_ID_COMPONENT, MGD_ID_COMPONENT_VARRAY and for this package DBMS_MGD_ID_UTL,
as well as for these read-only views: MGD_ID_CATEGORY and MGD_ID_SCHEMA and for
these metadata views: USER_MGD_ID_CATEGORY and USER_MGD_ID_SCHEMA, and for table
MGD_ID_XML_VALIDATOR.

### 102.2 DBMS_MGD_ID_UTL Constants

The **DBMS_MGD_ID_UTL** package defines several constants for specifying parameter
values.

These constants are shown in the following tables.

**Table 102-1  ** DBMS_MGD_ID_UTL Constants — Installed Category IDs and Names

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC_ENCODING_CATEGORY_ID</td>
<td>1</td>
</tr>
<tr>
<td>EPC_ENCODING_CATEGORY_NAME</td>
<td>EPC</td>
</tr>
</tbody>
</table>

**Table 102-2  ** DBMS_MGD_ID_UTL Constants — Logging Levels

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGGING_LEVEL_OFF</td>
<td>0</td>
</tr>
<tr>
<td>LOGGING_LEVEL_SEVERE</td>
<td>1</td>
</tr>
<tr>
<td>LOGGING_LEVEL_WARNING</td>
<td>2</td>
</tr>
<tr>
<td>LOGGING_LEVEL_INFO</td>
<td>3</td>
</tr>
<tr>
<td>LOGGING_LEVEL_FINE</td>
<td>4</td>
</tr>
<tr>
<td>LOGGING_LEVEL_FINER</td>
<td>5</td>
</tr>
<tr>
<td>LOGGING_LEVEL_FINEST</td>
<td>6</td>
</tr>
<tr>
<td>LOGGING_LEVEL_ALL</td>
<td>7</td>
</tr>
</tbody>
</table>
### 102.3 DBMS_MGD_ID_UTL Exceptions

The table in this topic lists the **DBMS_MGD_ID_UTL** exceptions.

#### Table 102-3    Exceptions Raised by DBMS_MGD_ID_UTL Package

<table>
<thead>
<tr>
<th>Name</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDTJavaException</td>
<td>-55200</td>
<td>During the tag data translation, a Java exception was raised.</td>
</tr>
<tr>
<td>TDTCategoryNotFound</td>
<td>-55201</td>
<td>The specified category was not found.</td>
</tr>
<tr>
<td>TDTSchemeNotFound</td>
<td>-55202</td>
<td>During the tag data translation, the specified scheme was not found.</td>
</tr>
<tr>
<td>TDTLevelNotFound</td>
<td>-55203</td>
<td>During the tag data translation, the specified level was not found.</td>
</tr>
<tr>
<td>TDTOptionNotFound</td>
<td>-55204</td>
<td>During the tag data translation, the specified option was not found.</td>
</tr>
<tr>
<td>TDTFieldValidationException</td>
<td>-55205</td>
<td>During the tag data translation, the validation operation failed on a field.</td>
</tr>
<tr>
<td>TDTUndefinedField</td>
<td>-55206</td>
<td>During the tag data translation, an undefined field was detected.</td>
</tr>
<tr>
<td>TDTRuleEvaluationFailed</td>
<td>-55207</td>
<td>During the tag data translation, the rule evaluation operation failed.</td>
</tr>
<tr>
<td>TDTTooManyMatchingLevels</td>
<td>-55208</td>
<td>During the tag data translation, too many matching levels were found.</td>
</tr>
</tbody>
</table>

### 102.4 Summary of DBMS_MGD_ID_UTL Subprograms

This table describes the utility subprograms in the **DBMS_MGD_ID_UTL** package.

All the values and names passed to the procedures defined in the **DBMS_MGD_ID_UTL** package are case insensitive unless otherwise mentioned. To preserve the case, enclose the values with double quotation marks.

#### Table 102-4    DBMS_MGD_ID_UTL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_SCHEME Procedure</td>
<td>Adds a tag data translation scheme to an existing category</td>
</tr>
<tr>
<td>CREATE_CATEGORY Function</td>
<td>Creates a new category or a new version of a category</td>
</tr>
<tr>
<td>EPC_TO_ORACLE_SCHEME Function</td>
<td>Converts the EPCglobal tag data translation (TDT) XML to Oracle tag data translation XML</td>
</tr>
<tr>
<td>GET_CATEGORY_ID Function</td>
<td>Returns the category ID given the category name and the category version</td>
</tr>
<tr>
<td>GET_COMPONENTS Function</td>
<td>Returns all relevant separated component names separated by semicolon (<code>;</code>) for the specified scheme</td>
</tr>
<tr>
<td>GET_ENCODINGS Function</td>
<td>Returns a list of semicolon (<code>;</code>) separated encodings (formats) for the specified scheme</td>
</tr>
</tbody>
</table>
### Table 102-4  (Cont.) DBMS_MGD_ID_UTL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_JAVA_LOGGING_LEVEL Function</td>
<td>Returns an integer representing the current Java trace logging level</td>
</tr>
<tr>
<td>GET_PLSQL_LOGGING_LEVEL Function</td>
<td>Returns an integer representing the current PL/SQL trace logging level</td>
</tr>
<tr>
<td>GET_SCHEME_NAMES Function</td>
<td>Returns a list of semicolon (<code>;</code>) separated scheme names for the specified category</td>
</tr>
<tr>
<td>GET_TDT_XML Function</td>
<td>Returns the Oracle tag data translation XML for the specified scheme</td>
</tr>
<tr>
<td>GET_VALIDATOR Function</td>
<td>Returns the Oracle Database tag data translation schema</td>
</tr>
<tr>
<td>REFRESH_CATEGORY Function</td>
<td>Refreshes the metadata information on the Java stack for the specified category</td>
</tr>
<tr>
<td>REMOVE_CATEGORY Procedure</td>
<td>Removes a category including all the related TDT XML if the value of <code>category_version</code> parameter is NULL</td>
</tr>
<tr>
<td>REMOVE_PROXY Procedure</td>
<td>Unsets the host and port of the proxy server</td>
</tr>
<tr>
<td>REMOVE_SCHEME Procedure</td>
<td>Removes a tag data translation scheme from a category</td>
</tr>
<tr>
<td>SET_JAVA_LOGGING_LEVEL Procedure</td>
<td>Sets the Java logging level</td>
</tr>
<tr>
<td>SET_PLSQL_LOGGING_LEVEL Procedure</td>
<td>Sets the PL/SQL tracing logging level</td>
</tr>
<tr>
<td>SET_PROXY Procedure</td>
<td>Sets the host and port of the proxy server for Internet access</td>
</tr>
<tr>
<td>VALIDATE_SCHEME Function</td>
<td>Validates the input tag data translation XML against the Oracle tag data translation schema</td>
</tr>
</tbody>
</table>

### 102.4.1 ADD_SCHEME Procedure

This procedure adds a tag data translation scheme to an existing category.

**Syntax**

```sql
DBMS_MGD_ID_UTL.ADD_SCHEME ( 
    category_id IN VARCHAR2, 
    tdt_xml    IN CLOB); 
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
<tr>
<td>tdt_xml</td>
<td>Tag data translation XML</td>
</tr>
</tbody>
</table>

**Examples**

This example performs the following actions:

1. Creates a category.
Chapter 102

Summary of DBMS_MGD_ID_UTL Subprograms

2.

Adds a contractor scheme and an employee scheme to the MGD_SAMPLE_CATEGORY
category.

3.

Validates the MGD_SAMPLE_CATEGORY scheme.

4.

Tests the tag translation of the contractor scheme and the employee scheme.

5.

Removes the contractor scheme.

6.

Tests the tag translation of the contractor scheme and this returns the expected exception
for the removed contractor scheme.

7.

Tests the tag translation of the employee scheme and this returns the expected values.

8.

Removes the MGD_SAMPLE_CATEGORY category.

--contents of add_scheme2.sql
SET LINESIZE 160
-----------------------------------------------------------------------CREATE CATEGORY, ADD_SCHEME, REMOVE_SCHEME, REMOVE_CATEGORY--------------------------------------------------------------------------DECLARE
amt
NUMBER;
buf
VARCHAR2(32767);
pos
NUMBER;
tdt_xml
CLOB;
validate_tdtxml VARCHAR2(1042);
category_id VARCHAR2(256);
BEGIN
-- remove the testing category if already existed
DBMS_MGD_ID_UTL.remove_category('MGD_SAMPLE_CATEGORY', '1.0');
-- Step 1. Create the testing category 'MGD_SAMPLE_CATEGORY', version 1.0.
category_id := DBMS_MGD_ID_UTL.CREATE_CATEGORY('MGD_SAMPLE_CATEGORY', '1.0', 'Oracle',
'http://www.example.com/mgd/sample');
-- Step 2. Add contractor scheme to the category.
DBMS_LOB.CREATETEMPORARY(tdt_xml, true);
DBMS_LOB.OPEN(tdt_xml, DBMS_LOB.LOB_READWRITE);
buf := '<?xml version="1.0" encoding="UTF-8"?>
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmlns="oracle.mgd.idcode">
<scheme name="CONTRACTOR_TAG" optionKey="1" xmlns="">
<level type="URI" prefixMatch="example.contractor.">
<option optionKey="1" pattern="example.contractor.([0-9]*).([0-9]*)"
grammar="''example.contractor.'' contractorID ''.'' divisionID">
<field seq="1" characterSet="[0-9]*" name="contractorID"/>
<field seq="2" characterSet="[0-9]*" name="divisionID"/>
</option>
</level>
<level type="BINARY" prefixMatch="11">
<option optionKey="1" pattern="11([01]{7})([01]{6})"
grammar="''11'' contractorID divisionID ">
<field seq="1" characterSet="[01]*" name="contractorID"/>
<field seq="2" characterSet="[01]*" name="divisionID"/>
</option>
</level>
</scheme>
</TagDataTranslation>';
amt := length(buf);
pos := 1;
DBMS_LOB.WRITE(tdt_xml, amt, pos, buf);

102-5


<xml version="1.0" encoding="UTF-8">
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmlns="oracle.mgd.idcode">
<scheme name="EMPLOYEE_TAG" optionKey="1" xmlns="">
<level type="URI" prefixMatch="example.employee.">
<option optionKey="1" pattern="example.employee.([0-9]*).([0-9]*)"
grammar="''example.employee.'' employeeID ''.'' divisionID">
</level>
<level type="BINARY" prefixMatch="01">
<option optionKey="1" pattern="01([01]{7})([01]{6})"
grammar="''01'' employeeID divisionID">
</option>
</level>
</scheme>
<TagDataTranslation/>
</xml>
'011111011101101',
NULL, 'URI'));

DBMS_MGD_ID_UTL.REMOVE_SCHEME(category_id, 'CONTRACTOR_TAG');

-- Step 6. Test tag translation of contractor scheme. Doesn't work any more.
BEGIN
  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    'example.contractor.123.45',
    NULL, 'BINARY'));

  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    '111111011101101',
    NULL, 'URI'));
EXCEPTION
  WHEN others THEN
    dbms_output.put_line('Contractor tag translation failed: '||SQLERRM);
END;

BEGIN
  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    'example.employee.123.45',
    NULL, 'BINARY'));

  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    '011111011101101',
    NULL, 'URI'));
EXCEPTION
  WHEN others THEN
    dbms_output.put_line('Employee tag translation failed: '||SQLERRM);
END;

-- Step 8. Remove the testing category, which also removes all the associated schemes
DBMS_MGD_ID_UTL.remove_category('MGD_SAMPLE_CATEGORY', '1.0');
END;
/
SHOW ERRORS;

SQL> @add_scheme3.sql
.
.
.
Validate the MGD_SAMPLE_CATEGORY Scheme
EMPLOYEE_TAG;URI,BINARY;divisionID,employeeID
Length of scheme xml is: 933
111111011101101 example.contractor.123.45
011111011101101 example.employee.123.45
Contractor tag translation failed: ORA-55203: Tag data translation level not found
ORA-06512: at "MGDSYS.DSMS_MGD_ID_UTL", line 54
ORA-06512: at "MGDSYS.MGD_ID", line 242
ORA-29532: Java call terminated by uncaught Java exception: oracle.mgd.idcode.exceptions.TDTLevelNotFound: Matching level not found for any configured scheme
011111011101101 example.employee.123.45
102.4.2 CREATE_CATEGORY Function

This function creates a new category or a new version of a category.

Syntax

```
DBMS_MGD_ID_UTL.CREATE_CATEGORY (  
    category_name    IN  VARCHAR2,  
    category_version IN  VARCHAR2,  
    agency           IN  VARCHAR2,  
    URI              IN  VARCHAR2)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_name</td>
<td>Name of category</td>
</tr>
<tr>
<td>category_version</td>
<td>Category version</td>
</tr>
<tr>
<td>agency</td>
<td>Organization that owns the category. For example, EPCglobal owns the category EPC.</td>
</tr>
<tr>
<td>URI</td>
<td>URI that provides additional information about the category</td>
</tr>
</tbody>
</table>

Usage Notes

The return value is the category ID.

Examples

See the ADD_SCHEME Procedure for an example of creating the MGD_SAMPLECATEGORY category.

102.4.3 EPC_TO_ORACLE_SCHEME Function

This function converts the EPCglobal tag data translation (TDT) XML to Oracle Database tag data translation XML.

Syntax

```
DBMS_MGD_ID_UTL.EPC_TO_ORACLE_SCHEME (  
    xml_scheme IN CLOB)  
RETURN CLOB;
```
Parameters

Table 102-7  EPC_TO_ORACLE_SCHEME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml_scheme</td>
<td>Name of EPC tag scheme to be converted</td>
</tr>
</tbody>
</table>

Usage Notes

The return value is the contents of the CLOB containing the Oracle Datanase tag data translation XML.

Examples

The following example converts standard EPCglobal Tag Data Translation (TDT) files into Oracle Database TDT files:

```sql
--Contents of MGD_ID_DOC2.sql
-------------------------------
-- EPC_TO_ORACLE_SCHEME --
-------------------------------
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
BEGIN
    DBMS_JAVA.set_output(1000000);
    DBMS_OUTPUT.ENABLE(1000000);
    DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_SEVERE);
END;
/
DECLARE
epcScheme CLOB;
oracleScheme CLOB;
amt NUMBER;
buf VARCHAR2(32767);
pos NUMBER;
seq BINARY_INTEGER;
validate_epcscheme VARCHAR2(256);
validate_oraclescheme VARCHAR2(256);
BEGIN
    DBMS_LOB.CREATETEMPORARY(epcScheme, true);
    DBMS_LOB.OPEN(epcScheme, DBMS_LOB.LOB_READWRITE);
    buf := '<?xml version="1.0" encoding="UTF-8"?>
<epcTagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
    epcTDSVersion="1.1r1.27"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema"
    xsi:noNamespaceSchemaLocation="EpcTagDataTranslation.xsd">
    <scheme name="GID-96" optionKey="1" tagLength="96">
        <level type="BINARY" prefixMatch="00110101"
            requiredFormattingParameters="taglength">
            <option optionKey="1" pattern="00110101{{01}{28}}{{01}{24}}{{01}{36}}"
                grammar="''00110101'' generalmanager objectclass serial">
                <field seq="1" decimalMinimum="0" decimalMaximum="268435455"
                    characterSet="01" bitLength="28" name="generalmanager"/>
                <field seq="2" decimalMinimum="0" decimalMaximum="16777215"
                    characterSet="01" bitLength="24" name="objectclass"/>
```
Chapter 102
Summary of DBMS_MGD_ID_UTL Subprograms

PL/SQL procedure successfully completed.
Length of oracle scheme xml is: 2475
<?xml version = '1.0' encoding = 'UTF-8'>
<TagDataTranslation version="0.04"
date="2005-04-18T16:05:00Z" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmns="oracle.mgd.idcode">\scheme name="GID-96" optionKey="1" xmlns=""/>\level
type="BINARY" prefixMatch="00110101" requiredFormattingParameters=""/>\option
optionKey="1" pattern="00110101((01)(28))((01)(24))((01)(36))" grammar="00110101' generalmanager objectclass serial"><field seq="1" decimalMinimum="0" decimalMaximum="268435455" characterSet="""" bitLength="28" name="generalmanager"/>
<option seq="2" decimalMinimum="0" decimalMaximum="16777215" characterSet="""" bitLength="24"
name="objectclass"/>\field seq="3" decimalMinimum="0" decimalMaximum="68719476735"
name="serial"/>
</level>

Validate the Oracle Scheme
Validation result:
GID-96;LEGACY,TAG_ENCODING,PURE_IDENTITY,BINARY;objectclass,generalmanager,serial,
PL/SQL procedure successfully completed.

102.4.4 GET_CATEGORY_ID Function
This function returns the category ID for a given category name and category version.

Syntax

```
DBMS_MGD_ID_UTL.GET_CATEGORY_ID (  
    category name IN VARCHAR2,  
    category_version IN VARCHAR2)  
RETURN VARCHAR2;
```
Parameters

Table 102-8 GET_CATEGORY_ID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_name</td>
<td>Name of category</td>
</tr>
<tr>
<td>category_version</td>
<td>Category version</td>
</tr>
</tbody>
</table>

Usage Notes

- If the value of category_version is NULL, then the ID of the latest version of the specified category is returned.
- The return value is the category ID for the specified category name.

Examples

The following example returns a category ID given a category name and its version:

```sql
-- Contents of get_category1.sql file
SELECT DBMS_MGD_ID_UTL.get_category_id('EPC', NULL) FROM DUAL;
```

```bash
SQL> @get_category1.sql
.
.
.
```

102.4.5 GET_COMPONENTS Function

This function returns all relevant separated component names separated by semicolon (;) for the specified scheme.

Syntax

```sql
DBMS_MGD_ID_UTL.GET_COMPONENTS (  
  category_id IN VARCHAR2,  
  scheme_name IN VARCHAR2)  
RETURN VARCHAR2;
```

Parameters

Table 102-9 GET_COMPONENTS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
<tr>
<td>scheme_name</td>
<td>Name of scheme</td>
</tr>
</tbody>
</table>
Usage Notes

The return value contains the component names separated by a semicolon (;) for the specified scheme.

Examples

The following example gets the components:

```sql
--Contents of get_components.sql
DECLARE
    id             mgd_id;
    getcomps       VARCHAR2(1000);
    getencodings   VARCHAR2(1000);
    getschemenames VARCHAR2(1000);
BEGIN
    DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);
    DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
    getcomps := DBMS_MGD_ID_UTL.get_components(1,'SGTIN-64');
    dbms_output.put_line('Component names are: ' || getcomps);
    getencodings := DBMS_MGD_ID_UTL.get_encodings(1,'SGTIN-64');
    dbms_output.put_line('Encodings are: ' || getencodings);
    getschemenames := DBMS_MGD_ID_UTL.get_scheme_names(1);
    dbms_output.put_line('Scheme names are: ' || getschemenames);
END;
/ 
SHOW ERRORS;

SQL> @get_components.sql
.
.
.
Component names are:
filter,gtin,companyprefixlength,companyprefix,companyprefixindex,itemref,serial
Encodings are: ONS_HOSTNAME,LEGACY,TAG_ENCODING,PURE_IDENTITY,BINARY
Scheme names are:
GIAI-64,GIAI-96,GID-96,GRAI-64,GRAI-96,SGLN-64,SGLN-96,SGTIN-64,SGTIN-96,SSCC-64
,SSCC-96,USDOD-64,USDOD-96
PL/SQL procedure successfully completed.
.
.
102.4.6 GET_ENCODINGS Function

This function returns a list of semicolon (;) separated encodings (formats) for the specified scheme.

Syntax

```
DBMS_MGD_ID_UTL.GET_ENCODINGS (category_id IN VARCHAR2, 
    scheme_name IN VARCHAR2) 
RETURN VARCHAR2;
```
Parameters

Table 102-10   GET_ENCODINGS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
<tr>
<td>scheme_name</td>
<td>Name of scheme</td>
</tr>
</tbody>
</table>

Usage Notes

The return value contains the encodings separated by a semicolon (;) for the specified scheme.

Examples

See the GET_COMPONENTS Function for an example.

102.4.7 GET_JAVA_LOGGING_LEVEL Function

This function returns an integer representing the current trace logging level.

Syntax

```
DBMS_MGD_ID_UTL.GET_JAVA_LOGGING_LEVEL
RETURN INTEGER;
```

Usage Notes

The return value is the integer value denoting the current Java logging level.

Examples

The following example gets the Java logging level.

```
--Contents of getjavalogginglevel.sql
DECLARE
    loglevel   NUMBER;
BEGIN
    DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);
    loglevel := DBMS_MGD_ID_UTL.get_java_logging_level();
    dbms_output.put_line('Java logging level = ' ||loglevel);
END;
/
SHOW ERRORS;
```

SQL> @getjavalogginglevel.sql
.
.
.
Java logging level = 0
PL/SQL procedure successfully completed.
.
.
.
102.4.8 GET_PLSQL_LOGGING_LEVEL Function

This function returns an integer representing the current PL/SQL trace logging level.

Syntax

```
DBMS_MGD_ID_UTL.GET_PLSQL_LOGGING_LEVEL
RETURN INTEGER;
```

Usage Notes

The return value is the integer value denoting the current PL/SQL logging level.

Examples

The following example gets the PL/SQL logging level.

```
--Contents of getplsqllogginglevel.sql
DECLARE
    loglevel   NUMBER;
BEGIN
    DBMS_MGD_ID_UTL.set_plsql_logging_level(0);
    loglevel := DBMS_MGD_ID_UTL.get_plsql_logging_level();
    dbms_output.put_line('PL/SQL logging level = ' ||loglevel);
END;
/
SHOW ERRORS;
```

SQL> @getplsqllogginglevel.sql
.
.
.
PL/SQL logging level = 0
PL/SQL procedure successfully completed.
.
.
.

102.4.9 GET_SCHEME_NAMES Function

This function returns a list of semicolon (;) separated scheme names for the specified category.

Syntax

```
DBMS_MGD_ID_UTL.GET_SCHEME_NAMES (
    category_id IN VARCHAR2)
RETURN VARCHAR2;
```
Parameters

Table 102-11 GET_SCHEME_NAMES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
</tbody>
</table>

Usage Notes
The return value contains the scheme names for the specified category ID.

Examples
See the GET_COMPONENTS Function for an example.

102.4.10 GET_TDT_XML Function

This function returns the Oracle Database tag data translation XML for the specified scheme.

Syntax

```sql
DBMS_MGD_ID_UTL.GET_TDT_XML (  
category_id IN VARCHAR2,  
scheme_name IN VARCHAR2)  
RETURN CLOB;
```

Parameters

Table 102-12 GET_TDT_XML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
<tr>
<td>scheme_name</td>
<td>Name of scheme</td>
</tr>
</tbody>
</table>

Usage Notes
The return value contains the Oracle Database tag data translation XML for the specified scheme.

Examples
The following example gets the Oracle Database TDT XML for the specified scheme:

```sql
--Contents of get_tdtxml.sql
DECLARE
  gettdxml CLOB;
BEGIN
  gettdxml := DBMS_MGD_ID_UTL.get_tdt_xml(1, 'SGTIN-64');
  dbms_output.put_line('Length of tdt XML is ' || DBMS_LOB.GETLENGTH(gettdxml));
  dbms_output.put_line(DBMS_LOB.SUBSTR(gettdxml, DBMS_LOB.GETLENGTH(gettdxml), 1));
END;
/
```
SHOW ERRORS;

SQL> @get_tdtxml.sql
.
.
.
Length of tdt XML is 22884
<?xml version = '1.0' encoding = "UTF-8"?>
<TagDataTranslation version="0.04"
date="2005-04-18T16:05:00Z" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xnls="oracle.mgd.idcode">scheme name="SGTIN-64"
optionKey="companyprefixlength" xmlns="">
  <level type="BINARY"
prefixMatch="10" requiredFormattingParameters="filter">
    <option
optionKey="12" pattern="10([01]{3})([01]{14})([01]{20})([01]{25})" grammar="'10'
filter companyprefixindex itemref serial">
      <field seq="1"
        decimalMinimum="0" decimalMaximum="7" characterSet="[01]*" bitLength="3"
        length="1" padChar="0" padDir="LEFT" name="filter"/>
      <field seq="2"
        decimalMinimum="0" decimalMaximum="16383" characterSet="[01]*" bitLength="14"
        name="companyprefixindex"/>
      <field seq="3" decimalMinimum="0" decimalMaximum="9" characterSet="[01]*" bitLength="20" length="1" padChar="0"
        padDir="LEFT" name="itemref"/>
      <field seq="4" decimalMinimum="0" decimalMaximum="33554431" characterSet="[01]*" bitLength="25" length="1" name="serial"/>
      .
      .
      .
      <field seq="1" decimalMinimum="0" decimalMaximum="9999999" characterSet="[0-9]*" bitLength="7" length="1" padChar="0" padDir="LEFT" name="itemref"/>
      <field seq="2" decimalMinimum="0" decimalMaximum="9999999" characterSet="[0-9]*" length="6"
        padChar="0" padDir="LEFT" name="companyprefix"/>
    </option>
  </level>
</TagDataTranslation>

PL/SQL procedure successfully completed.
.
.
.

102.4.11 GET_VALIDATOR Function

This function returns the Oracle Database tag data translation schema.

Syntax

```
DBMS_MGD_ID UTL.GET_VALIDATOR
RETURN CLOB;
```

Usage Notes

The return value contains the Oracle Database tag data translation schema.

Examples

This example returns the Oracle Database TDT schema.
DECLARE
    getvalidator CLOB;
BEGIN
    getvalidator := DBMS_MGD_ID_UTL.get_validator;
    dbms_output.put_line('Length of validated oracle scheme xml is '||
        DBMS_LOB.GETLENGTH(getvalidator));
    dbms_output.put_line(DBMS_LOB.SUBSTR(getvalidator, DBMS_LOB.GETLENGTH(getvalidator), 1));
END;
/
SHOW ERRORS;

SQL> @get_validator.sql
.
.
.
Length of validated oracle scheme xml is 5780
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="oracle.mgd.idcode"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tdt="oracle.mgd.idcode" elementFormDefault="unqualified"
attributeFormDefault="unqualified" version="1.0">
<xsd:annotation>
</xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="LevelTypeList">
</xsd:restriction base="xsd:string">
</xsd:simpleType>
<xsd:simpleType name="TagLengthList">
</xsd:restriction base="xsd:string">
</xsd:simpleType>
<xsd:simpleType name="SchemeNameList">
    <xsd:restriction base="xsd:string">
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="InputFormatList">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="BINARY"/>
        <xsd:enumeration value="STRING"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ModeList">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="EXTRACT"/>
        <xsd:enumeration value="FORMAT"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="CompactionMethodList">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="32-bit"/>
        <xsd:enumeration value="16-bit"/>
        <xsd:enumeration value="8-bit"/>
        <xsd:enumeration value="7-bit"/>
        <xsd:enumeration value="6-bit"/>
        <xsd:enumeration value="5-bit"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PadDirectionList">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="LEFT"/>
        <xsd:enumeration value="RIGHT"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="Field">
    <xsd:attribute name="seq" type="xsd:integer" use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="bitLength" type="xsd:integer"/>
    <xsd:attribute name="characterSet" type="xsd:string" use="required"/>
    <xsd:attribute name="compaction" type="tdt:CompactionMethodList"/>
    <xsd:attribute name="compression" type="xsd:string"/>
    <xsd:attribute name="padChar" type="xsd:string"/>
    <xsd:attribute name="padDir" type="tdt:PadDirectionList"/>
</xsd:complexType>
<xsd:attribute name="decimalMinimum" type="xsd:long"/>
</xsd:complexType>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:complexType>
Chapter 102

Summary of DBMS_MGD_ID_UTL Subprograms

102.4.12 REFRESH CATEGORY Function

This function refreshes the metadata information on the Java stack for the specified category.
This function must be called before using MGD_ID functions.

Syntax

```-plsql
DBMS_MGD_ID_UTL.REFRESH_CATEGORY (category_id  IN VARCHAR2);
```

Parameters

Table 102-13  REFRESH CATEGORY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
</tbody>
</table>
Examples

The following example refreshes the metadata information for the EPC category ID.

```sql
--Contents of tostring3.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
DECLARE
    id MGD_ID;
BEGIN
    DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);
    DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
    dbms_output.put_line('..Testing to_string');
    DBMS_OUTPUT.PUT_LINE('test to_string');
    id := mgd_id('EPC', NULL, 'urn:epc:id:gid:0037000.30241.1041970',
        'scheme=GID-96');
    DBMS_OUTPUT.PUT_LINE('mgd_id object as a string');
    DBMS_OUTPUT.PUT_LINE(id.to_string);
END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();
```

SQL> @tostring3.sql
..Testing to_string
test to_string
mgd_id object as a string
category_id =1;schemes = GID-96;objectclass = 30241;generalmanager = 0037000;scheme = GID-96;1 = 1;serial = 1041970

PL/SQL procedure successfully completed.

102.4.13 REMOVE_CATEGORY Procedure

This procedure removes a category including all the related TDT XML.

This procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

Removes a category based on the specified category ID.

```sql
DBMS_MGD_ID_UTL.REMOVE_CATEGORY (category_id IN VARCHAR2);
```

Removes a category based on the specified category name and category version.

```sql
DBMS_MGD_ID_UTL.REMOVE_CATEGORY (category_name IN VARCHAR2,
    category_version IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
</tbody>
</table>
### Table 102-14  (Cont.) REMOVE_CATEGORY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_name</td>
<td>Name of category</td>
</tr>
<tr>
<td>category_version</td>
<td>Category version</td>
</tr>
</tbody>
</table>

#### Usage Notes

If the value of `category_version` is NULL, all versions for the specified category will be removed.

#### Examples

See the [ADD_SCHEME Procedure](#) for an example of removing a category.

### 102.4.14 REMOVE_PROXY Procedure

This procedure unsets the host and port of the proxy server.

#### Syntax

```sql
DBMS_MGD_ID_UTL.REMOVE_PROXY;
```

#### Examples

See the [REFRESH_CATEGORY Function](#) for an example.

### 102.4.15 REMOVE_SCHEME Procedure

This procedure removes a tag data translation scheme from a category.

#### Syntax

```sql
DBMS_MGD_ID_UTL.REMOVE_SCHEME (  
    category_id  IN  VARCHAR2,  
    scheme_name  IN  VARCHAR2);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category ID</td>
</tr>
<tr>
<td>scheme_name</td>
<td>Name of scheme</td>
</tr>
</tbody>
</table>

#### Usage Notes

#### Examples

See the [ADD_SCHEME Procedure](#) for an example of removing a scheme.
102.4.16 SET_JAVA_LOGGING_LEVEL Procedure

This procedure sets the Java trace logging level.

Syntax

```
DBMS_MGD_ID_UTL.SET_JAVA_LOGGING_LEVEL (
    logginglevel IN INTEGER);
```

Parameters

Table 102-16  SET_JAVA_LOGGING_LEVEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logginglevel</td>
<td>Logging level. The Java logging level can be one of the following values in descending order:</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_OFF CONSTANT INTEGER := 0</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_SEVERE CONSTANT INTEGER := 1</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_WARNING CONSTANT INTEGER := 2</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_INFO CONSTANT INTEGER := 3</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_FINE CONSTANT INTEGER := 4</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_FINER CONSTANT INTEGER := 5</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_FINEST CONSTANT INTEGER := 6</td>
</tr>
<tr>
<td></td>
<td>- LOGGING_LEVEL_ALL CONSTANT INTEGER := 7</td>
</tr>
</tbody>
</table>

Examples

See the GET_JAVA_LOGGING_LEVEL Function for an example.

102.4.17 SET_PLSQL_LOGGING_LEVEL Procedure

This procedure sets the PL/SQL trace logging level.

Syntax

```
DBMS_MGD_ID_UTL.SET_PLSQL_LOGGING_LEVEL (
    level IN INTEGER);
```

PRAGMA restrict_references(set_plsql Logging level, WNDS);
Parameters

Table 102-17  SET_PLSQL_LOGGING_LEVEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>Logging level. The PL/SQL logging level can be one of the following values in descending order:</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL_OFF CONSTANT INTEGER := 0</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL_SEVERE CONSTANT INTEGER := 1</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL WARNING CONSTANT INTEGER := 2</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL_INFO CONSTANT INTEGER := 3</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL_FINE CONSTANT INTEGER := 4</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL_FINER CONSTANT INTEGER := 5</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL_FINEST CONSTANT INTEGER := 6</td>
</tr>
<tr>
<td></td>
<td>• LOGGING_LEVEL_ALL CONSTANT INTEGER := 7</td>
</tr>
</tbody>
</table>

Examples

See the GET_PLSQL_LOGGING_LEVEL Function for an example.

102.4.18 SET_PROXY Procedure

This procedure sets the host and port of the proxy server for Internet access.

This procedure must be called if the database server accesses the Internet using a proxy server. Internet access is necessary because some rules need to look up the Object Naming Service (ONS) table to get the company prefix index.

You do not need to call this procedure does if you are only using schemes that do not contain any rules requiring Internet access.

Syntax

```
DBMS_MGD_ID_UTL.SET_PROXY (  
  proxt_host IN VARCHAR2,  
  proxy_port IN VARCHAR2);  
```

Parameters

Table 102-18  SET_PROXY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy_host</td>
<td>Name of host</td>
</tr>
<tr>
<td>proxy_port</td>
<td>Host port number</td>
</tr>
</tbody>
</table>

Examples

See the REFRESHCATEGORY Function for an example.
102.4.19 VALIDATE_SCHEME Function

This function validates the input tag data translation XML against the Oracle Database tag data translation schema.

Syntax

```sql
DBMS_MGD_ID_UTL.VALIDATE_SCHEME (xml_scheme IN CLOB)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml_scheme</td>
<td>Scheme to be validated.</td>
</tr>
</tbody>
</table>

Usage Notes

The return value contains the components names for the specified scheme.

Examples

See the ADD_SCHEME Procedure or the EPC_TO_ORACLE_SCHEME Function for an example.
DBMS_MGWADM defines the Messaging Gateway administrative interface. The package and object types are owned by SYS.

**Note:**
You must run the `catmgw.sql` script to load the Messaging Gateway packages and types into the database.

**See Also:**
*Oracle Database Advanced Queuing User's Guide* contains information on loading database objects and using DBMS_MGWADM.

This chapter contains the following topics:

- Security Model
- Deprecated Subprograms
- Constants
- Data Structures
- Summary of DBMS_MGWADM Subprograms

### 103.1 DBMS_MGWADM Security Model

A user with administrative privilege can run all procedures in DBMS_MGWADM.

### 103.2 DBMS_MGWADM Deprecated Subprograms

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 11g Release 1 having been superseded by improved technology:

- **ADD_SUBSCRIBER** Procedure - use instead CREATE_JOB Procedure
- **ALTER_PROPAGATION_SCHEDULE** Procedure - use instead ALTER_JOB Procedure
- **ALTER_SUBSCRIBER** Procedure - use instead ALTER_JOB Procedure
- **DB_CONNECT_INFO** Procedure - use instead ALTER_AGENT Procedures
• **DISABLE_PROPAGATION_SCHEDULE** Procedure - use instead **DISABLE_JOB** Procedure
• **ENABLE_PROPAGATION_SCHEDULE** Procedure - use instead **ENABLE_JOB** Procedure
• **REMOVE_SUBSCRIBER** Procedure - use instead **REMOVE_JOB** Procedure
• **RESET_SUBSCRIBER** Procedure - use instead **RESET_JOB** Procedure
• **SCHEDULE_PROPAGATION** Procedure - use instead **CREATE_JOB** Procedure
• **UNSCHEDULE_PROPAGATION** Procedure - use instead **REMOVE_JOB** Procedure

# 103.3 DBMS_MGWADM Constants

The **DBMS_MGWADM** package defines various constants for specifying parameter values.

- Table 103-1
- Table 103-2
- Table 103-3
- Table 103-4
- Table 103-5
- Table 103-6
- Table 103-7
- Table 103-8
- Table 103-9
- Table 103-10
- Table 103-11

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAN_STARTUP_STATE</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Sets the Messaging Gateway agent to a known state so that it can be started</td>
</tr>
<tr>
<td>CLEAN_LOG_QUEUES</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Messaging Gateway agent will clean log queues for all configured messaging system links</td>
</tr>
<tr>
<td>RESET_SUBMISSING_LOG_REC</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing log record</td>
</tr>
<tr>
<td>RESET_SUBMISSING_MESSAG</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing persistent source message</td>
</tr>
</tbody>
</table>
### Table 103-2  DBMS_MGWADM Constants—Force Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents a forced action</td>
</tr>
<tr>
<td>NO_FORCE</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents a normal, nonforced action</td>
</tr>
</tbody>
</table>

### Table 103-3  DBMS_MGWADM Constants—Logging Levels

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC_LOGGING</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>The standard (the least) information written to the log file</td>
</tr>
<tr>
<td>TRACE_DEBUG_LOGGING</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>The greatest information written to the log file</td>
</tr>
<tr>
<td>TRACE_HIGH_LOGGING</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>The third level of detail of logging information written to the log file</td>
</tr>
<tr>
<td>TRACE_LITE_LOGGING</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>The second level detail of logging information written to the log file</td>
</tr>
</tbody>
</table>

### Table 103-4  DBMS_MGWADM Constants—Named Property Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGWPROP_PREFIX</td>
<td>CONSTANT VARCHAR2</td>
<td>A constant (MGWPROP$) for the reserved property name prefix</td>
</tr>
<tr>
<td>MGWPROP_REMOVE</td>
<td>CONSTANT VARCHAR2</td>
<td>A constant (MGWPROP$ REMOVE) for the reserved property name used to remove an existing property</td>
</tr>
<tr>
<td>MGWPROP_REMOVE_ALL</td>
<td>CONSTANT VARCHAR2</td>
<td>A constant (MGWPROP$ REMOVE ALL) for the reserved property name used to remove all properties</td>
</tr>
</tbody>
</table>

### Table 103-5  DBMS_MGWADM Constants—Other Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS_CONNECTION</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Used to indicate that JMS connections will be used to access JMS destinations</td>
</tr>
<tr>
<td>JMS_QUEUE_CONNECTION</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Used to indicate that JMS queue connections will be used to access JMS destinations</td>
</tr>
<tr>
<td>JMS_TOPIC_CONNECTION</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Used to indicate that JMS topic connections will be used to access JMS destinations</td>
</tr>
<tr>
<td>NO_CHANGE</td>
<td>CONSTANT VARCHAR2</td>
<td>Indicates that an existing value should be preserved (not changed). This is used for certain APIs where the desire is to change one or more parameters but leave others unchanged.</td>
</tr>
</tbody>
</table>
### Table 103-5  DBMS_MGWADM Constants—Other Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_AGENT</td>
<td>CONSTANT VARCHAR2</td>
<td>Name of the Messaging Gateway default agent</td>
</tr>
</tbody>
</table>

### Table 103-6  DBMS_MGWADM Constants—Propagation Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INBOUND_PROPAGATION</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents the propagation type for non-Oracle to Oracle Database Advanced Queuing propagation. The propagation source is a queue in a foreign (non-Oracle) messaging system and the destination is a local Oracle Database Advanced Queuing queue.</td>
</tr>
<tr>
<td>OUTBOUND_PROPAGATION</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents the propagation type for Oracle Database Advanced Queuing to non-Oracle propagation. The propagation source is a local Oracle Database Advanced Queuing queue and the destination is a queue in a foreign (non-Oracle) messaging system.</td>
</tr>
</tbody>
</table>

### Table 103-7  DBMS_MGWADM Constants—Queue Domain Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN_QUEUE</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents a queue destination. A JMS queue (point-to-point model) is classified as a queue.</td>
</tr>
<tr>
<td>DOMAIN_TOPIC</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents a topic destination. A JMS topic (publish-subscribe model) is classified as a topic.</td>
</tr>
</tbody>
</table>

### Table 103-8  DBMS_MGWADM Constants—Shutdown Modes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHUTDOWN_IMMEDIATE</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents the immediate shutdown mode</td>
</tr>
<tr>
<td>SHUTDOWN_NORMAL</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents the normal shutdown mode</td>
</tr>
</tbody>
</table>

### Table 103-9  DBMS_MGWADM Constants—WebSphere MQ Interface Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQSERIES_BASE_JAVA_INTERFACE</td>
<td>CONSTANT BINARY_INTEGER</td>
<td>Represents the Base Java interface for the WebSphere MQ messaging system</td>
</tr>
</tbody>
</table>
Table 103-10  DBMS_MGWADM Constants—target_type Argument of SET_OPTION and REMOVE_OPTION Procedures

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_JAVA_PROP</td>
<td>CONSTANT PLS_INTEGER</td>
<td>Used for an agent option used to set a Java System property</td>
</tr>
<tr>
<td>MSGLINK_OPTION</td>
<td>CONSTANT PLS_INTEGER</td>
<td>Used for a messaging system link option</td>
</tr>
<tr>
<td>JOB_OPTION</td>
<td>CONSTANT PLS_INTEGER</td>
<td>Used for a propagation job option</td>
</tr>
</tbody>
</table>

Table 103-11  DBMS_MGWADM Constants—conntype Argument of CREATE_AGENT and ALTER_AGENT Procedures

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC_OCI</td>
<td>CONSTANT VARCHAR2</td>
<td>Used to specify the JDBC OCI driver</td>
</tr>
<tr>
<td>JDBC_THIN</td>
<td>CONSTANT VARCHAR2</td>
<td>Used to specify the JDBC Thin driver</td>
</tr>
</tbody>
</table>

103.4 DBMS_MGWADM Data Structures

The DBMS_MGWADM package defines several OBJECT types.

DBMS_MGWADM Object Types

- SYS.MGW_MQSERIES_PROPERTIES Object Type
- SYS.MGW_PROPERTIES Object Type
- SYS.MGW_PROPERTY Object Type
- SYS.MGW_TIBRV_PROPERTIES Object Type

103.4.1 DBMS_MGWADM SYS.MGW_MQSERIES_PROPERTIES Object Type

This type specifies basic properties for a WebSphere MQ messaging system link.

Syntax

```
TYPE SYS.MGW_MQSERIES_PROPERTIES IS OBJECT (
queue_manager VARCHAR2(64),
hostname VARCHAR2(64),
port INTEGER,
channel VARCHAR2(64),
interface_type INTEGER,
max_connections INTEGER,
username VARCHAR2(64),
password VARCHAR2(64),
inbound_log_queue VARCHAR2(64),
outbound_log_queue VARCHAR2(64),

-- Methods
STATIC FUNCTION construct RETURN SYS.MGW_MQSERIES_PROPERTIES,
```
STATIC FUNCTION alter_construct
RETURN SYS.MGW_MQSERIES_PROPERTIES ;

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_manager</td>
<td>The name of the WebSphere MQ queue manager</td>
</tr>
<tr>
<td>hostname</td>
<td>The host on which the WebSphere MQ messaging system resides. If hostname is NULL, then a WebSphere MQ bindings connection is used. If not NULL, then a client connection is used and requires that a port and channel be specified.</td>
</tr>
<tr>
<td>port</td>
<td>The port number. This is used only for client connections; that is, when hostname is not NULL.</td>
</tr>
<tr>
<td>channel</td>
<td>The channel used when establishing a connection to the queue manager. This is used only for client connections; that is, when hostname is not NULL.</td>
</tr>
<tr>
<td>interface_type</td>
<td>The type of messaging interface to use. Values:</td>
</tr>
<tr>
<td></td>
<td>· DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE if the WebSphere MQ Base Java interface should be used.</td>
</tr>
<tr>
<td></td>
<td>· DBMS_MGWADM.JMS_CONNECTION if the link is to be used to access JMS destinations in a unified, domain-independent manner.</td>
</tr>
<tr>
<td></td>
<td>· DBMS_MGWADM.JMS_QUEUE_CONNECTION if the link is to be used for accessing JMS queues</td>
</tr>
<tr>
<td></td>
<td>· DBMS_MGWADM.JMS_TOPIC_CONNECTION if the link is to be used for accessing JMS topics.</td>
</tr>
<tr>
<td>max_connections</td>
<td>The maximum number of messaging connections to the WebSphere MQ messaging system</td>
</tr>
<tr>
<td>username</td>
<td>The username used for authentication to the WebSphere MQ messaging system</td>
</tr>
<tr>
<td>password</td>
<td>The password used for authentication to the WebSphere MQ messaging system</td>
</tr>
<tr>
<td>inbound_log_queue</td>
<td>The name of the WebSphere MQ queue used for propagation recovery purposes when this messaging link is used for inbound propagation; that is, when queues associated with this link serve as a propagation source:</td>
</tr>
<tr>
<td></td>
<td>· For MQSERIES_BASE_JAVA_INTERFACE, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</td>
</tr>
<tr>
<td></td>
<td>· For the JMS_CONNECTION interface and the JMS_QUEUE_CONNECTION interface, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</td>
</tr>
<tr>
<td></td>
<td>· For JMS_TOPIC_CONNECTION interface, this specifies the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by subscribers of that topic must be created using WebSphere MQ administration tools. By default, the physical queue used is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.</td>
</tr>
</tbody>
</table>
Table 103-12  (Cont.) SYS.MGW_MQSERIES_PROPERTIES Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outbound_log_queue</td>
<td>The name of the WebSphere MQ queue used for propagation recovery purposes when this messaging link is used for outbound propagation; that is, when queues associated with this link serve as a propagation destination:</td>
</tr>
<tr>
<td></td>
<td>· For MQSERIES_BASE_JAVA_INTERFACE, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</td>
</tr>
<tr>
<td></td>
<td>· For the JMS_CONNECTION interface and the JMS_QUEUE_CONNECTION interface, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</td>
</tr>
<tr>
<td></td>
<td>· For JMS_TOPIC_CONNECTION interface, this specifies the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by subscribers of that topic must be created using WebSphere MQ administration tools.</td>
</tr>
<tr>
<td></td>
<td>By default, the physical queue used is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.</td>
</tr>
</tbody>
</table>

Methods

Table 103-13  SYS.MGW_MQSERIES_PROPERTIES Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>construct</td>
<td>Constructs a new SYS.MGW_MQSERIES_PROPERTIES instance. All attributes are assigned a value of NULL.</td>
</tr>
<tr>
<td>alter_construct</td>
<td>Constructs a new SYS.MGW_MQSERIES_PROPERTIES instance for altering the properties of an existing messaging link. All attributes having a VARCHAR2 datatype are assigned a value of DBMS_MGWADM.NO_CHANGE. Attributes of other datatypes are assigned a value of NULL.</td>
</tr>
</tbody>
</table>

103.4.2 DBMS_MGWADM SYS.MGW_PROPERTIES Object Type

This type specifies an array of properties.

Syntax

```sql
TYPE SYS.MGW_PROPERTIES AS VARRAY (2000) OF SYS.MGW_PROPERTY;
```

Attributes

Table 103-14  SYS.MGW_PROPERTIES Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Property name</td>
</tr>
<tr>
<td>value</td>
<td>Property value</td>
</tr>
</tbody>
</table>
Usage Notes

Unless noted otherwise, Messaging Gateway uses named properties as follows:

- Names with the `MGWPROP$` prefix are reserved. They are used for special purposes and are invalid when used as a normal property name.
- A property name can exist only once in a property list; that is, a list can contain only one value for a given name. The name is case-insensitive.
- In general, a property list is order-independent, and the property names may appear in any order. An alter property list is an exception.
- You can use a new property list to alter an existing property list. Each new property modifies the original list in one of the following ways: adds a new property, modifies a property, removes a property, or removes all properties.

The alter list is processed in order, from the first element to the last element. Thus the order in which the elements appear in the alter list is meaningful, especially when the alter list is used to remove properties from an existing list.

The property name and value are used to determine how that element affects the original list. The following rules apply:

- Add or modify property
  ```
  MGW_PROPERTY.NAME = property_name
  MGW_PROPERTY.VALUE = property_value
  ```
  If a property of the given name already exists, then the current value is replaced with the new value; otherwise the new property is added to the end of the list.

- Remove property
  ```
  MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE'
  MGW_PROPERTY.VALUE = name_of_property_to_remove
  ```
  No action is taken if the property name does not exist in the original list.

- Remove all properties
  ```
  MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE_ALL'
  MGW_PROPERTY.VALUE = not used
  ```

See Also:

"The `DBMS_MGWADM` package defines constants to represent the reserved property names on Table 103-4"

103.4.3 DBMS_MGWADM SYS.MGW_PROPERTY Object Type

This type specifies a named property which is used to specify optional properties for messaging links, foreign queues, and subscribers.

Syntax

```
TYPE SYS.MGW_PROPERTY IS OBJECT(
  name  VARCHAR2(500),
```
value VARCHAR2(4000),

-- Methods
STATIC FUNCTION construct    --- (1)
RETURN SYS.MGW_PROPERTY,

STATIC FUNCTION construct(   --- (2)
  p_name   IN VARCHAR2,
  p_value  IN VARCHAR2)
RETURN SYS.MGW_PROPERTY );

Attributes

Table 103-15  SYS.MGWPROPERTY Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Property name</td>
</tr>
<tr>
<td>value</td>
<td>Property value</td>
</tr>
</tbody>
</table>

Methods

Table 103-16  SYS.MGWPROPERTY Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>construct</td>
<td>Constructs a new MGW_PROPERTY instance. All attributes are assigned a value of NULL</td>
</tr>
<tr>
<td>construct</td>
<td>Constructs a new MGW_PROPERTY instance initialized using the given parameters</td>
</tr>
</tbody>
</table>

See Also:

"The DBMS_MGWADM package defines constants to represent the reserved property names on Table 103-4"

103.4.4 SYS.MGW_TIBRV_PROPERTIES Object Type

A type that specifies basic properties for a TIB/Rendezvous messaging system link. The Messaging Gateway agent creates a TIB/Rendezvous transport of type TibrvRvdTransport for each Messaging Gateway link.

Syntax

TYPE SYS.MGW_TIBRV_PROPERTIES IS OBJECT(
  service   VARCHAR2(128),
  daemon    VARCHAR2(128),
  network   VARCHAR2(256),
  cm_name   VARCHAR2(256),
  cm_ledger VARCHAR2(256),

  -- Methods
  STATIC FUNCTION construct

...
RETURN SYS.MGW_TIBRV_PROPERTIES,

STATIC FUNCTION alter_construct
RETURN SYS.MGW_TIBRV_PROPERTIES );

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>The service parameter for the rvd transport</td>
</tr>
<tr>
<td>daemon</td>
<td>The daemon parameter for the rvd transport</td>
</tr>
<tr>
<td>network</td>
<td>The network parameter for the rvd transport</td>
</tr>
<tr>
<td>cm_name</td>
<td>The CM correspondent name. Reserved for future use.</td>
</tr>
<tr>
<td>cm_ledger</td>
<td>The CM ledger file name. Reserved for future use.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>construct</td>
<td>Constructs a new SYS.MGW_TIBRV_PROPERTIES instance. All attributes will be assigned a value of NULL.</td>
</tr>
<tr>
<td>alter_construct</td>
<td>Constructs a new SYS.MGW_TIBRV_PROPERTIES instance. This function is useful for altering the properties of an existing messaging link. All attributes having a VARCHAR2 datatype will be assigned a value of DBMS_MGWADM.NO_CHANGE. Attributes of other datatypes will be assigned a value of NULL.</td>
</tr>
</tbody>
</table>

103.5 Summary of DBMS_MGWADM Subprograms

This table lists the DBMS_MGWADM subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_SUBSCRIBER Procedure</td>
<td>Adds a subscriber used to consume messages from a source queue for propagation to a destination</td>
</tr>
<tr>
<td>ALTER_AGENT Procedures</td>
<td>Alters Messaging Gateway agent parameters</td>
</tr>
<tr>
<td>ALTER_JOB Procedure</td>
<td>Alters the properties of a propagation job</td>
</tr>
<tr>
<td>ALTER_MGSYSTEM_LINK Procedure for TIB/Rendezvous</td>
<td>Alters the properties of a TIB/Rendezvous messaging system link</td>
</tr>
<tr>
<td>ALTER_MGSYSTEM_LINK Procedure for WebSphere MQ</td>
<td>Alters the properties of a WebSphere MQ messaging system link</td>
</tr>
<tr>
<td>ALTER_PROPAGATION_SCHEDULE Procedure</td>
<td>Alters a propagation schedule</td>
</tr>
</tbody>
</table>
## Table 103-19  (Cont.) DBMS_MGWADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_SUBSCRIBER Procedure</td>
<td>Alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination</td>
</tr>
<tr>
<td>CLEANUP_GATEWAY Procedures</td>
<td>Cleans up Messaging Gateway</td>
</tr>
<tr>
<td>CREATE_AGENT Procedure</td>
<td>Creates a Messaging Gateway agent that will be used to process propagation jobs</td>
</tr>
<tr>
<td>CREATE_JOB Procedure</td>
<td>Creates a job used to propagate message from a source to a destination</td>
</tr>
<tr>
<td>CREATE_MSGSYSTEM_LINK Procedures for TIB/Rendezvous</td>
<td>Creates a messaging system link to a TIB/Rendezvous messaging system</td>
</tr>
<tr>
<td>CREATE_MSGSYSTEM_LINK Procedures for WebSphere MQ</td>
<td>Creates a messaging system link to a WebSphere MQ messaging system</td>
</tr>
<tr>
<td>DB_CONNECT_INFO Procedure</td>
<td>Configures connection information used by the Messaging Gateway agent for connections to Oracle Database</td>
</tr>
<tr>
<td>DISABLE_JOB Procedure</td>
<td>Disables a propagation job</td>
</tr>
<tr>
<td>DISABLE_PROPAGATION_SCHEDULE Procedure</td>
<td>Disables a propagation schedule</td>
</tr>
<tr>
<td>ENABLE_JOB Procedure</td>
<td>Enables a propagation job</td>
</tr>
<tr>
<td>ENABLE_PROPAGATION_SCHEDULE Procedure</td>
<td>Enables a propagation schedule</td>
</tr>
<tr>
<td>REGISTER_FOREIGN_QUEUE Procedure</td>
<td>Registers a non-Oracle queue entity in Messaging Gateway</td>
</tr>
<tr>
<td>REMOVE_AGENT Procedure</td>
<td>Removes a Messaging Gateway agent</td>
</tr>
<tr>
<td>REMOVE_JOB Procedure</td>
<td>Removes a propagation job</td>
</tr>
<tr>
<td>REMOVE_MSGSYSTEM_LINK Procedure</td>
<td>Removes a messaging system link for a non-Oracle messaging system</td>
</tr>
<tr>
<td>REMOVE_OPTION Procedure</td>
<td>Removes a Messaging Gateway agent configuration option</td>
</tr>
<tr>
<td>REMOVE_SUBSCRIBER Procedure</td>
<td>Removes a subscriber used to consume messages from a source queue for propagation to a destination</td>
</tr>
<tr>
<td>RESET_JOB Procedure</td>
<td>Resets the propagation error state for a propagation job</td>
</tr>
<tr>
<td>RESET_SUBSCRIBER Procedure</td>
<td>Resets the propagation error state for a subscriber</td>
</tr>
<tr>
<td>SCHEDULE_PROPAGATION Procedure</td>
<td>Schedules message propagation from a source to a destination</td>
</tr>
<tr>
<td>SET_LOG_LEVEL Procedures</td>
<td>Dynamically alters the Messaging Gateway agent logging level</td>
</tr>
<tr>
<td>SET_OPTION Procedure</td>
<td>Sets a Messaging Gateway configuration option</td>
</tr>
<tr>
<td>SHUTDOWN Procedures</td>
<td>Shuts down the Messaging Gateway agent</td>
</tr>
<tr>
<td>STARTUP Procedures</td>
<td>Starts the Messaging Gateway agent</td>
</tr>
<tr>
<td>UNREGISTER_FOREIGN_QUEUE Procedure</td>
<td>Removes a non-Oracle queue entity in Messaging Gateway</td>
</tr>
<tr>
<td>UNSCHEDULE_PROPAGATION Procedure</td>
<td>Removes a propagation schedule</td>
</tr>
</tbody>
</table>
103.5.1 ADD_SUBSCRIBER Procedure

This procedure adds a subscriber used to consume messages from a source queue for propagation to a destination.

Note:
This subprogram has been deprecated as a result of improved technology (see CREATE_JOB Procedure), and is retained only for reasons of backward compatibility.

Syntax

```sql
DBMS_MGWADM.ADD_SUBSCRIBER(
    subscriber_id     IN VARCHAR2,
    propagation_type  IN BINARY_INTEGER,
    queue_name        IN VARCHAR2,
    destination       IN VARCHAR2,
    rule              IN VARCHAR2 DEFAULT NULL,
    transformation    IN VARCHAR2 DEFAULT NULL,
    exception_queue   IN VARCHAR2 DEFAULT NULL,
    options           IN SYS.MGW_PROPERTIES DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriber_id</td>
<td>Specifies a user-defined name that identifies this subscriber.</td>
</tr>
<tr>
<td>propagation_type</td>
<td>Specifies the type of message propagation. DBMS_MGWADM.OUTBOUND_PROPAGATION is for Oracle Database Advanced Queuing to non-Oracle propagation. DBMS_MGWADM.INBOUND_PROPAGATION is for non-Oracle to Oracle Database Advanced Queuing propagation.</td>
</tr>
<tr>
<td>queue_name</td>
<td>Specifies the source queue to which this subscriber is being added. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
<tr>
<td>destination</td>
<td>Specifies the destination queue to which messages consumed by this subscriber are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
<tr>
<td>rule</td>
<td>Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. This is NULL if no rule is needed. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
</tbody>
</table>
Table 103-20  (Cont.) ADD_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transformation</td>
<td>Specifies the transformation needed to convert between the Oracle Database Advanced Queuing payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the value specified for propagation_type. If NULL, then the Oracle Database Advanced Queuing payload type must be supported by Messaging Gateway.</td>
</tr>
<tr>
<td>exception_queue</td>
<td>Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. If NULL, then an exception queue is not used and propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the value specified for propagation_type. The source queue and exception queue cannot be the same queue.</td>
</tr>
<tr>
<td>options</td>
<td>Optional subscriber properties. NULL if there are none. Typically these are lesser used configuration properties supported by the messaging system.</td>
</tr>
</tbody>
</table>

**Usage Notes**

**See Also:**


If the non-Oracle messaging link being accessed for the subscriber uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Database Advanced Queuing queues. Otherwise the native Oracle Database Advanced Queuing interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.

Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be NULL.

**See Also:**

For additional information regarding subscriber options

- "WebSphere MQ System Properties" in Oracle Database Advanced Queuing User’s Guide
- "TIB/Rendezvous System Properties" in Oracle Database Advanced Queuing User’s Guide

OUTBOUND_PROPAGATION Subscribers
The parameters for a subscriber used for outbound propagation are interpreted as follows:

- **queue_name** specifies the local Oracle Database Advanced Queuing queue that is the propagation source. This must have a syntax of schema.queue.
- **destination** specifies the foreign queue to which messages are propagated. This must have a syntax of registered_queue@message_link.
- **rule** specifies an optional Oracle Database Advanced Queuing subscriber rule if the native Oracle Database Advanced Queuing interface is used, or a JMS selector if the Oracle JMS interface is used. If NULL, then no rule or selector is used.
- **transformation** specifies the transformation used to convert the Oracle Database Advanced Queuing payload to an ADT defined by Messaging Gateway.

Messaging Gateway propagation dequeues messages from the Oracle Database Advanced Queuing queue using the transformation to convert the Oracle Database Advanced Queuing payload to a known ADT defined by Messaging Gateway. The message is then enqueued in the foreign messaging system based on the Messaging Gateway ADT.

- **exception_queue** specifies the name of a local Oracle Database Advanced Queuing queue to which messages are moved if an exception occurs. This must have a syntax of schema.queue.

If the native Oracle Database Advanced Queuing interface is used, then a subscriber will be added to the Oracle Database Advanced Queuing queue when this procedure is called, whether or not Messaging Gateway is running. The local subscriber will be of the form sys.aq$_agent('MGW_subscriber_id', NULL, NULL).

If the Oracle JMS interface is used, then the Messaging Gateway agent will create a JMS durable subscriber with the name of MGW_subscriber_id. If the agent is not running when this procedure is called, then the durable subscriber will be created the next time the agent starts.

The exception queue has the following caveats:

- The user is responsible for creating the Oracle Database Advanced Queuing queue to be used as the exception queue.
- The payload type of the source and exception queue must match.
- The exception queue must be created as a queue type of DBMS_AQADM.NORMAL_QUEUE rather than DBMS_AQADM.EXCEPTION_QUEUE. Enqueue restrictions prevent Messaging Gateway propagation from using an Oracle Database Advanced Queuing queue of type EXCEPTION_QUEUE as a Messaging Gateway exception queue.

INBOUND_PROPAGATION Subscribers

The parameters for a subscriber used for inbound propagation are interpreted as follows:

- **queue_name** specifies the foreign queue that is the propagation source. This must have a syntax of registered_queue@message_link.
- **destination** specifies the local Oracle Database Advanced Queuing queue to which messages are propagated. This must have a syntax of schema.queue.
• **rule** specifies an optional subscriber rule that is valid for the foreign messaging system. This is **NULL** if no rule is needed.

• **transformation** specifies the transformation used to convert an ADT defined by Messaging Gateway to the Oracle Database Advanced Queuing payload type.

Messaging Gateway propagation dequeues messages from the foreign messaging system and converts the message body to a known ADT defined by Messaging Gateway. The transformation is used to convert the Messaging Gateway ADT to an Oracle Database Advanced Queuing payload type when the message is enqueued to the Oracle Database Advanced Queuing queue.

• **exception_queue** specifies the name of a foreign queue to which messages are moved if an exception occurs. This must have a syntax of `registered_queue@message_link`.

Whether or not a subscriber is needed depends on the requirements of the non-Oracle messaging system. If a durable subscriber is necessary, then it will be created by the Messaging Gateway agent. If the agent is not running at the time this procedure is called, then the creation of the subscriber on the non-Oracle messaging system will occur when the agent next starts.

The exception queue has the following caveats:

- The exception queue must be a registered non-Oracle queue.
- The source and exception queues must use the same messaging system link.

### 103.5.2 ALTER_AGENT Procedures

This procedure configures Messaging Gateway agent parameters.

**Syntax**

```sql
DBMS_MGWADM.ALTER_AGENT (  
    max_connections IN BINARY_INTEGER DEFAULT NULL,  
    max_memory IN BINARY_INTEGER DEFAULT NULL,  
    max_threads IN BINARY_INTEGER DEFAULT NULL,  
    service IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );

DBMS_MGWADM.ALTER_AGENT (  
    agent_name IN VARCHAR2,  
    username IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,  
    password IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,  
    database IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,  
    conntype IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,  
    max_memory IN PLS_INTEGER DEFAULT NULL,  
    max_threads IN PLS_INTEGER DEFAULT NULL,  
    service IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,  
    initfile IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,  
    comment IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```
### Parameters

**Table 103-21**  ALTER_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_connections</td>
<td>The maximum number of messaging connections to Oracle Database used by the Messaging Gateway agent. If it is NULL, then the current value is unchanged.</td>
</tr>
<tr>
<td></td>
<td><strong>Caution:</strong> This parameter has been deprecated.</td>
</tr>
<tr>
<td>max_memory</td>
<td>The maximum heap size, in MB, used by the Messaging Gateway agent. If it is NULL, then the current value is unchanged.</td>
</tr>
<tr>
<td>max_threads</td>
<td>The number of messaging threads that the Messaging Gateway agent creates. If it is NULL, then the current value is unchanged. The maximum value of max_threads is limited to 128.</td>
</tr>
<tr>
<td>service</td>
<td>Specifies the database service that the Oracle Scheduler job class used by this agent will have affinity to. In an Oracle RAC environment, this means that the Messaging Gateway agent will run on only those database instances that are assigned to the service. If NULL, the job class used by this agent will be altered to belong to the default service which is mapped to every instance. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.</td>
</tr>
<tr>
<td>agent_name</td>
<td>Identifies the Messaging Gateway agent. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.</td>
</tr>
<tr>
<td>username</td>
<td>Specifies the username used for connections to the Oracle Database. NULL is not allowed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged. If a username is specified then a password must also be specified.</td>
</tr>
<tr>
<td>password</td>
<td>Specifies the password used for connections to the Oracle Database. NULL is not allowed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged. A password must be specified if a username is specified.</td>
</tr>
<tr>
<td>database</td>
<td>Specifies the database connect string used for connections to the Oracle Database. NULL indicates that a local connection should be used. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged. Oracle strongly recommends that a connect string, rather than NULL, be specified. Usually it will be a net service name from tnsnames.ora.</td>
</tr>
<tr>
<td>conntype</td>
<td>Specifies the type of connection to the Oracle Database. DBMS_MGWADM.JDBC_OCI or DBMS_MGWADM.JDBC_THIN. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged</td>
</tr>
<tr>
<td>initfile</td>
<td>Specifies a Messaging Gateway initialization file used by this agent. NULL indicates that the default initialization file is used. If a value is specified, it should be the full path name of the file. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.</td>
</tr>
<tr>
<td>comment</td>
<td>Optional comments for this agent. NULL if a comment is not desired. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.</td>
</tr>
</tbody>
</table>
Note:
The max_connections parameter included in previous versions of this subprogram has been deprecated and is non-operational

Usage Notes

- Default values for these configuration parameters are set when the Messaging Gateway agent is installed.
- Changes to the max_memory and max_threads parameters take effect the next time the Messaging Gateway agent is active. If the Messaging Gateway agent is currently active, then it must be shut down and restarted for the changes to take effect.
- The service parameter is used to set an Oracle Scheduler job class attribute. The job class is used to create a Scheduler job that starts the Messaging Gateway agent. An Oracle administrator must create the database service. If the value is NULL, the job class will belong to an internal service that is mapped to all instances.
- The max_connections parameter is being deprecated as of the Oracle RDBMS 11g release. The number of messaging connections used by the Messaging Gateway Agent is based on the value of the max_threads parameter.
- The username, password, and database parameters specify connection information used by the Messaging Gateway agent for connections to the Oracle Database. An Oracle administrator should create the user and grant it the role MGW_AGENT_ROLE.

103.5.3 ALTER_JOB Procedure

This procedure alters the properties of a propagation job.

Syntax

```
DBMS_MGWADM.ALTER_JOB (
  job_name          IN   VARCHAR2,
  rule              IN   VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  transformation    IN   VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  exception_queue   IN   VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  poll_interval     IN   PLS_INTEGER DEFAULT 0,
  options           IN   SYS.MGW_PROPERTIES DEFAULT NULL,
  comments          IN   VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

Parameters

Table 103-22  ALTER_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Identifies the propagation job</td>
</tr>
<tr>
<td>rule</td>
<td>Specifies an optional subscription rule used to dequeue messages from the propagation source. The syntax and interpretation of this parameter depend on the propagation type. A NULL value indicates that no subscription rule is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.</td>
</tr>
</tbody>
</table>
Table 103-22  (Cont.) ALTER_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transformation</td>
<td>Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the value specified for propagation_type. A NULL value indicates that no transformation is needed. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.</td>
</tr>
<tr>
<td>exception_queue</td>
<td>Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. In cases in which no exception queue is associated with the job, propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the propagation type. A NULL value indicates that no exception queue is used. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.</td>
</tr>
<tr>
<td>poll_interval</td>
<td>Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available. Values: NULL, 0, or value &gt; 0: • If zero (default), the current value will not be changed. • If NULL, the current value will be reset and the Messaging Gateway default polling interval will be used. The default polling interval is 5 seconds and can be overridden by the Messaging Gateway initialization file.</td>
</tr>
<tr>
<td>options</td>
<td>Optional job properties. If NULL, no options will be changed. If not NULL, then the properties specified in this list are combined with the current optional properties to form a new set of job options.</td>
</tr>
<tr>
<td>comments</td>
<td>An optional comment for this agent, or NULL if one is not desired. If DBMS_MGWADM.NO_CHANGE, the current value will not be changed.</td>
</tr>
</tbody>
</table>

Usage Notes

• If the non-Oracle messaging link being accessed for the propagation job uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.

• The subscriber rule cannot be altered when propagating from a JMS source. Instead, the propagation job must be dropped and re-created with the new rule. For JMS, changing the message selector on a durable subscription is equivalent to deleting and re-creating the subscription.

• Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be DBMS_MGWADM.NO_CHANGE (the default value).

• The options parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner; add a new property, replace an existing property, remove an existing property or remove all properties.
103.5.4 ALTER_MSGSYSTEM_LINK Procedure for TIB/Rendezvous

This procedure alters the properties of a TIB/Rendezvous messaging system link.

Syntax

```sql
DBMS_MGWADM.ALTER_MSGSYSTEM_LINK (    linkname IN VARCHAR2,    properties IN SYS.MGW_TIBRV_PROPERTIES,    options IN SYS.MGW_PROPERTIES DEFAULT NULL,    comment IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

Parameters

**Table 103-23  ALTER_MSGSYSTEM_LINK Procedure Parameters for TIB/Rendezvous**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkname</td>
<td>The messaging system link name</td>
</tr>
<tr>
<td>properties</td>
<td>Basic properties for a TIB/Rendezvous messaging system link. If NULL, then no link properties will be changed.</td>
</tr>
<tr>
<td>options</td>
<td>Optional link properties. If NULL, then no options will be changed. If not NULL, then the properties specified in this list are combined with the current options properties to form a new set of link options.</td>
</tr>
<tr>
<td>comment</td>
<td>A user-specified description, or NULL if one is not desired. If DBMS_MGWADM.NO_CHANGE, then the current value will not be changed.</td>
</tr>
</tbody>
</table>

Usage Notes

To retain an existing value for a messaging link property with a VARCHAR2 datatype, specify DBMS_MGWADM.NO_CHANGE for that particular property. To preserve an existing value for a property of another datatype, specify NULL for that property.

The `options` parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.
Some properties cannot be modified, and this procedure will fail if an attempt is made to alter such a property. For properties and options that can be changed, a few are dynamic, and Messaging Gateway uses the new values immediately. Others require the Messaging Gateway agent to be shut down and restarted before they take effect.

103.5.5 ALTER_MSGSYSTEM_LINK Procedure for WebSphere MQ

This procedure alters the properties of a WebSphere MQ messaging system link.

**Syntax**

```
DBMS_MGWADM.ALTER_MSGSYSTEM_LINK (  
  linkname   IN  VARCHAR2,  
  properties IN  SYS.MGW_MQSERIES_PROPERTIES,  
  options    IN  SYS.MGW_PROPERTIES DEFAULT NULL,  
  comment    IN  VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE);
```

**Parameters**

**Table 103-24  ALTER_MSGSYSTEM_LINK Procedure Parameters for WebSphere MQ**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkname</td>
<td>The messaging system link name</td>
</tr>
<tr>
<td>properties</td>
<td>Basic properties for a WebSphere MQ messaging system link. If it is NULL, then no link properties are changed.</td>
</tr>
<tr>
<td>options</td>
<td>Optional link properties. NULL if no options are changed. If not NULL, then the properties specified in this list are combined with the current options properties to form a new set of link options.</td>
</tr>
<tr>
<td>comment</td>
<td>An optional description or NULL if not desired. If DBMS_MGWADM.NO_CHANGE is specified, then the current value is not changed.</td>
</tr>
</tbody>
</table>

**Usage Notes**

To retain an existing value for a messaging link property with a VARCHAR2 datatype, specify DBMS_MGWADM.NO_CHANGE for that particular property. To preserve an existing value for a property of another datatype, specify NULL for that property.
The `options` parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.

See Also:

SYS.MGW_PROPERTIES Object Type

Some properties cannot be modified, and this procedure will fail if an attempt is made to alter such a property. For properties and options that can be changed, a few are dynamic, and Messaging Gateway uses the new values immediately. Others require the Messaging Gateway agent to be shut down and restarted before they take effect.

See Also:

"WebSphere MQ System Properties" in Oracle Database Advanced Queuing User's Guide for more information about the messaging system properties and options

103.5.6 ALTER_PROPAGATION_SCHEDULE Procedure

This procedure alters a propagation schedule.

Note:

This subprogram has been deprecated as a result of improved technology (see ALTER_JOB Procedure), and is retained only for reasons of backward compatibility.

Syntax

```sql
DBMS_MGWADM.ALTER_PROPAGATION_SCHEDULE (  
schedule_id  IN VARCHAR2,  
duration     IN NUMBER DEFAULT NULL,  
next_time    IN VARCHAR2 DEFAULT NULL,  
latency      IN NUMBER DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_id</td>
<td>Identifies the propagation schedule to be altered</td>
</tr>
<tr>
<td>duration</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>next_time</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>
**Table 103-25**  (Cont.) ALTER_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>latency</td>
<td>Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available in the source queue, then the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available. Values: NULL or value &gt; 0. If latency is NULL, then the Messaging Gateway agent default polling interval will be used. The default polling interval is 5 seconds, but it can be overridden by the Messaging Gateway initialization file.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure always overwrites the existing value for each parameter. If a given parameter is not specified, then the existing values are overwritten with the default value.

### 103.5.7 ALTER_SUBSCRIBER Procedure

This procedure alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination.

**Note:**

This subprogram has been deprecated as a result of improved technology (see ALTER_JOB Procedure), and is retained only for reasons of backward compatibility.

**Syntax**

```sql
DBMS_MGWADM.ALTER_SUBSCRIBER(
    subscriber_id IN VARCHAR2,
    rule             IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    transformation   IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    exception_queue  IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    options          IN SYS.MGW_PROPERTIES DEFAULT NULL);
```

**Parameters**

**Table 103-26**  ALTER_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriber_id</td>
<td>Identifies the subscriber to be altered</td>
</tr>
</tbody>
</table>
Table 103-26  (Cont.) ALTER_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule</td>
<td>Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. The syntax and interpretation of this parameter depend on the subscriber propagation type.</td>
</tr>
<tr>
<td></td>
<td>A NULL value indicates that no subscription rule is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies the transformation needed to convert between the Oracle Database Advanced Queuing payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the subscriber propagation type.</td>
</tr>
<tr>
<td></td>
<td>A NULL value indicates that no transformation is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.</td>
</tr>
<tr>
<td>exception_queue</td>
<td>Specifies a queue used for exception message logging. This queue must be on the same messaging system as the propagation source. If no exception queue is associated with the subscriber, then propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the subscriber propagation type.</td>
</tr>
<tr>
<td></td>
<td>A NULL value indicates that no exception queue is used. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.</td>
</tr>
<tr>
<td></td>
<td>The source queue and exception queue cannot be the same queue.</td>
</tr>
<tr>
<td>options</td>
<td>Optional subscriber properties. If NULL, then no options will be changed. If not NULL, then the properties specified in this list are combined with the current optional properties to form a new list of subscriber options.</td>
</tr>
</tbody>
</table>

Usage Notes

If the non-Oracle messaging link being accessed for the subscriber uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Database Advanced Queuing queues. Otherwise the native Oracle Database Advanced Queuing interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.

When propagating from a JMS source, the subscriber rule cannot be altered. Instead, the subscriber must be removed and added with the new rule. For JMS, changing the message selector on a durable subscription is equivalent to deleting and re-creating the subscription.

Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be DBMS_MGWADM.NO_CHANGE (the default value).

The options parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.
103.5.8 CLEANUP_GATEWAY Procedures

This procedure cleans up Messaging Gateway. The procedure performs cleanup or recovery actions that may be needed when Messaging Gateway is left in some abnormal or unexpected condition. The MGW_GATEWAY view lists Messaging Gateway status and configuration information that pertains to the cleanup actions.

Syntax

```sql
DBMS_MGWADM.CLEANUP_GATEWAY(
   action       IN   BINARY_INTEGER,
   sarg         IN   VARCHAR2 DEFAULT NULL);

DBMS_MGWADM.CLEANUP_GATEWAY(
   agent_name   IN   VARCHAR2,   action       IN   BINARY_INTEGER,
   sarg         IN   VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 103-27  CLEANUP_GATEWAY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>The cleanup action to be performed. Values:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_MGWADM.CLEAN_STARTUP_STATE for Messaging Gateway start up state recovery</td>
</tr>
<tr>
<td></td>
<td>- DBMS_MGWADM.CLEAN_LOG_QUEUES for log queue cleanup</td>
</tr>
<tr>
<td></td>
<td>- DBMS_MGWADM.RESET_SUB_MISSING_LOG_REC for propagation job recovery due to missing log record</td>
</tr>
<tr>
<td></td>
<td>- DBMS_MGWADM.RESET_SUB_MISSING_MESSAGE for propagation job recovery due to missing message</td>
</tr>
<tr>
<td>sarg</td>
<td>Optional argument whose meaning depends on the value specified for action. This should be NULL if it is not used for the specified action.</td>
</tr>
</tbody>
</table>
Table 103-27  (Cont.) CLEANUP_GATEWAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Identifies the Messaging Gateway agent.</td>
</tr>
<tr>
<td></td>
<td>DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.</td>
</tr>
</tbody>
</table>

**Usage Notes**

**CLEAN_STARTUP_STATE**

*sarg* is not used and must be **NULL**.

The **CLEAN_STARTUP_STATE** action recovers Messaging Gateway to a known state when the Messaging Gateway agent has crashed or some other abnormal event occurs, and Messaging Gateway cannot be restarted. This should be done only when the Messaging Gateway agent has been started but appears to have crashed or has been nonresponsive for an extended period of time.

The **CLEAN_STARTUP_STATE** action may be needed when the **MGW_GATEWAY** view shows that the **AGENT_STATUS** value is something other than **NOT_STARTED** or **START_SCHEDULED**, and the **AGENT_PING** value is **UNREACHABLE** for an extended period of time.

If the **AGENT_STATUS** value is **BROKEN**, then the Messaging Gateway agent cannot be started until the problem has been resolved and the **CLEAN_STARTUP_STATE** action used to reset the agent status. A **BROKEN** status can indicate that the Messaging Gateway start job detected a Messaging Gateway agent already running. This condition that should never occur under normal use.

Cleanup tasks include:

- Removing the Scheduler job used to start the external Messaging Gateway agent process.
- Setting certain configuration information to a known state. For example, setting the agent status to **NOT_STARTED**.

Execution of this command fails if:

- The agent status is **NOT_STARTED** or **START_SCHEDULED**.
- No shutdown attempt has been made prior to calling this procedure, except if the agent status is **STARTING**.
- The Messaging Gateway agent is successfully contacted.

The assumption is that the agent is active, and this procedure fails. If the agent does not respond after several attempts have been made, then the cleanup tasks are performed. This procedure takes at least several seconds and possibly up to one minute. This is expected behavior under conditions where this particular cleanup action is appropriate and necessary.
Note:

Terminate any Messaging Gateway agent process that may still be running after a CLEAN_STARTUP_STATE action has been successfully performed. This should be done before calling DBMS_MGWADM.STARTUP to start Messaging Gateway. The process is usually named extprocmgwextproc.

CLEAN_LOG_QUEUES

sarg is not used and must be NULL.

The Messaging Gateway agent will clean log queues for all configured messaging system links. The agent will temporarily stop all propagation activity and then remove all obsolete and bad log records from the log queues for all links. The procedure will fail if the Messaging Gateway agent is not running.

This cleanup action is automatically performed each time the Messaging Gateway agent is started.

Note:

The CLEAN_LOG_QUEUES action is performed only on agent startup. If this procedure is called when the agent is running, then the Messaging Gateway agent ignores it.

RESET_SUB_MISSING_LOG_REC

sarg specifies a Messaging Gateway job name (or subscriber ID) to be reset. It must not be NULL.

The Messaging Gateway agent recovers a Messaging Gateway propagation job that has failed due to a missing log record. The agent will reset the source and destination log records. The procedure will fail if the Messaging Gateway agent is not running.

Note:

If the messages in the source queue had already been propagated to the destination queue, then this action may result in duplicate messages.

RESET_SUB_MISSIING_MESSAGE

sarg specifies a Messaging Gateway job name (or subscriber ID) to be reset. It must not be NULL.

The Messaging Gateway agent recovers a Messaging Gateway propagation job that has failed due to a missing persistent source message. The agent will treat the message as a non-persistent message and continue processing that propagation job. The procedure will fail if the Messaging Gateway agent is not running.
103.5.9 CREATE_AGENT Procedure

This procedure creates a Messaging Gateway agent that will be used to process propagation jobs.

Syntax

```sql
DBMS_MGWADM.CREATE_AGENT(
    agent_name     IN   VARCHAR2,
    username       IN   VARCHAR2 DEFAULT NULL,
    password       IN   VARCHAR2 DEFAULT NULL,
    database       IN   VARCHAR2 DEFAULT NULL,
    conntype       IN   VARCHAR2 DEFAULT DBMS_MGWADM.JDBC_OCI,
    max_memory     IN   PLS_INTEGER DEFAULT 64,
    max_threads    IN   PLS_INTEGER DEFAULT 1,
    service        IN   VARCHAR2 DEFAULT NULL,
    initfile       IN   VARCHAR2 DEFAULT NULL,
    comment        IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103-28  CREATE_AGENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>A name used to identify the agent</td>
</tr>
<tr>
<td>username</td>
<td>Specifies the username used for connections to the Oracle Database</td>
</tr>
<tr>
<td>password</td>
<td>Specifies the password used for connections to the Oracle Database. A password must be specified if a username is specified.</td>
</tr>
<tr>
<td>database</td>
<td>Specifies the database connect string used for connections to the Oracle Database. NULL indicates that a local connection should be used. A value can be specified only if a username is specified. Oracle strongly recommends that a connect string, rather than NULL be specified. Usually it will be a net service name from <code>tnsnames.ora</code>.</td>
</tr>
<tr>
<td>conntype</td>
<td>Specifies the type of connection to the Oracle Database. Values: <code>DBMS_MGWADM.JDBC_OCI, DBMS_MGWADM.JDBC_THIN</code></td>
</tr>
<tr>
<td>max_memory</td>
<td>Specifies the maximum heap size, in MB, used by the Messaging Gateway agent</td>
</tr>
<tr>
<td>max_threads</td>
<td>Specifies the number of messaging threads that the Messaging Gateway agent creates. This determines the number of propagation jobs that the agent can concurrently process. The maximum value of <code>max_threads</code> is limited to 128.</td>
</tr>
<tr>
<td>service</td>
<td>Specifies the database service that the Oracle Scheduler job class used by this agent will have affinity to. In an Oracle RAC environment, this means that the Messaging Gateway agent will only run on those database instances that are assigned to the service. If NULL, then the job class will belong to the default service which is mapped to every instance.</td>
</tr>
<tr>
<td>initfile</td>
<td>Specifies a Messaging Gateway initialization file used by this agent. NULL indicates that the default initialization file is used. If a value is specified, it should be the full path name of the file.</td>
</tr>
<tr>
<td>comment</td>
<td>An optional comment for this agent. NULL if one is not desired.</td>
</tr>
</tbody>
</table>
Usage Notes

- The Messaging Gateway automatically configures a default agent when Messaging Gateway is installed. The name of the default agent is DEFAULT_AGENT. This procedure can be used to create additional agents.

- The username, password, and database parameters specify connection information used by the Messaging Gateway agent for connections to the Oracle Database. An Oracle administrator should create the database user and grant it the role MGW_AGENT_ROLE. It is not mandatory that the connection information be specified when this procedure is called but it must be set before the agent can be started.

- The service parameter is used to create an Oracle Scheduler job class. The job class is used to create a Scheduler job that starts the Messaging Gateway agent. An Oracle administrator must create the database service. If the value is NULL, the job class will belong to an internal service that is mapped to all instances.

103.5.10 CREATE_JOB Procedure

This procedure creates a job used to propagate message from a source to a destination.

Syntax

```sql
DBMS_MGWADM.CREATE_JOB ( job_name           IN   VARCHAR2,
propagation_type   IN   PLS_INTEGER,
source             IN   VARCHAR2,
destination        IN   VARCHAR2,
rule               IN   VARCHAR2 DEFAULT NULL,
transformation     IN   VARCHAR2 DEFAULT NULL,
exception_queue    IN   VARCHAR2 DEFAULT NULL,
poll_interval      IN   PLS_INTEGER DEFAULT NULL,
options            IN   SYS.MGW_PROPERTIES DEFAULT NULL,
enabled            IN   BOOLEAN DEFAULT TRUE,
comments           IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 103-29  CREATE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>A user defined name to identify the propagation job</td>
</tr>
<tr>
<td>propagation_type</td>
<td>Specifies the type of message propagation.</td>
</tr>
<tr>
<td>source</td>
<td>Specifies the source queue whose messages are to be propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
<tr>
<td>destination</td>
<td>Specifies the destination queue to which messages are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
</tbody>
</table>
Table 103-29 (Cont.) CREATE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule</td>
<td>Specifies an optional subscription rule used to dequeue messages from the source queue. This should be NULL if no rule is needed. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the value specified for propagation_type. If no transformation is specified the Oracle Streams AQ payload type must be supported by Messaging Gateway.</td>
</tr>
<tr>
<td>exception_queue</td>
<td>Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. If NULL, an exception queue will not be used and propagation will stop if a problem occurs. The syntax and interpretation of this parameter depend on the value specified for propagation_type. The source queue and exception queue cannot be the same queue.</td>
</tr>
<tr>
<td>poll_interval</td>
<td>Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available. Values: NULL or value &gt; 0. If NULL, then the Messaging Gateway default polling interval will be used. The default polling interval is 5 seconds and can be overridden by the Messaging Gateway initialization file.</td>
</tr>
<tr>
<td>options</td>
<td>Optional job properties, NULL if there are none. Typically these are lesser used configuration properties supported by the messaging system.</td>
</tr>
<tr>
<td>enabled</td>
<td>Specifies whether this propagation job is enabled after creation. Values: TRUE, FALSE.</td>
</tr>
<tr>
<td>comments</td>
<td>An optional comment for this job. NULL if one is not desired.</td>
</tr>
</tbody>
</table>

Usage Notes

- The job must be enabled and Messaging Gateway agent started in order for messages to be propagated.
- If the non-Oracle messaging link being accessed for the propagation job uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.
Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be NULL.

OUTBOUND_PROPAGATION Jobs

The parameters for an outbound propagation job are interpreted as follows:

- **source** specifies the local Oracle Streams AQ queue that is the propagation source. This must have syntax of schema.queue. This can be either a multiple consumer queue or a single consumer queue.

- **destination** specifies the non-Oracle queue to which messages are propagated. This must have syntax of registered_queue@message_link.

- **rule** specifies an optional Oracle Streams AQ subscriber rule if the native Oracle Stream AQ interface is used, or a JMS selector if the Oracle JMS interface is used. If NULL, then no rule or selector is used. This parameter must be NULL if the native Oracle Stream AQ interface is used and the propagation source is a single consumer queue.

- **transformation** specifies the transformation used to convert the Oracle Streams AQ payload to an ADT defined by Messaging Gateway. The full transformation name (schema.name) should be used if one is specified.

  Messaging Gateway propagation dequeues messages from the Oracle Streams AQ queue using the transformation to convert the Oracle Streams AQ payload to a known ADT defined by Messaging Gateway. The message is then enqueued in the non-Oracle messaging system based on the Messaging Gateway ADT.

- **exception_queue** specifies the name of a local Oracle Streams AQ queue to which messages are moved if an exception occurs. The syntax must be schema.queue.

  If the native Oracle Streams AQ interface is used and the source is a multiple consumer queue, then a subscriber will be added to the Oracle Streams AQ queue when this procedure is called, whether or not the Messaging Gateway agent is running. The local subscriber will be of the form sys.aq$_agent('MGW_job_name', NULL, NULL).

  If the Oracle JMS interface is used, then the Messaging Gateway agent will create a JMS durable subscriber with the name of MGW_job_name. If the agent is not running when this procedure is called, then the durable subscriber will be created the next time the agent starts.

  The exception queue has the following conditions:

  - The user is responsible for creating the Oracle Streams AQ queue to be used as the exception queue.
  - The payload type of the source queue and exception queue must match.
  - The exception queue must be created as a queue type of DBMS_AQADM.NORMAL_QUEUE. Enqueue restrictions prevent Messaging Gateway from using an Oracle Streams AQ queue of type DBMS_AQADM.EXCEPTION_QUEUE as a Messaging Gateway exception queue.

INBOUND_PROPAGATION Jobs

The parameters for an inbound propagation job are interpreted as follows:

- **source** specifies the non-Oracle queue that is the propagation source. The syntax must be registered_queue@message_link.
destination specifies the local Oracle Streams AQ queue to which messages are propagated. The syntax must be schema.queue.

• rule specifies an optional subscriber rule that is valid for the non-Oracle messaging system. This should be NULL if no rule is needed.

• transformation specifies the transformation used to convert an ADT defined by Messaging Gateway to the Oracle Streams AQ payload type. The full transformation name (schema.name) should be used if one is specified

Messaging Gateway propagation dequeues messages from the non-Oracle messaging system and converts the message body to a known ADT defined by Messaging Gateway. The transformation is used to convert the Messaging Gateway ADT to an Oracle Streams AQ payload type when the message is enqueued to the Oracle Streams AQ queue.

• exception_queue specifies the name of a registered non-Oracle queue to which messages are moved if an exception occurs. The syntax must be registered_queue@message_link.

Whether or not a subscriber is needed for the source queue depends on the requirements of the non-Oracle messaging system. If a durable subscriber is necessary, then the Messaging Gateway agent will create it. If the agent is not running when this procedure is called, then the subscriber will be created on the non-Oracle messaging system the next time the agent starts.

The exception queue has the following conditions:

- The exception queue must be a registered non-Oracle queue.
- The source queue and exception queue must use the same messaging system link.

103.5.11 CREATE_MSGSYSTEM_LINK Procedures for TIB/Rendezvous

This procedure creates a link to a TIB/Rendezvous messaging system.

Syntax

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK (
    linkname     IN  VARCHAR2,
    properties   IN  SYS.MGW_TIBRV_PROPERTIES,
    options      IN  SYS.MGW_PROPERTIES DEFAULT NULL,
    comment      IN  VARCHAR2 DEFAULT NULL );
```

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK (
    linkname     IN  VARCHAR2,
    agent_name   IN  VARCHAR2,
    properties   IN  SYS.MGW_TIBRV_PROPERTIES,
    options      IN  SYS.MGW_PROPERTIES DEFAULT NULL,
    comment      IN  VARCHAR2 DEFAULT NULL );
```

Parameters

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkname</td>
<td>A user-defined name to identify this messaging system link</td>
</tr>
<tr>
<td>properties</td>
<td>Basic properties of a TIB/Rendezvous messaging system link.</td>
</tr>
</tbody>
</table>
```
Table 103-30 (Cont.) CREATE_MSGSYSTEM_LINK Procedure Parameters for TIB/Rendezvous

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Optional link properties. NULL if there are none. These are less frequently used configuration properties supported by the messaging system.</td>
</tr>
<tr>
<td>comment</td>
<td>A user-specified description. NULL if one is not desired.</td>
</tr>
<tr>
<td>agent_name</td>
<td>Specifies the Messaging Gateway agent that will be used to process all propagation jobs associated with this link. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.</td>
</tr>
</tbody>
</table>

Usage Notes

The Messaging Gateway default agent will process the propagation jobs associated with this link if an agent name is not specified.

See Also:

"TIB/Rendezvous System Properties" in Oracle Database Advanced Queuing User's Guide for more information about the messaging system properties and options.

103.5.12 CREATE_MSGSYSTEM_LINK Procedures for WebSphere MQ

This procedure creates a messaging system link to a WebSphere MQ messaging system.

Syntax

```sql
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
    linkname    IN VARCHAR2,
    properties  IN SYS.MGW_MQSERIES_PROPERTIES,
    options     IN SYS.MGW_PROPERTIES DEFAULT NULL,
    comment     IN VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
    linkname    IN VARCHAR2,
    agent_name  IN VARCHAR2,
    properties  IN SYS.MGW_MQSERIES_PROPERTIES,
    options     IN SYS.MGW_PROPERTIES DEFAULT NULL,
    comment     IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 103-31  CREATE_MSGSYSTEM_LINK Procedure Parameters for WebSphere MQ

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkname</td>
<td>A user-defined name to identify the messaging system link</td>
</tr>
<tr>
<td>properties</td>
<td>Basic properties of a WebSphere MQ messaging system link</td>
</tr>
<tr>
<td>options</td>
<td>Optional link properties. NULL if there are none. These are less frequently used configuration properties supported by the messaging system.</td>
</tr>
<tr>
<td>comment</td>
<td>A user-specified description. NULL if one is not desired</td>
</tr>
<tr>
<td>agent_name</td>
<td>Specifies the Messaging Gateway agent that will be used to process all propagation jobs associated with this link. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.</td>
</tr>
</tbody>
</table>

Usage Notes

The Messaging Gateway default agent will process the propagation jobs associated with this link if an agent name is not specified.

See Also:

"WebSphere MQ System Properties“ in Oracle Database Advanced Queuing User's Guide for more information about the messaging system properties and options

103.5.13 DB_CONNECT_INFO Procedure

This deprecated procedure configures connection information used by the Messaging Gateway default agent for connections to Oracle Database.

Note:

This subprogram has been deprecated as a result of improved technology (see ALTER_AGENT Procedures), and is retained only for reasons of backward compatibility.

Syntax

DBMS_MGWADM.DB_CONNECT_INFO (  
  username IN VARCHAR2,  
  password IN VARCHAR2,  
  database IN VARCHAR2 DEFAULT NULL);
Parameters

Table 103-32  DB_CONNECT_INFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>The username used for connections to Oracle Database. NULL is not allowed</td>
</tr>
<tr>
<td>password</td>
<td>The password used for connections to Oracle Database. NULL is not allowed</td>
</tr>
<tr>
<td>database</td>
<td>The database connect string used by the Messaging Gateway agent. NULL indicates that a local connection should be used. Oracle strongly recommends that a not NULL value be specified. Usually it will be a net service name from tnsnames.ora.</td>
</tr>
</tbody>
</table>

Usage Notes

The Messaging Gateway agent connects to Oracle Database as the user configured by this procedure. An Oracle administrator should create the user, grant it the role MGW_AGENT_ROLE, and then call this procedure to configure Messaging Gateway. Role MGW_AGENT_ROLE is used to grant this user special privileges needed to access Messaging Gateway configuration information stored in the database, enqueue or dequeue messages to and from Oracle Database Advanced Queuing queues, and perform certain Oracle Database Advanced Queuing administration tasks.

103.5.14 DISABLE_JOB Procedure

This procedure disables a propagation job.

Syntax

```sql
DBMS_MGWADM.DISABLE_JOB (job_name IN VARCHAR2);
```

Parameters

Table 103-33  DISABLE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Identifies the propagation job</td>
</tr>
</tbody>
</table>
103.5.15 DISABLE_PROPAGATION_SCHEDULE Procedure

This deprecated procedure disables a propagation schedule.

**Note:**

This subprogram has been deprecated as a result of improved technology (see DISABLE_JOB Procedure), and is retained only for reasons of backward compatibility.

**Syntax**

```sql
DBMS_MGWADM.DISABLE_PROPAGATION_SCHEDULE (schedule_id IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_id</td>
<td>Identifies the propagation schedule to be disabled</td>
</tr>
</tbody>
</table>

103.5.16 ENABLE_JOB Procedure

This procedure enables a propagation job.

**Syntax**

```sql
DBMS_MGWADM.ENABLE_JOB (job_name IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Identifies the propagation job</td>
</tr>
</tbody>
</table>
103.5.17 ENABLE_PROPAGATION_SCHEDULE Procedure

This deprecated procedure enables a propagation schedule.

**Note:**
This subprogram has been deprecated as a result of improved technology (see ENABLE_JOB Procedure), and is retained only for reasons of backward compatibility.

Syntax

```sql
DBMS_MGWADM.ENABLE_PROPAGATION_SCHEDULE ( 
    schedule_id  IN VARCHAR2 );
```

Parameters

**Table 103-36**  ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_id</td>
<td>Identifies the propagation schedule to be enabled</td>
</tr>
</tbody>
</table>

103.5.18 REGISTER_FOREIGN_QUEUE Procedure

This procedure registers a non-Oracle queue entity in Messaging Gateway.

Syntax

```sql
DBMS_MGWADM.REGISTER_FOREIGN_QUEUE( 
    name            IN VARCHAR2, 
    linkname        IN VARCHAR2, 
    provider_queue  IN VARCHAR2 DEFAULT NULL, 
    domain          IN INTEGER DEFAULT NULL, 
    options         IN SYS.MGW_PROPERTIES DEFAULT NULL, 
    comment         IN VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 103-37**  REGISTER_FOREIGN_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The registered queue name. This name identifies the foreign queue within Messaging Gateway and need not match the name of the queue in the foreign messaging system.</td>
</tr>
<tr>
<td>linkname</td>
<td>The link name for the messaging system on which this queue exists</td>
</tr>
<tr>
<td>provider_queue</td>
<td>The message provider (native) queue name. If NULL, then the value provided for the name parameter is used as the provider queue name.</td>
</tr>
</tbody>
</table>
### Table 103-37  (Cont.) REGISTER_FOREIGN_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>The domain type of the queue. NULL means the domain type is automatically determined based on the messaging system of the queue. DBMS_MGWADM.DOMAIN_QUEUE is for a queue (point-to-point model). DBMS_MGWADM.DOMAIN_TOPIC is for a topic (publish-subscribe model).</td>
</tr>
<tr>
<td>options</td>
<td>Optional queue properties</td>
</tr>
<tr>
<td>comment</td>
<td>A user-specified description. Can be NULL.</td>
</tr>
</tbody>
</table>

#### Usage Notes

This procedure does not create the physical queue in the non-Oracle messaging system. The non-Oracle queue must be created using the administration tools for that messaging system.

#### See Also:

For more information when registering queues for the WebSphere MQ messaging system or the TIB/Rendezvous messaging system, specifically "Optional Foreign Queue Configuration Properties" in *Oracle Database Advanced Queuing User's Guide*.

### 103.5.19 REMOVE_AGENT Procedure

This procedure removes a Messaging Gateway agent.

#### Syntax

```
DBMS_MGWADM.REMOVE_AGENT(
    agent_name   IN   VARCHAR2 );
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Identifies the Messaging Gateway agent</td>
</tr>
</tbody>
</table>

#### Usage Notes

All messaging system links associated with this Messaging Gateway agent must be removed and the agent must be stopped before it can be removed. The Messaging Gateway default agent cannot be removed.
103.5.20 REMOVE_JOB Procedure

This procedure removes a propagation job.

Syntax

DBMS_MGWADM.REMOVE_JOB(
    job_name IN VARCHAR2, force IN PLS_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE);

Parameters

Table 103-39 REMOVE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Identifies the propagation job</td>
</tr>
<tr>
<td>force</td>
<td>Specifies whether the procedure should succeed even if Messaging Gateway is not able to perform all cleanup actions pertaining to this propagation job. Values: DBMS_MGWADM.NO_FORCE, DBMS_MGWADM.FORCE</td>
</tr>
<tr>
<td></td>
<td>NO_FORCE (default) means the job is not removed if Messaging Gateway is unable to clean up successfully</td>
</tr>
<tr>
<td></td>
<td>FORCE means the job is removed even though all cleanup actions may not be done</td>
</tr>
</tbody>
</table>

Usage Notes

- The Messaging Gateway agent uses various resources of the Oracle Database and the non-Oracle messaging system for its propagation work. These resources need to be released when the job is removed. For example, Messaging Gateway may create a durable subscriber on the source queue that should be removed when the job is removed. Therefore, this procedure should normally be called when the Messaging Gateway agent is running and able to access the non-Oracle messaging system associated with this job.

- For outbound propagation, a local subscriber is removed from the Oracle Streams AQ queue when the propagation source is a multiple consumer queue.

103.5.21 REMOVE_MSGSYSTEM_LINK Procedure

This procedure removes a messaging system link for a non-Oracle messaging system.

Syntax

DBMS_MGWADM.REMOVE_MSGSYSTEM_LINK(
    linkname IN VARCHAR2);
**Parameters**

**Table 103-40  REMOVE_MSGSYSTEM_LINK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkname</td>
<td>The messaging system link name</td>
</tr>
</tbody>
</table>

**Usage Notes**

All registered queues associated with this link must be removed before the messaging system link can be removed. This procedure fails if there is a registered foreign (non-Oracle) queue that references this link.

103.5.22 REMOVE_OPTION Procedure

This procedure removes a Messaging Gateway configuration option. It can be used to remove an agent option, a messaging link option, or a propagation job option.

**Syntax**

```sql
DBMS_MGWADM.REMOVE_OPTION (  
    target_type   IN   PLS_INTEGER,  
    target_name   IN   VARCHAR2,  
    option_name   IN   VARCHAR2);
```

**Parameters**

**Table 103-41  REMOVE_OPTION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_type</td>
<td>Specifies the target type of the Messaging Gateway entity:</td>
</tr>
<tr>
<td>target_name</td>
<td>Name or identifier of the target. The value for this parameter depends</td>
</tr>
<tr>
<td>option_name</td>
<td>Option name. This must not be NULL.</td>
</tr>
</tbody>
</table>

**Usage Notes**

DBMS_MGWADM.AGENT_JAVA_PROP Target
The procedure removes an agent option used to set a Java System property when the Messaging Gateway agent is started. The agent must be restarted for the change to take effect.

The parameters are interpreted as follows:

- **target_name** specifies the name of the Messaging Gateway agent.
  - DBMS_MGWADM.DEFAULT_AGENT can be used for the default agent.
- **option_name** specifies the Java System property
- **encrypted** can be either TRUE or FALSE

**DBMS_MGWADM.MSGLINK_OPTION** Target

The procedure removes a single option for a Messaging Gateway messaging system link. This is equivalent to calling `DBMS_MGWADM.ALTER_MSGSYSTEM_LINK` and using the options parameter to remove an option.

The parameters are interpreted as follows:

- **target_name** specifies the name of the message system link
- **option_name** specifies the option to set
- **encrypted** must be FALSE

**DBMS_MGWADM.JOB_OPTION** Target

The procedure removes a single option for a Messaging Gateway propagation job. This is equivalent to calling `DBMS_MGWADM.ALTER_JOB` and using the options parameter to remove an option.

The parameters are interpreted as follows:

- **target_name** specifies the name of the propagation job
- **option_name** specifies the option to set
- **encrypted** must be FALSE

**103.5.23 REMOVE_SUBSCRIBER Procedure**

This procedure removes a subscriber used to consume messages from a source queue for propagation to a destination.

**Note:**

This subprogram has been deprecated as a result of improved technology (see **REMOVE_JOB Procedure**), and is retained only for reasons of backward compatibility.

**Syntax**

```sql
DBMS_MGWADM.REMOVE_SUBSCRIBER (  
    subscriber_id  IN VARCHAR2,  
    force          IN BINARY_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE );
```
Parameters

Table 103-42  REMOVE_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriber_id</td>
<td>Identifies the subscriber to be removed</td>
</tr>
<tr>
<td>force</td>
<td>Specifies whether this procedure should succeed even if Messaging Gateway is not able to perform all cleanup actions pertaining to this subscriber. Values: DBMS_MGWADM.NO_FORCE, DBMS_MGWADM.FORCE</td>
</tr>
<tr>
<td></td>
<td>• NO_FORCE means the subscriber is not removed if Messaging Gateway is unable to clean up successfully (default)</td>
</tr>
<tr>
<td></td>
<td>• FORCE means the subscriber is removed even though all cleanup actions may not be done</td>
</tr>
</tbody>
</table>

Usage Notes

• The Messaging Gateway agent uses various resources of Oracle Database and the non-Oracle messaging system for its propagation work. These resources are typically associated with each subscriber and need to be released when the subscriber is no longer needed. Therefore, this procedure should only be called when the Messaging Gateway agent is running and able to access the non-Oracle messaging system associated with this subscriber.

• For outbound propagation, a local subscriber is removed from the Oracle Database Advanced Queuing queue.

103.5.24 RESET_JOB Procedure

This procedure resets the propagation error state for a propagation job.

Syntax

```
DBMS_MGWADM.RESET_JOB(
    job_name IN VARCHAR2);
```

Parameters

Table 103-43  RESET_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Identifies the propagation job</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure can be used to reset a propagation job that has been set to a failed state and propagation activities have been stopped. The administrator should correct the problem and then call this procedure to allow the agent to retry the propagation job. The STATUS field of the MGW_JOBS view indicates the job status.
103.5.25 RESET_SUBSCRIBER Procedure

This procedure resets the propagation error state for a subscriber.

**Note:**
This subprogram has been deprecated as a result of improved technology (see RESET_JOB Procedure), and is retained only for reasons of backward compatibility.

**Syntax**

```sql
DBMS_MGWADM.RESET_SUBSCRIBER ( 
    subscriber_id  IN VARCHAR2 );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriber_id</td>
<td>Identifies the subscriber</td>
</tr>
</tbody>
</table>

103.5.26 SCHEDULE_PROPAGATION Procedure

This procedure schedules message propagation from a source to a destination.

The schedule must be enabled and Messaging Gateway started in order for messages to be propagated.

**Note:**
This subprogram has been deprecated as a result of improved technology (see CREATE_JOB Procedure), and is retained only for reasons of backward compatibility.

**Syntax**

```sql
DBMS_MGWADM.SCHEDULE_PROPAGATION ( 
    schedule_id       IN VARCHAR2, 
    propagation_type  IN BINARY_INTEGER, 
    source            IN VARCHAR2, 
    destination       IN VARCHAR2, 
    start_time        IN DATE DEFAULT SYSDATE, 
    duration          IN NUMBER DEFAULT NULL, 
    next_time         IN VARCHAR2 DEFAULT NULL, 
    latency           IN NUMBER DEFAULT NULL); 
```
Parameters

Table 103-45  SCHEDULE_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_id</td>
<td>Specifies a user-defined name that identifies the schedule</td>
</tr>
<tr>
<td>propagation_type</td>
<td>Specifies the type of message propagation. DBMS_MGWADM.OUTBOUND_PROPAGATION is for Oracle Database Advanced Queuing to non-Oracle propagation. DBMS_MGWADM.INBOUND_PROPAGATION is for non-Oracle to Oracle Database Advanced Queuing propagation.</td>
</tr>
<tr>
<td>source</td>
<td>Specifies the source queue whose messages are to be propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
<tr>
<td>destination</td>
<td>Specifies the destination queue to which messages are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.</td>
</tr>
<tr>
<td>start_time</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>duration</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>next_time</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>latency</td>
<td>Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available in the source queue, then the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available. Values: NULL or value &gt; 0. If latency is NULL, then the Messaging Gateway agent default polling interval will be used. The default polling interval is 5 seconds but it can be overridden by the Messaging Gateway initialization file.</td>
</tr>
</tbody>
</table>

Usage Notes

For outbound propagation, parameters are interpreted as follows:

- source specifies the local Oracle Database Advanced Queuing queue from which messages are propagated. This must have a syntax of schema.queue.
- destination specifies the foreign queue to which messages are propagated. This must have a syntax of registered_queue@message_link.

For inbound propagation, parameters are interpreted as follows:

- source specifies the foreign queue from which messages are propagated. This must have a syntax of registered_queue@message_link.
- destination specifies the local Oracle Database Advanced Queuing queue to which messages are propagated. This must have a syntax of schema.queue.

The schedule is set to an enabled state when it is created.
103.5.27 SET_LOG_LEVEL Procedures

This procedure dynamically alters the Messaging Gateway agent logging level. The Messaging Gateway agent must be running.

Syntax

```
DBMS_MGWADM.SET_LOG_LEVEL (log_level IN BINARY_INTEGER);
```

```
DBMS_MGWADM.SET_LOG_LEVEL (agent_name IN VARCHAR2,
                              log_level IN BINARY_INTEGER);
```

Parameters

Table 103-46 SET_LOG_LEVEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_level</td>
<td>Level at which the Messaging Gateway agent logs information. DBMS_MGWADM.BASIC_LOGGING generates the least information while DBMS_MGWADM.TRACE_DEBUG_LOGGING generates the most information.</td>
</tr>
<tr>
<td>agent_name</td>
<td>Identifies the Messaging Gateway agent. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.</td>
</tr>
</tbody>
</table>

See Also:

Table 103-3 for details on the log_level parameter

103.5.28 SET_OPTION Procedure

This procedure sets a Messaging Gateway configuration option. It can be used to set an agent option, a messaging link option, or a propagation job option.

Syntax

```
DBMS_MGWADM.SET_OPTION (target_type IN PLS_INTEGER,
                        target_name IN VARCHAR2,
                        option_name IN VARCHAR2,
                        option_value IN VARCHAR2,
                        encrypted IN BOOLEAN DEFAULT FALSE )
```
Parameters

Table 103-47  SET_OPTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_type</td>
<td>Specifies the target type of the Messaging Gateway entity:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_MGWADM.AGENT_JAVA_PROP to set a Java System property for a Messaging</td>
</tr>
<tr>
<td></td>
<td>Gateway agent</td>
</tr>
<tr>
<td></td>
<td>- DBMS_MGWADM.MSGLINK_OPTION to set a messaging link option</td>
</tr>
<tr>
<td></td>
<td>- DBMS_MGWADM.JOB_OPTION to set a propagation job option</td>
</tr>
<tr>
<td>target_name</td>
<td>Name or identifier of the target. The value for this parameter depends</td>
</tr>
<tr>
<td></td>
<td>on the value specified for target_type parameter. This must not be NULL.</td>
</tr>
<tr>
<td>option_name</td>
<td>Option name. This must not be NULL.</td>
</tr>
<tr>
<td>option_value</td>
<td>Option value</td>
</tr>
<tr>
<td>encrypted</td>
<td>Indicates whether the value should be stored as encrypted:</td>
</tr>
<tr>
<td></td>
<td>- TRUE if the value should be stored in an encrypted form</td>
</tr>
<tr>
<td></td>
<td>- FALSE if the value should be stored in a cleartext form</td>
</tr>
</tbody>
</table>

See Also:
- Table 103-10 regarding options for the option_type parameter

Usage Notes

DBMS_MGWADM.AGENT_JAVA_PROP Target

The procedure will store an agent option used to set a Java System property when the Messaging Gateway agent is started. The agent must be restarted for the change to take effect.

The parameters are interpreted as follows:
- target_name specifies the name of the Messaging Gateway agent.
  DBMS_MGWADM.DEFAULT_AGENT can be used for the default agent.
- option_name specifies the Java System property
- encrypted can be either TRUE or FALSE

DBMS_MGWADM.MSGLINK_OPTION Target

The procedure will set or alter a single option for a Messaging Gateway messaging system link. This is equivalent to calling DBMS_MGWADM.ALTER_MSGSYSTEM_LINK and using the options parameter to set an option.

The parameters are interpreted as follows:
- target_name specifies the name of the message system link
- option_name specifies the option to set
- encrypted must be FALSE
DBMS_MGWADM.JOB_OPTION Target

The procedure will set or alter a single option for a Messaging Gateway propagation job. This is equivalent to calling DBMS_MGWADM.ALTER_JOB and using the options parameter to set an option.

The parameters are interpreted as follows:

- target_name specifies the name of the propagation job
- option_name specifies the option to set
- encrypted must be FALSE

103.5.29 SHUTDOWN Procedures

This procedure shuts down the Messaging Gateway agent. No propagation activity occurs until Messaging Gateway is restarted.

Syntax

```sql
DBMS_MGWADM.SHUTDOWN (
    sdmode      IN BINARY_INTEGER DEFAULT DBMS_MGWADM.SHUTDOWN_NORMAL);
```

```sql
DBMS_MGWADM.SHUTDOWN (
    agent_name  IN VARCHAR2);
```

Parameters

Table 103-48  SHUTDOWN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdmode</td>
<td>The shutdown mode. The only value currently supported is</td>
</tr>
<tr>
<td></td>
<td>DBMS_MGWADM.SHUTDOWN_NORMAL for normal shutdown. The Messaging Gateway agent</td>
</tr>
<tr>
<td></td>
<td>may attempt to complete any propagation work currently in progress.</td>
</tr>
<tr>
<td>agent_name</td>
<td>Identifies the Messaging Gateway agent. DBMS_MGWADM.DEFAULT_AGENT specifies</td>
</tr>
<tr>
<td></td>
<td>the default agent.</td>
</tr>
</tbody>
</table>

Usage Notes

The Messaging Gateway default agent is shut down if no agent name is specified.

103.5.30 STARTUP Procedures

This procedure starts the Messaging Gateway agent. It must be called before any propagation activity can take place.

Syntax

```sql
DBMS_MGWADM.STARTUP (
    instance       IN  BINARY_INTEGER DEFAULT 0,
    force          IN  BINARY_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE);
```

```sql
DBMS_MGWADM.STARTUP (
    agent_name     IN  VARCHAR2);
```
Parameters

Table 103-49  STARTUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>Specifies which instance can run the job queue job used to start the Messaging Gateway agent. If this is zero, then the job can be run by any instance. <strong>Caution: This parameter has been deprecated.</strong></td>
</tr>
<tr>
<td>force</td>
<td>If this is <code>DBMS_MGWADM.FORCE</code>, then any positive integer is acceptable as the job instance. If this is <code>DBMS_MGWADM.NO_FORCE</code> (the default), then the specified instance must be running; otherwise the routine raises an exception. <strong>Caution: This parameter has been deprecated.</strong></td>
</tr>
<tr>
<td>agent_name</td>
<td>Identifies the Messaging Gateway agent. <code>DBMS_MGWADM.DEFAULT_AGENT</code> specifies the default agent.</td>
</tr>
</tbody>
</table>

Usage Notes

- The Messaging Gateway default agent will be started if an agent name is not specified.
- The `force` and `instance` parameters are no longer used and will be ignored. If the `instance` affinity parameters were being used to start the default agent on a specific instance, the administrator will need to create a database service and then assign that service to the default agent using the `DBMS_MGWADM.ALTER_AGENT` procedure.
- The Messaging Gateway agent cannot be started until an agent user has been configured by the `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT` subprograms.

103.5.31 UNREGISTER_FOREIGN_QUEUE Procedure

This procedure removes a non-Oracle queue entity in Messaging Gateway.

Syntax

```sql
DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE(
    name      IN VARCHAR2,
    linkname  IN VARCHAR2);
```

Parameters

Table 103-50  UNREGISTER_FOREIGN_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The queue name</td>
</tr>
<tr>
<td>linkname</td>
<td>The link name for the messaging system on which the queue exists</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure does not remove the physical queue in the non-Oracle messaging system.
All propagation jobs, subscribers and schedules referencing this queue must be removed before it can be unregistered. This procedure fails if a propagation job, subscriber, or propagation schedule references the non-Oracle queue.

103.5.32 UNSCHEDULE_PROPAGATION Procedure

This deprecated procedure removes a propagation schedule.

Note:
This subprogram has been deprecated as a result of improved technology (see REMOVE_JOB Procedure), and is retained only for reasons of backward compatibility.

Syntax

DBMS_MGWADM.UNSCHEDULE_PROPAGATION (  
    schedule_id  IN VARCHAR2 );

Parameters

Table 103-51  UNSCHEDULE_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_id</td>
<td>Identifies the propagation schedule to be removed</td>
</tr>
</tbody>
</table>
DBMS_MGWMSG provides object types used by the canonical message types to convert message bodies, and methods, constants, and subprograms for working with Messaging Gateway message types.

This chapter contains the following topics:

- Security Model
- Constants
- Types
- Summary of DBMS_MGWMSG Subprograms

See Also:

DBMS_MGWADM which describes the Messaging Gateway administrative interface, DBMS_MGWADM

104.1 DBMS_MGWMSG Security Model

The EXECUTE privilege is granted to PUBLIC on all types defined in the DBMS_MGWMSG package as well as the canonical types. The DBMS_MGWMSG packages and object types are owned by SYS.

Note:

You must run the catmgw.sql script to load the Messaging Gateway packages and object types into the database. Refer to the Oracle Database Advanced Queuing User's Guide for information on loading database objects and using DBMS_MGWMSG.

104.2 DBMS_MGWMSG Constants

The DBMS_MGWMSG package defines constants for specifying various parameter values.

Table 104-1  DBMS_MGWMSG Constants: Value Types and Constants Representing the Type of Value for a SYS.MGW_NAME_VALUE_T Object

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 1</td>
</tr>
<tr>
<td>RAW_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 2</td>
</tr>
</tbody>
</table>
Table 104-1   (Cont.) DBMS_MGWMSG Constants: Value Types and Constants Representing the Type of Value for a SYS.MGW_NAME_VALUE_T Object

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 3</td>
</tr>
<tr>
<td>BYTE_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 4</td>
</tr>
<tr>
<td>SHORT_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 5</td>
</tr>
<tr>
<td>INTEGER_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 6</td>
</tr>
<tr>
<td>LONG_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 7</td>
</tr>
<tr>
<td>FLOAT_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 8</td>
</tr>
<tr>
<td>DOUBLE_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 9</td>
</tr>
<tr>
<td>DATE_VALUE</td>
<td>CONSTANT BINARY_INTEGER := 10</td>
</tr>
</tbody>
</table>

Table 104-2   DBMS_MGWMSG Constants: Boolean Values—Constants Representing a Boolean as a Numeric Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN_FALSE</td>
<td>CONSTANT BINARY_INTEGER := 0</td>
</tr>
<tr>
<td>BOOLEAN_TRUE</td>
<td>CONSTANT BINARY_INTEGER := 1</td>
</tr>
</tbody>
</table>

Table 104-3   DBMS_MGWMSG Constants: Case Comparisons

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE_SENSITIVE</td>
<td>CONSTANT BINARY_INTEGER := 0</td>
</tr>
<tr>
<td>CASE_INSENSITIVE</td>
<td>CONSTANT BINARY_INTEGER := 1</td>
</tr>
</tbody>
</table>

Table 104-4   Constants for the TIB/Rendezvous field type

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBRVMSG_BOOL</td>
<td>CONSTANT INTEGER := 1</td>
</tr>
<tr>
<td>TIBRVMSG_F32</td>
<td>CONSTANT INTEGER := 2</td>
</tr>
<tr>
<td>TIBRVMSG_F64</td>
<td>CONSTANT INTEGER := 3</td>
</tr>
<tr>
<td>TIBRVMSG_I8</td>
<td>CONSTANT INTEGER := 4</td>
</tr>
<tr>
<td>TIBRVMSG_I16</td>
<td>CONSTANT INTEGER := 5</td>
</tr>
<tr>
<td>TIBRVMSG_I32</td>
<td>CONSTANT INTEGER := 6</td>
</tr>
<tr>
<td>TIBRVMSG_I64</td>
<td>CONSTANT INTEGER := 7</td>
</tr>
<tr>
<td>TIBRVMSG_IPADDR32</td>
<td>CONSTANT INTEGER := 8</td>
</tr>
<tr>
<td>TIBRVMSG_IPPORT16</td>
<td>CONSTANT INTEGER := 9</td>
</tr>
<tr>
<td>TIBRVMSG_DATETIME</td>
<td>CONSTANT INTEGER := 10</td>
</tr>
<tr>
<td>TIBRVMSG_F32ARRAY</td>
<td>CONSTANT INTEGER := 11</td>
</tr>
<tr>
<td>TIBRVMSG_F64ARRAY</td>
<td>CONSTANT INTEGER := 12</td>
</tr>
</tbody>
</table>
Table 104-4  (Cont.) Constants for the TIB/Rendezvous field type

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBRVMSG_I8ARRAY</td>
<td>CONSTANT INTEGER := 13</td>
</tr>
<tr>
<td>TIBRVMSG_I16ARRAY</td>
<td>CONSTANT INTEGER := 14</td>
</tr>
<tr>
<td>TIBRVMSG_I32ARRAY</td>
<td>CONSTANT INTEGER := 15</td>
</tr>
<tr>
<td>TIBRVMSG_I64ARRAY</td>
<td>CONSTANT INTEGER := 16</td>
</tr>
<tr>
<td>TIBRVMSG_OPAQUE</td>
<td>CONSTANT INTEGER := 17</td>
</tr>
<tr>
<td>TIBRVMSG_STRING</td>
<td>CONSTANT INTEGER := 18</td>
</tr>
<tr>
<td>TIBRVMSG_XML</td>
<td>CONSTANT INTEGER := 19</td>
</tr>
</tbody>
</table>

104.3 DBMS_MGWMSG Types

DBMS_MGWMSG uses types to specify certain information.

- SYS.MGW_NAME_VALUE_T Type
- SYS.MGW_NAME_VALUE_T Type-Attribute Mapping
- SYS.MGW_NAME_TYPE_ARRAY_T Type
- SYS.MGW_TEXT_VALUE_T Type
- SYS.MGW_RAW_VALUE_T Type
- SYS.MGW_BASIC_MSG_T Type
- SYS.MGW_NUMBER_ARRAY_T Type
- SYS.MGW_TIBRV_FIELD_T Type
- SYS.MGW_TIBRV_MSG_T Type

104.3.1 DBMS_MGWMSG SYS.MGW_NAME_VALUE_T Type

This type specifies a named value.

The name attribute, type attribute, and one of the <>_value attributes are typically not NULL.

Syntax

```
TYPE SYS.MGW_NAME_VALUE_T IS OBJECT(
   name               VARCHAR2(250),
   type               INTEGER,
   integer_value      INTEGER,
   number_value       NUMBER,
   text_value         VARCHAR2(4000),
   raw_value          RAW(2000),
   date_value         DATE,

   -- Methods
   STATIC FUNCTION CONSTRUCT
   RETURN SYS.MGW_NAME_VALUE_T,

   STATIC FUNCTION CONSTRUCT_BOOLEAN (
      name    IN VARCHAR2,  
```

value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_BYTE (  
   name  IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_SHORT (  
   name  IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_INTEGER (  
   name  IN VARCHAR2,
   value  IN INTEGER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_LONG (  
   name  IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_FLOAT (  
   name  IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DOUBLE (  
   name  IN VARCHAR2,
   value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_TEXT (  
   name  IN VARCHAR2,
   value  IN VARCHAR2 )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_RAW (  
   name  IN VARCHAR2,
   value  IN RAW )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DATE (  
   name  IN VARCHAR2,
   value  IN DATE )
RETURN SYS.MGW_NAME_VALUE_T );

Attributes

Table 104-5 SYS.MGW_NAME_VALUE_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name associated with the value</td>
</tr>
<tr>
<td>type</td>
<td>Value type. Refer to the DBMS_MGWMSG.&lt;&gt;_VALUE constants in Table 104-1. This indicates which Java datatype and class are associated with the value. It also indicates which attribute stores the value.</td>
</tr>
</tbody>
</table>
Table 104-5  (Cont.) SYS.MGW_NAME_VALUE_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer_value</td>
<td>Stores a numeric integer value</td>
</tr>
<tr>
<td>number_value</td>
<td>Stores a numeric float or large integer value</td>
</tr>
<tr>
<td>text_value</td>
<td>Stores a text value</td>
</tr>
<tr>
<td>raw_value</td>
<td>Stores a RAW (bytes) value</td>
</tr>
<tr>
<td>date_value</td>
<td>Stores a date value</td>
</tr>
</tbody>
</table>

SYS.MGW_NAME_VALUE_T Type-Attribute Mapping

Table 104-6 shows the mapping between the value type and the attribute used to store the value.

Table 104-6  SYS.MGW_NAME_VALUE_T Type Attribute Mapping

<table>
<thead>
<tr>
<th>Type</th>
<th>Value Stored in Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_MGWMSG.TEXT_VALUE</td>
<td>text_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.RAW_VALUE</td>
<td>raw_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.BOOLEAN_VALUE</td>
<td>integer_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.BYTE_VALUE</td>
<td>integer_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.SHORT_VALUE</td>
<td>integer_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.INTEGER_VALUE</td>
<td>integer_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.LONG_VALUE</td>
<td>number_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.FLOAT_VALUE</td>
<td>number_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.DOUBLE_VALUE</td>
<td>number_value</td>
</tr>
<tr>
<td>DBMS_MGWMSG.DATE_VALUE</td>
<td>date_value</td>
</tr>
</tbody>
</table>

CONSTRUCT Method

This method constructs a new SYS.MGW_NAME_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

```sql
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_NAME_VALUE_T;
```

CONSTRUCT_TYPE Methods

These methods construct a new SYS.MGW_NAME_VALUE_T instance initialized with the value of a specific type. Each method sets the name and type attributes and one of the <>_value attributes, as shown in the mappings in Table 104-6.

Syntax

```sql
STATIC FUNCTION CONSTRUCT_<>
{
    name   IN VARCHAR2,
```
value IN datatype )
RETURN SYS.MGW_NAME_VALUE_T;

Usage Notes
The construct_boolean method sets the value to either DBMS_MGWMSG.BOOLEAN_TRUE or DBMS_MGWMSG.BOOLEAN_FALSE.

104.3.2 DBMS_MGWMSG SYS.MGW_NAME_TYPE_ARRAY_T Type
This type specifies an array of name-value pairs. An object of SYS.MGW_NAME_VALUE_ARRAY_T type can have up to 1024 elements.

Syntax
TYPE SYS.MGW_NAME_VALUE_ARRAY_T
AS VARRAY (1024) OF SYS.MGW_NAME_VALUE_T;

104.3.3 DBMS_MGWMSG SYS.MGW_TEXT_VALUE_T Type
This type specifies a TEXT value. It can store a large value as a CLOB or a smaller value (size <= 4000) as VARCHAR2. Only one of the `< >` value attributes should be set.

Syntax
TYPE SYS.MGW_TEXT_VALUE_T IS OBJECT(
   small_value VARCHAR2(4000),
   large_value CLOB,

   -- Methods
   STATIC FUNCTION CONSTRUCT
   RETURN SYS.MGW_TEXT_VALUE_T);

Attributes

Table 104-7  SYS.MGW_TEXT_VALUE_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>small_value</td>
<td>Small TEXT value. Used for values &lt;= 4000.</td>
</tr>
<tr>
<td>large_value</td>
<td>Large TEXT value. Used when the value is too large for the small_value attribute.</td>
</tr>
</tbody>
</table>

CONSTRUCT Method
This method constructs a new SYS.MGW_TEXT_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_TEXT_VALUE_T;
104.3.4 DBMS_MGWMSG SYS.MGW_RAW_VALUE_T Type

This type specifies a RAW value. This type can store a large value as a BLOB or a smaller value (size <= 2000) as RAW. You must set no more than one of the < >_value attributes.

Syntax

TYPE SYS.MGW_RAW_VALUE_T IS OBJECT(
    small_value RAW(2000),
    large_value BLOB,

--Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_RAW_VALUE_T);

Attributes

Table 104-8 SYS.MGW_RAW_VALUE_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>small_value</td>
<td>Small RAW (bytes) value &lt;= 2000</td>
</tr>
<tr>
<td>large_value</td>
<td>Large RAW value. Used when the value is too large for the small_value attribute.</td>
</tr>
</tbody>
</table>

CONSTRUCT Method

This method constructs a new SYS.MGW_RAW_VALUE_T instance. All attributes are assigned a value of NULL.

Syntax

STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_RAW_VALUE_T;

104.3.5 DBMS_MGWMSG SYS.MGW_BASIC_MSG_T Type

This is a canonical type for a basic TEXT or RAW message. Only a single TEXT or RAW value is typically set. An object of this type must not have both TEXT and RAW set to a not NULL value at the same time.

Syntax

TYPE SYS.MGW_BASIC_MSG_T IS OBJECT(
    header   SYS.MGW_NAME_VALUE_ARRAY_T,
    text_body SYS.MGW_TEXT_VALUE_T,
    raw_body  SYS.MGW_RAW_VALUE_T,

--Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_BASIC_MSG_T);
### Attributes

**Table 104-9  SYS.MGW_BASIC_MSG_T Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>Message header information as an array of name-value pairs</td>
</tr>
<tr>
<td>text_body</td>
<td>Message body for a TEXT message</td>
</tr>
<tr>
<td>raw_body</td>
<td>Message body for a RAW (bytes) message</td>
</tr>
</tbody>
</table>

### CONSTRUCT Method

This method constructs a new SYS.MGW_BASIC_MSG_T instance. All attributes are assigned a value of NULL.

**Syntax**

```sql
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_BASIC_MSG_T;
```

### 104.3.6 DBMS_MGWMSG SYS.MGW_NUMBER_ARRAY_T Type

A type that specifies an array of numbers.

**Syntax**

```sql
TYPE SYS.MGW_NUMBER_ARRAY_T AS VARRAY(1024) OF NUMBER;
```

### 104.3.7 DBMS_MGWMSG SYS.MGW_TIBRV_FIELD_T Type

This type represents a TIB/Rendezvous message field, typically used in a read-only fashion to retrieve field information from a SYS.MGW_TIBRV_MSG_T instance.

**Syntax**

```sql
TYPE SYS.MGW_TIBRV_FIELD_T IS OBJECT(
    field_name       VARCHAR2(256),
    field_id         INTEGER,
    field_type       INTEGER,
    number_value     NUMBER,
    number_array_value  SYS.MGW_NUMBER_ARRAY_T,
    text_value       VARCHAR2(4000),
    raw_value        RAW(2000),
    date_value       DATE,
    clob_value       CLOB,
    blob_value       BLOB);
```

### Attributes

**Table 104-10  SYS.MGW_TIBRV_FIELD_T Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_name</td>
<td>Field name. This will be NULL if the field has no name.</td>
</tr>
</tbody>
</table>
Table 104-10 (Cont.) SYS.MGW_TIBRV_FIELD_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_id</td>
<td>Field identifier. If the field identifier is zero (0), then that field is</td>
</tr>
<tr>
<td></td>
<td>considered not to have a field identifier. Otherwise the field identifier</td>
</tr>
<tr>
<td></td>
<td>is a nonzero value that is unique for all fields of that message.</td>
</tr>
<tr>
<td>field_type</td>
<td>Field wire format datatype. The DBMS_MGWMSG.TIBRVMSG_&lt;&gt; constants represent</td>
</tr>
<tr>
<td></td>
<td>valid values for this attribute. The value of this field discriminates which</td>
</tr>
<tr>
<td></td>
<td>value attribute is used to store the field data.</td>
</tr>
<tr>
<td>number_value</td>
<td>Used to store a numeric value</td>
</tr>
<tr>
<td>number_array_value</td>
<td>Used to store a numeric array value</td>
</tr>
<tr>
<td>text_value</td>
<td>Used to store a small text value</td>
</tr>
<tr>
<td>raw_value</td>
<td>Used to store a small raw value</td>
</tr>
<tr>
<td>date_value</td>
<td>Used to store a date value</td>
</tr>
<tr>
<td>clob_value</td>
<td>Used to store a large text value. This is used when the text data will</td>
</tr>
<tr>
<td></td>
<td>not fit in text_value, that is, when size is larger than 4000.</td>
</tr>
<tr>
<td>blob_value</td>
<td>Used to store a large raw value. This is used when the raw data will</td>
</tr>
<tr>
<td></td>
<td>not fit in raw_value; that is, when size is larger than 2000.</td>
</tr>
</tbody>
</table>

SYS.MGW_TIBRV_FIELD_T Type and Attribute Mapping

Table 104-11 describes the mapping in type SYS.MGW_TIBRV_FIELD_T between the field type and attribute used to store the value.

Table 104-11  SYS.MGW_TIBRV_FIELD_T Type and Attribute Mapping

<table>
<thead>
<tr>
<th>Field Type (DBMS_MGWMSG constant)</th>
<th>Value Stored in Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBRVMSG_BOOL</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_F32</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_F64</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_I8</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_I16</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_I32</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_I64</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_IPADDR32</td>
<td>text_value</td>
</tr>
<tr>
<td>TIBRVMSG_IPPORT16</td>
<td>number_value</td>
</tr>
<tr>
<td>TIBRVMSG_DATETIME</td>
<td>date_value</td>
</tr>
<tr>
<td>TIBRVMSG_F32ARRAY</td>
<td>number_array_value</td>
</tr>
<tr>
<td>TIBRVMSG_F64ARRAY</td>
<td>number_array_value</td>
</tr>
<tr>
<td>TIBRVMSG_I8ARRAY</td>
<td>number_array_value</td>
</tr>
<tr>
<td>TIBRVMSG_I16ARRAY</td>
<td>number_array_value</td>
</tr>
<tr>
<td>TIBRVMSG_I32ARRAY</td>
<td>number_array_value</td>
</tr>
</tbody>
</table>
### Table 104-11 (Cont.) SYS.MGW_TIBRV_FIELD_T Type and Attribute Mapping

<table>
<thead>
<tr>
<th>Field Type (DBMS_MGWMSG constant)</th>
<th>Value Stored in Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBRVMSG_I64ARRAY</td>
<td>number_array_value</td>
</tr>
<tr>
<td>TIBRVMSG_OPAQUE</td>
<td>raw_value or blob_value</td>
</tr>
<tr>
<td>TIBRVMSG_STRING</td>
<td>text_value or clob_value</td>
</tr>
<tr>
<td>TIBRVMSG_XML</td>
<td>raw_value or blob_value</td>
</tr>
</tbody>
</table>

### 104.3.8 DBMS_MGWMSG SYS.MGW_TIBRV_MSG_T Type

This type represents a TIB/Rendezvous message. You must never directly reference the attributes of this type. Instead use the type methods.

**Syntax**

```sql
TYPE SYS.MGW_TIBRV_MSG_T IS OBJECT(
    send_subject    VARCHAR2(256),
    reply_subject   VARCHAR2(256),
    cm_time_limit   NUMBER,
    cm_sender_name  VARCHAR2(256),
    cm_sequence_num NUMBER,
    fields          SYS.MGW_TIBRV_IFIELDS_T,
    clob_data1      CLOB,
    clob_data2      CLOB,
    clob_data3      CLOB,
    blob_data1      BLOB,
    blob_data2      BLOB,
    blob_data3      BLOB,

    STATIC FUNCTION construct
    RETURN SYS.MGW_TIBRV_MSG_T,

    MEMBER PROCEDURE add_bool ( name IN VARCHAR2, id IN INTEGER, value IN INTEGER ),

    MEMBER PROCEDURE add_f32 ( name IN VARCHAR2, id IN INTEGER, value IN FLOAT ),

    MEMBER PROCEDURE add_f64 ( name IN VARCHAR2, id IN INTEGER, value IN DOUBLE ),

    MEMBER PROCEDURE add_i8 ( name IN VARCHAR2, id IN INTEGER, value IN INTEGER ),

    MEMBER PROCEDURE add_i16 ( name IN VARCHAR2, id IN INTEGER, value IN INTEGER ),
```

---

ORACLE
value IN INTEGER ),

MEMBER PROCEDURE add_i32 (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN INTEGER ),

MEMBER PROCEDURE add_i64 (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN NUMBER ),

MEMBER PROCEDURE add_ipaddr32 (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN VARCHAR2 ),

MEMBER PROCEDURE add_ipport16 (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN INTEGER ),

MEMBER PROCEDURE add_datetime (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN DATE ),

MEMBER PROCEDURE add_f32array (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_f64array (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i8array (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i16array (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i32array (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i64array (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_string (  
    name IN VARCHAR2,  
    id IN INTEGER,  
    value IN VARCHAR2 ),
MEMBER PROCEDURE add_string (  
  name IN VARCHAR2,  
  id IN INTEGER,  
  value IN CLOB ),

MEMBER PROCEDURE add_opaque (  
  name IN VARCHAR2,  
  id IN INTEGER,  
  value IN RAW ),

MEMBER PROCEDURE add_opaque (  
  name IN VARCHAR2,  
  id IN INTEGER,  
  value IN BLOB ),

MEMBER PROCEDURE add_xml (  
  name IN VARCHAR2,  
  id IN INTEGER,  
  value IN RAW ),

MEMBER PROCEDURE add_xml (  
  name IN VARCHAR2,  
  id IN INTEGER,  
  value IN BLOB ),

MEMBER PROCEDURE set_send_subject (  
  value IN VARCHAR2 ),

MEMBER PROCEDURE set_reply_subject (  
  value IN VARCHAR2 ),

MEMBER PROCEDURE set_cm_time_limit (  
  value IN NUMBER ),

MEMBER PROCEDURE set_cm_sender_name (  
  value IN VARCHAR2 ),

MEMBER PROCEDURE set_cm_sequence_num (  
  value IN NUMBER ),

MEMBER FUNCTION get_send_subject  
RETURN VARCHAR2,

MEMBER FUNCTION get_reply_subject  
RETURN VARCHAR2,

MEMBER FUNCTION get_cm_time_limit  
RETURN NUMBER,

MEMBER FUNCTION get_cm_sender_name  
RETURN VARCHAR2,

MEMBER FUNCTION get_cm_sequence_num  
RETURN NUMBER,

MEMBER FUNCTION get_field_count  
RETURN INTEGER,

MEMBER FUNCTION get_field (  
  idx IN INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION get_field_by_name (  
    name   IN  VARCHAR2 )  
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION get_field_by_id (  
    id     IN  INTEGER )  
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION find_field_name (  
    name   IN  VARCHAR2,  
    start_idx IN INTEGER )  
RETURN INTEGER,

MEMBER FUNCTION find_field_id (  
    id     IN INTEGER,  
    start_idx IN INTEGER )  
RETURN INTEGER
);

Attributes

Table 104-12   SYS.MGW_TIBRV_MSG_T Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send_subject</td>
<td>Send subject name</td>
</tr>
<tr>
<td>reply_subject</td>
<td>Reply subject name</td>
</tr>
<tr>
<td>cm_time_limit</td>
<td>Time limit for a certified message</td>
</tr>
<tr>
<td>cm_sender_name</td>
<td>Sender name of a certified message</td>
</tr>
<tr>
<td>cm_sequence_num</td>
<td>Sequence number of a certified message</td>
</tr>
<tr>
<td>fields</td>
<td>Collection of message fields</td>
</tr>
<tr>
<td>clob_data1</td>
<td>Used to store a large text value</td>
</tr>
<tr>
<td>clob_data2</td>
<td>Used to store a large text value</td>
</tr>
<tr>
<td>clob_data3</td>
<td>Used to store a large text value</td>
</tr>
<tr>
<td>blob_data1</td>
<td>Used to store a large raw value</td>
</tr>
<tr>
<td>blob_data2</td>
<td>Used to store a large raw value</td>
</tr>
<tr>
<td>blob_data3</td>
<td>Used to store a large raw value</td>
</tr>
</tbody>
</table>

Construct Method

Constructs a new SYS.MGW_TIBRV_MSG_T instance. All attributes are set to NULL.

Syntax

STATIC FUNCTION construct  
RETURN SYS.MGW_TIBRV_MSG_T;

ADD_<> Methods

Adds a new field to the message.

Syntax
MEMBER PROCEDURE ADD_<> (  
    name  IN  VARCHAR2,  
    id    IN  INTEGER,  
    value IN datatype );

Parameters

Table 104-13  SYS.MGW_TIBRV_MSG_T ADD_<> Method Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Field name</td>
</tr>
<tr>
<td>id</td>
<td>Field identifier</td>
</tr>
<tr>
<td>value</td>
<td>Field data</td>
</tr>
</tbody>
</table>

Table 104-14 shows, for each add method, the field type that will be assigned and valid values for the field data.

Table 104-14  MGW_TIBRV_MSG_T Add Method Field Types

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Field Type Assigned</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_bool</td>
<td>TIBRVMSG_BOOL</td>
<td>Valid values: 0 (false), 1 (true)</td>
</tr>
<tr>
<td>add_f32</td>
<td>TIBRVMSG_F32</td>
<td>n/a</td>
</tr>
<tr>
<td>add_f64</td>
<td>TIBRVMSG_F64</td>
<td>n/a</td>
</tr>
<tr>
<td>add_i8</td>
<td>TIBRVMSG_I8</td>
<td>Valid range: -128...127</td>
</tr>
<tr>
<td>add_i16</td>
<td>TIBRVMSG_I16</td>
<td>Valid range: -32768...32767</td>
</tr>
<tr>
<td>add_i32</td>
<td>TIBRVMSG_I32</td>
<td>Valid range: -2147483648...2147483647</td>
</tr>
<tr>
<td>add_i64</td>
<td>TIBRVMSG_I64</td>
<td>n/a</td>
</tr>
<tr>
<td>add_ipaddr32</td>
<td>TIBRVMSG_IPADDR32</td>
<td>n/a</td>
</tr>
<tr>
<td>add_ipport16</td>
<td>TIBRVMSG_IPPORT16</td>
<td>n/a</td>
</tr>
<tr>
<td>add_datetime</td>
<td>TIBRVMSG_DATETIME</td>
<td>n/a</td>
</tr>
<tr>
<td>add_f32array</td>
<td>TIBRVMSG_F32ARRAY</td>
<td>n/a</td>
</tr>
<tr>
<td>add_f64array</td>
<td>TIBRVMSG_F64ARRAY</td>
<td>n/a</td>
</tr>
<tr>
<td>add_i8array</td>
<td>TIBRVMSG_I8ARRAY</td>
<td>Valid range: -128...127</td>
</tr>
<tr>
<td>add_i16array</td>
<td>TIBRVMSG_I16ARRAY</td>
<td>Valid range: -32768...32767</td>
</tr>
<tr>
<td>add_i32array</td>
<td>TIBRVMSG_I32ARRAY</td>
<td>Valid range: -2147483648...2147483647</td>
</tr>
<tr>
<td>add_i64array</td>
<td>TIBRVMSG_I64ARRAY</td>
<td>n/a</td>
</tr>
<tr>
<td>addOpaque</td>
<td>TIBRVMSG_OPAQUE</td>
<td>Value stored as RAW if size &lt; 2000; otherwise value stored in BLOB</td>
</tr>
<tr>
<td>add_string</td>
<td>TIBRVMSG_STRING</td>
<td>Value stored as VARCHAR2 if size &lt; 4000; otherwise value stored in CLOB</td>
</tr>
<tr>
<td>add_xml</td>
<td>TIBRVMSG_XML</td>
<td>Value stored as RAW if size &lt; 2000; otherwise value stored in BLOB</td>
</tr>
</tbody>
</table>
SET_<> Methods
Accessor methods to set an instance attribute to a specific value.

Syntax
MEMBER PROCEDURE SET_<> (value IN datatype);

Parameters

Table 104-15   SYS.MGW_TIBRV_MSG_T SET_<> Method Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value to be assigned</td>
</tr>
</tbody>
</table>

GET_<> Methods
Accessor methods to retrieve the value for an instance attribute.

Syntax
MEMBER PROCEDURE GET_<>
RETURN datatype;

Parameters
None

Return Values
Returns the attribute value.

GET_FIELD_COUNT Function
Gets the number of message fields.

Syntax
MEMBER PROCEDURE get_field_count
RETURN INTEGER;

Parameters
None

Return Values
Returns the number of fields, or zero (0) if there are none.

GET_FIELD Function
Retrieves field information for the field having a given field collection index. This method should only be called if the GET_FIELD_COUNT Function returns a nonzero value and idx must specify a valid collection index; that is, 1<=idx<=get_field_count().

Syntax
MEMBER PROCEDURE get_field (    
    idx IN INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T;

Parameters

Table 104-16   SYS.MGW_TIBRV_MSG_T GET_FIELD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idx</td>
<td>Specifies the 1-based field collection index of the field to retrieve</td>
</tr>
</tbody>
</table>

Note:
A 1-based index begins at one (1) instead of zero (0).

Return Values

Returns the field information.

GET_FIELD_BY_NAME Function

Retrieves field information for the first field that has a given field name. The name comparison is case-sensitive.

Syntax

MEMBER PROCEDURE get_field_by_name (    
    name IN VARCHAR2 )
RETURN SYS.MGW_TIBRV_FIELD_T;

Parameters

Table 104-17   SYS.MGW_TIBRV_MSG_T GET_FIELD_BY_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the field name to search for. This can be NULL to find the first field that does not have a field name.</td>
</tr>
</tbody>
</table>

Return Values

Returns the field information, or NULL if no match was found.

GET_FIELD_BY_ID Function

Retrieves field information for the first field that has a given field identifier.

A field can have either a unique identifier or no identifier. If the field identifier value is zero (0) or NULL, then the field is considered to have no identifier. Otherwise, the identifier is a nonzero value that is unique for all the fields of this message.

Syntax
MEMBER PROCEDURE get_field_by_id (  
    id   IN  INTEGER  )  
RETURN SYS.MGW_TIBRV_FIELD_T;

Parameters

Table 104-18  SYS.MGW_TIBRV_MSG_T GET_FIELD_BY_ID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Specifies the field identifier to search for. This can be zero (0) or NULL to find the first field that does not have an identifier.</td>
</tr>
</tbody>
</table>

Return Values

Returns the field information, or NULL if no match was found.

FIND_FIELD_NAME Function

Searches for a field with a given field name, starting from a given index of the field collection. It returns the index of that field. The name comparison is case-sensitive. This function is useful for finding all the fields that have the same name.

Syntax

MEMBER PROCEDURE find_field_name (  
    name      IN  VARCHAR2,  
    start_idx IN  INTEGER  )  
RETURN INTEGER;

Parameters

Table 104-19  SYS.MGW_TIBRV_MSG_T FIND_FIELD_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the field name to search for. This can be NULL to search for a field that does not have a field name.</td>
</tr>
<tr>
<td>start_idx</td>
<td>Specifies the 1-based field collection index from which the search should start.</td>
</tr>
</tbody>
</table>

Return Values

Returns the field index (> 0) if a match was found, or zero (0) if no match was found.

FIND_FIELD_ID Function

Searches for a field with a given field identifier, starting from a given index of the field collection. It returns the index of that field.

Syntax

MEMBER PROCEDURE find_field_id (  
    id        IN  INTEGER,  
    start_idx IN INTEGER  )  
RETURN INTEGER;

Parameters
Table 104-20  SYS.MGW_TIBRV_MSG_T FIND_FIELD_ID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Specifies the field identifier to search for. This can be zero (0) or NULL to find a field that does not have an identifier.</td>
</tr>
<tr>
<td>start_idx</td>
<td>Specifies the 1-based field collection index from which the search should start.</td>
</tr>
</tbody>
</table>

Return Values

Returns the field index (> 0) if a match was found, or zero (0) if no match was found.

104.4 Summary of DBMS_MGWMSG Subprograms

This table lists the DDBMS_MGWMSG subprograms and briefly describes them.

Table 104-21  DBMS_MGWMSG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCR_TO_XML Function</td>
<td>Converts a SYS.ANYDATA object encapsulating a row LCR (LCR$_ROW_RECORD) or a DDL LCR (LCR$_DDL_RECORD) to a SYS.XMLTYPE object</td>
</tr>
<tr>
<td>NVARRAY_ADD Procedure</td>
<td>Appends a name-value element to the end of a name-value array</td>
</tr>
<tr>
<td>NVARRAY_FIND_NAME Function</td>
<td>Searches a name-value array for the element with the name you specify in p_name</td>
</tr>
<tr>
<td>NVARRAY_FIND_NAME_TY PE Function</td>
<td>Searches a name-value array for an element with the name and value type you specify</td>
</tr>
<tr>
<td>NVARRAY_GET Function</td>
<td>Gets the name-value element of the name you specify in p_name from a name-value array</td>
</tr>
<tr>
<td>NVARRAY_GET_BOOLEAN Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the BOOLEAN_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_BYTE Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the BYTE_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_DATE Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the DATE_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_DOUBLE Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the DOUBLE_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_FLOAT Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the FLOAT_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_INTEGER Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the INTEGER_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_LONG Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the LONG_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_RAW Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the RAW_VALUE value type</td>
</tr>
</tbody>
</table>
Table 104-21  (Cont.) DBMS_MGWMSG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVARRAY_GET_SHORT Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the SHORT_VALUE value type</td>
</tr>
<tr>
<td>NVARRAY_GET_TEXT Function</td>
<td>Gets the value of the name-value array element that you specify in p_name and with the TEXT_VALUE value type</td>
</tr>
<tr>
<td>XML_TO_LCR Function</td>
<td>Converts a SYS.XMLTYPE object to a SYS.ANYDATA object encapsulating a row LCR (LCR$_ROW_RECORD) or a DDL LCR (LCR$_DDL_RECORD)</td>
</tr>
</tbody>
</table>

104.4.1 LCR_TO_XML Function

This function converts a SYS.ANYDATA object encapsulating a row LCR (Logical Change Record, in this case a LCR$_ROW_RECORD) or a DDL LCR (LCR$_DDL_RECORD) to a SYS.XMLTYPE object.

See Also:
XML_TO_LCR Function

Syntax

```
DBMS_MGWMSG.LCR_TO_XML (  
  p_anydata IN SYS.ANYDATA  )
RETURN SYS.XMLTYPE;
```

Parameters

Table 104-22  LCR_TO_XML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_anydata</td>
<td>An ANYDATA object to be converted</td>
</tr>
</tbody>
</table>

Return Values

Returns a SYS.XMLTYPE object.

Usage Notes

An exception is raised if the encapsulated type p_anydata is not an LCR.
104.4.2 NVARRAY_ADD Procedure

This procedure appends a name-value element to the end of a name-value array.

**Syntax**

```sql
DBMS_MGWMSG.NVARRAY_ADD (
    p_array  IN OUT SYS.MGW_NAME_VALUE_ARRAY_T,
    p_value  IN     SYS.MGW_NAME_VALUE_T );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>On input, the name-value array instance to modify. If NULL, then a new array is created. On output, the modified name-value array instance.</td>
</tr>
<tr>
<td>p_value</td>
<td>The value to add. If NULL, then p_array is not changed.</td>
</tr>
</tbody>
</table>

104.4.3 NVARRAY_FIND_NAME Function

This function searches a name-value array for the element with the name you specify in p_name.

**Syntax**

```sql
DBMS_MGWMSG.NVARRAY_FIND_NAME (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN BINARY_INTEGER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array to search</td>
</tr>
<tr>
<td>p_name</td>
<td>The name to find</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

**Return Values**

Returns a positive integer that is the array index of the matching element or zero (0) if the specified name is not found.
104.4.4 NVARRAY_FIND_NAME_TYPE Function

This function searches a name-value array for an element with the name and value type you specify.

Syntax

```
DBMS_MGWMSG.NVARRAY_FIND_NAME_TYPE (  
  p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,  
  p_name     IN VARCHAR2,  
  p_type     IN BINARY_INTEGER  
  p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )  
RETURN BINARY_INTEGER;
```

Parameters

Table 104-25  NVARRAY_FIND_NAME_TYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array to search</td>
</tr>
<tr>
<td>p_name</td>
<td>The name to find</td>
</tr>
<tr>
<td>p_type</td>
<td>The value type. Refer to the value type constants in Table 104-1.</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values

Returns a positive integer that is the array index of the matching element, zero (0) if the specified name is not found, or negative one (-1) if the specified name is found but a type mismatch exists.

104.4.5 NVARRAY_GET Function

This function gets the name-value element of the name you specify in p_name from a name-value array.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET (  
  p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,  
  p_name     IN VARCHAR2,  
  p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )  
RETURN SYS.MGW_NAME_VALUE_T;
```

Parameters

Table 104-26  NVARRAY_GET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
</tbody>
</table>
### 104.4.6 NVARRAY_GET_BOOLEAN Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `BOOLEAN_VALUE` value type.

**Syntax**

```sql
DBMS_MGWMSG.NVARRAY_GET_BOOLEAN (  
    p_array IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name IN VARCHAR2,  
    p_compare IN BINARY_INTEGER DEFAULT CASE_SENSITIVE  
) RETURN INTEGER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code>.</td>
</tr>
</tbody>
</table>

**Return Values**

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

### 104.4.7 NVARRAY_GET_BYTE Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `BYTE_VALUE` value type.

**Syntax**

```sql
DBMS_MGWMSG.NVARRAY_GET_BYTE (  
    p_array IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name IN VARCHAR2,  
    p_compare IN BINARY_INTEGER DEFAULT CASE_SENSITIVE  
) RETURN INTEGER;
```
Parameters

Table 104-28  NVARRAY_GET_BYTE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

104.4.8 NVARRAY_GET_DATE Function

This function gets the value of the name-value array element that you specify in p_name and with the DATE_VALUE value type.

Syntax

```sql
DBMS_MGWMSG.NVARRAY_GET_DATE (  
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name     IN VARCHAR2,  
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )  
RETURN DATE;
```

Parameters

Table 104-29  NVARRAY_GET_DATE Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

104.4.9 NVARRAY_GET_DOUBLE Function

This function gets the value of the name-value array element that you specify in p_name and with the DOUBLE_VALUE value type.

Syntax

```sql
DBMS_MGWMSG.NVARRAY_GET_DOUBLE (  
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name     IN VARCHAR2,
```

Chapter 104
Summary of DBMS_MGWMSG Subprograms
p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;

Parameters

Table 104-30  NVARRAY_GET_DOUBLE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

104.4.10 NVARRAY_GET_FLOAT Function

This function gets the value of the name-value array element that you specify in p_name and with the FLOAT_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_FLOAT ( 
  p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
  p_name     IN VARCHAR2,
  p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

Parameters

Table 104-31  NVARRAY_GET_FLOAT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.
104.4.11 NVARRAY_GET_INTEGER Function

This function gets the value of the name-value array element that you specify in p_name and with the INTEGER_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_INTEGER (  
    p_array IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name IN VARCHAR2,  
    p_compare IN BINARY_INTEGER DEFAULT CASE_SENSITIVE  
) RETURN INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

104.4.12 NVARRAY_GET_LONG Function

This function gets the value of the name-value array element that you specify in p_name and with the LONG_VALUE value type.

Syntax

```
DBMS_MGWMSG.NVARRAY_GET_LONG (  
    p_array IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name IN VARCHAR2,  
    p_compare IN BINARY_INTEGER DEFAULT CASE_SENSITIVE  
) RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.
Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

### 104.4.13 NVARRAY_GET_RAW Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `RAW_VALUE` value type.

**Syntax**

```sql
DBMS_MGWMSG.NVARRAY_GET_RAW (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN RAW;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_array</code></td>
<td>The name-value array</td>
</tr>
<tr>
<td><code>p_name</code></td>
<td>The value name</td>
</tr>
<tr>
<td><code>p_compare</code></td>
<td>Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code>.</td>
</tr>
</tbody>
</table>

**Return Values**

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

### 104.4.14 NVARRAY_GET_SHORT Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `SHORT_VALUE` value type.

**Syntax**

```sql
DBMS_MGWMSG.NVARRAY_GET_SHORT (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_array</code></td>
<td>The name-value array</td>
</tr>
</tbody>
</table>

**Table 104-35 NVARRAY_GET_SHORT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_array</code></td>
<td>The name-value array</td>
</tr>
</tbody>
</table>
Table 104-35  (Cont.) NVARRAY_GET_SHORT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values
Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

104.4.15 NVARRAY_GET_TEXT Function

This function gets the value of the name-value array element that you specify in p_name and with the TEXT_VALUE value type.

Syntax

```sql
DBMS_MGWMSG.NVARRAY_GET_TEXT (  
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name     IN VARCHAR2,  
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE  
) RETURN VARCHAR2;
```

Parameters

Table 104-36  NVARRAY_GET_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_array</td>
<td>The name-value array</td>
</tr>
<tr>
<td>p_name</td>
<td>The value name</td>
</tr>
<tr>
<td>p_compare</td>
<td>Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.</td>
</tr>
</tbody>
</table>

Return Values
Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

104.4.16 XML_TO_LCR Function

This function converts a SYS.XMLTYPE object to a SYS.ANYDATA object encapsulating a row LCR (LCSR_ROW_RECORD) or a DDL LCR (LCSR_DDL_RECORD).

See Also:
LCR_TO_XML Function
Syntax

DBMS_MGWMSG.XML_TO_LCR (  
    p_xmldata IN SYS.XMLTYPE  )  
RETURN SYS.ANYDATA;

Parameters

Table 104-37 XML_TO_LCR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_xmldata</td>
<td>An XMLTYPE object representing an LCR</td>
</tr>
</tbody>
</table>

Return Values

Returns a SYS.ANYDATA object.

Usage Notes

An exception is raised if p_xmldata cannot be converted to an LCR.
The **DBMS_MONITOR** package enables you to use PL/SQL for controlling additional tracing and statistics gathering.

The chapter contains the following topics:

- Summary of DBMS_MONITOR Subprograms

### 105.1 Summary of DBMS_MONITOR Subprograms

This table lists the **DBMS_MONITOR** subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_ID_STAT_DISABLE Procedure</td>
<td>Disables statistic gathering previously enabled for a given Client Identifier</td>
</tr>
<tr>
<td>CLIENT_ID_STAT_ENABLE Procedure</td>
<td>Enables statistic gathering for a given Client Identifier</td>
</tr>
<tr>
<td>CLIENT_ID_TRACE_DISABLE Procedure</td>
<td>Disables the trace previously enabled for a given Client Identifier</td>
</tr>
<tr>
<td>CLIENT_ID_TRACE_ENABLE Procedure</td>
<td>Enables the trace for a given Client Identifier globally for the database</td>
</tr>
<tr>
<td>DATABASE_TRACE_DISABLE Procedure</td>
<td>Disables SQL trace for the whole database or a specific instance</td>
</tr>
<tr>
<td>DATABASE_TRACE_ENABLE Procedure</td>
<td>Enables SQL trace for the whole database or a specific instance</td>
</tr>
<tr>
<td>SERV_MOD_ACT_STAT_DISABLE Procedure</td>
<td>Disables statistic gathering enabled for a given combination of Service Name, MODULE and ACTION</td>
</tr>
<tr>
<td>SERV_MOD_ACT_STAT_ENABLE Procedure</td>
<td>Enables statistic gathering for a given combination of Service Name, MODULE and ACTION</td>
</tr>
<tr>
<td>SERV_MOD_ACT_TRACE_DISABLE Procedure</td>
<td>Disables the trace for ALL enabled instances for a or a given combination of Service Name, MODULE and ACTION globally</td>
</tr>
<tr>
<td>SERV_MOD_ACT_TRACE_ENABLE Procedure</td>
<td>Enables SQL tracing for a given combination of Service Name, MODULE and ACTION globally unless an instance_name is specified</td>
</tr>
<tr>
<td>SESSION_TRACE_DISABLE Procedure</td>
<td>Disables the previously enabled trace for a given database session identifier (SID) on the local instance</td>
</tr>
<tr>
<td>SESSION_TRACE_ENABLE Procedure</td>
<td>Enables the trace for a given database session identifier (SID) on the local instance</td>
</tr>
</tbody>
</table>
105.1.1 CLIENT_ID_STAT_DISABLE Procedure

This procedure will disable statistics accumulation for all instances and remove the accumulated results from $CLIENT_STATS view enabled by the CLIENT_ID_STAT_ENABLE Procedure.

Syntax

```sql
DBMS_MONITOR.CLIENT_ID_STAT_DISABLE(
    client_id            IN   VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>Client Identifier for which statistic aggregation is disabled</td>
</tr>
</tbody>
</table>

Examples

To disable accumulation:

```sql
EXECUTE DBMS_MONITOR.CLIENT_ID_STAT_DISABLE('janedoe');
```

105.1.2 CLIENT_ID_STAT_ENABLE Procedure

This procedure enables statistic gathering for a given Client Identifier.

Statistics gathering is global for the database and persistent across instance starts and restarts. That is, statistics are enabled for all instances of the same database, including restarts. Statistics are viewable through $CLIENT_STATS views.

Syntax

```sql
DBMS_MONITOR.CLIENT_ID_STAT_ENABLE(
    client_id            IN   VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>Client Identifier for which statistic aggregation is enabled</td>
</tr>
</tbody>
</table>

Examples

To enable statistic accumulation for a client with a given client ID:

```sql
EXECUTE DBMS_MONITOR.CLIENT_ID_STAT_ENABLE('janedoe');
```
105.1.3 CLIENT_ID_TRACE_DISABLE Procedure

This procedure will disable tracing enabled by the CLIENT_ID_TRACE_ENABLE Procedure.

Syntax

DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE(
    client_id IN VARCHAR2);

Parameters

Table 105-4 CLIENT_ID_TRACE_DISABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>Client Identifier for which SQL tracing is disabled</td>
</tr>
</tbody>
</table>

Examples

EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE ('janedoe');

105.1.4 CLIENT_ID_TRACE_ENABLE Procedure

This procedure will enable the trace for a given client identifier globally for the database.

Syntax

DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE(
    client_id IN VARCHAR2,
    waits IN BOOLEAN DEFAULT TRUE,
    binds IN BOOLEAN DEFAULT FALSE,
    plan_stat IN VARCHAR2 DEFAULT NULL);

Parameters

Table 105-5 CLIENT_ID_TRACE_ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>Database Session Identifier for which SQL tracing is enabled</td>
</tr>
<tr>
<td>waits</td>
<td>If TRUE, wait information is present in the trace</td>
</tr>
<tr>
<td>binds</td>
<td>If TRUE, bind information is present in the trace</td>
</tr>
<tr>
<td>plan_stat</td>
<td>Frequency at which we dump row source statistics. Value should be 'NEVER','FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.</td>
</tr>
</tbody>
</table>

Usage Notes

- The trace will be written to multiple trace files because more than one Oracle shadow process can work on behalf of a given client identifier.
- The tracing is enabled for all instances and persistent across restarts.
Examples

EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE('janedoe', TRUE, FALSE);

105.1.5 DATABASE_TRACE_DISABLE Procedure

This procedure disables SQL trace for the whole database or a specific instance.

Syntax

DBMS_MONITOR.DATABASE_TRACE_DISABLE(
    instance_name  IN VARCHAR2 DEFAULT NULL);

Parameters

Table 105-6  DATABASE_TRACE_DISABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance_name</td>
<td>Disables tracing for the named instance</td>
</tr>
</tbody>
</table>

105.1.6 DATABASE_TRACE_ENABLE Procedure

This procedure enables SQL trace for the whole database or a specific instance.

Syntax

DBMS_MONITOR.DATABASE_TRACE_ENABLE(
    waits          IN BOOLEAN DEFAULT TRUE,
    binds          IN BOOLEAN DEFAULT FALSE,
    instance_name  IN VARCHAR2 DEFAULT NULL,
    plan_stat      IN VARCHAR2 DEFAULT NULL);

Parameters

Table 105-7  DATABASE_TRACE_ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>waits</td>
<td>If TRUE, wait information will be present in the trace</td>
</tr>
<tr>
<td>binds</td>
<td>If TRUE, bind information will be present in the trace</td>
</tr>
<tr>
<td>instance_name</td>
<td>If set, restricts tracing to the named instance</td>
</tr>
<tr>
<td>plan_stat</td>
<td>Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.</td>
</tr>
</tbody>
</table>
105.1.7 SERV_MOD_ACT_STAT_DISABLE Procedure

This procedure will disable statistics accumulation and remove the accumulated results from V$SERV_MOD_ACT_STATS view.

Statistics disabling is persistent for the database. That is, service statistics are disabled for instances of the same database (plus dblinks that have been activated as a result of the enable).

Syntax

DBMS_MONITOR.SERV_MOD_ACT_STAT_DISABLE(
    service_name    IN VARCHAR2,
    module_name     IN VARCHAR2,
    action_name     IN VARCHAR2 DEFAULT ALL_ACTIONS);

Parameters

Table 105-8  SERV_MOD_ACT_STAT_DISABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service for which statistic aggregation is disabled</td>
</tr>
<tr>
<td>module_name</td>
<td>Name of the MODULE. An additional qualifier for the service. It is a</td>
</tr>
<tr>
<td></td>
<td>required parameter.</td>
</tr>
<tr>
<td>action_name</td>
<td>Name of the ACTION. An additional qualifier for the Service and MODULE</td>
</tr>
<tr>
<td></td>
<td>name. Omitting the parameter (or supplying ALL_ACTIONS constant) means</td>
</tr>
<tr>
<td></td>
<td>enabling aggregation for all Actions for a given Service/MODULE combination.</td>
</tr>
<tr>
<td></td>
<td>In this case, statistics are aggregated on the module level.</td>
</tr>
</tbody>
</table>

Usage Notes

Regarding statistics gathering, when you change the module or action, the change takes effect when the next user call is executed in the session. For example, if a module is set to 'module 1' in a session, and the module is reset to 'module 2' in a user call in the session, then the module remains 'module 1' during this user call. The module is changed to 'module 2' in the next user call in the session.

105.1.8 SERV_MOD_ACT_STAT_ENABLE Procedure

This procedure enables statistic gathering for a given combination of Service Name, MODULE and ACTION.

Calling this procedure enables statistic gathering for a hierarchical combination of Service name, MODULE name, and ACTION name on all instances for the same database. Statistics are accessible by means of the V$SERV_MOD_ACT_STATS view.

Syntax

DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE(
    service_name    IN VARCHAR2,
    module_name     IN VARCHAR2,
    action_name     IN VARCHAR2 DEFAULT ALL_ACTIONS);
Parameters

Table 105-9  SERV_MOD_ACT_STAT_ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service for which statistic aggregation is enabled</td>
</tr>
<tr>
<td>module_name</td>
<td>Name of the MODULE. An additional qualifier for the service. It is a required parameter.</td>
</tr>
<tr>
<td>action_name</td>
<td>Name of the ACTION. An additional qualifier for the Service and MODULE name. Omitting the parameter (or supplying ALL_ACTIONS constant) means enabling aggregation for all Actions for a given Service/MODULE combination. In this case, statistics are aggregated on the module level.</td>
</tr>
</tbody>
</table>

Usage Notes

Enabling statistic aggregation for the given combination of Service/Module/Action names is slightly complicated by the fact that the Module/Action values can be empty strings which are indistinguishable from NULLs. For this reason, we adopt the following conventions:

A special constant (unlikely to be a real action names) is defined:

```sql
ALL_ACTIONS constant VARCHAR2 := '###ALL_ACTIONS';
```

Using ALL_ACTIONS for an action specification means that aggregation is enabled for all actions with a given module name, while using NULL (or empty string) means that aggregation is enabled for an action whose name is an empty string.

Regarding statistics gathering, when you change the module or action, the change takes effect when the next user call is executed in the session. For example, if a module is set to 'module 1' in a session, and the module is reset to 'module 2' in a user call in the session, then the module remains 'module 1' during this user call. The module is changed to 'module 2' in the next user call in the session.

Examples

To enable statistic accumulation for a given combination of Service name and MODULE:

```sql
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE( 'APPS1','PAYROLL');
```

To enable statistic accumulation for a given combination of Service name, MODULE and ACTION:

```sql
EXECUTE
DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE('APPS1','GLEDGER','DEBIT_ENTRY');
```

If both of the preceding commands are issued, statistics are accumulated as follows:

- For the APPS1 service, because accumulation for each Service Name is the default.
- For all actions in the PAYROLL Module.
- For the DEBIT_ENTRY Action within the GLEDGER Module.
105.1.9 SERV_MOD_ACT_TRACE_DISABLE Procedure

This procedure will disable the trace at **ALL** enabled instances for a given combination of Service Name, **MODULE**, and **ACTION** name globally.

**Syntax**

```sql
DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE(
    service_name    IN  VARCHAR2,
    module_name     IN  VARCHAR2,
    action_name     IN  VARCHAR2 DEFAULT ALL_ACTIONS,
    instance_name   IN  VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service for which tracing is disabled.</td>
</tr>
<tr>
<td>module_name</td>
<td>Name of the <strong>MODULE</strong>. An additional qualifier for the service.</td>
</tr>
<tr>
<td>action_name</td>
<td>Name of the <strong>ACTION</strong>. An additional qualifier for the Service and <strong>MODULE</strong> name.</td>
</tr>
<tr>
<td>instance_name</td>
<td>If set, this restricts tracing to the named <strong>instance_name</strong>.</td>
</tr>
</tbody>
</table>

**Usage Notes**

Specifying **NULL** for the **module_name** parameter means that statistics will no longer be accumulated for the sessions which do not set the **MODULE** attribute.

**Examples**

To enable tracing for a Service named APPS1:

```sql
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1',
    DBMS_MONITOR.ALL_MODULES,  DBMS_MONITOR.ALL_ACTIONS,TRUE,
    FALSE,NULL);
```

To disable tracing specified in the previous step:

```sql
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE('APPS1');
```

To enable tracing for a given combination of Service and **MODULE** (**all** **ACTIONs**):

```sql
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1','PAYROLL',
    DBMS_MONITOR.ALL_ACTIONS,TRUE,FALSE,NULL);
```

To disable tracing specified in the previous step:

```sql
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE('APPS1','PAYROLL');
```
105.1.10 SERV_MOD_ACT_TRACE_ENABLE Procedure

This procedure will enable SQL tracing for a given combination of Service Name, Module and Action globally unless an instance_name is specified.

Syntax

```sql
DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE(
    service_name    IN VARCHAR2,
    module_name     IN VARCHAR2 DEFAULT ANY_MODULE,
    action_name     IN VARCHAR2 DEFAULT ANY_ACTION,
    waits           IN BOOLEAN DEFAULT TRUE,
    binds           IN BOOLEAN DEFAULT FALSE,
    instance_name   IN VARCHAR2 DEFAULT NULL,
    plan_stat       IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service for which SQL trace is enabled</td>
</tr>
<tr>
<td>module_name</td>
<td>Name of the Module for which SQL trace is enabled. An optional additional qualifier for the service. If omitted, SQL trace is enabled or all modules and actions in a given service.</td>
</tr>
<tr>
<td>action_name</td>
<td>Name of the Action for which SQL trace is enabled. An optional additional qualifier for the Service and Module name. If omitted, SQL trace is enabled for all actions in a given module.</td>
</tr>
<tr>
<td>waits</td>
<td>If TRUE, wait information is present in the trace</td>
</tr>
<tr>
<td>binds</td>
<td>If TRUE, bind information is present in the trace</td>
</tr>
<tr>
<td>instance_name</td>
<td>If set, this restricts tracing to the named instance_name</td>
</tr>
<tr>
<td>plan_stat</td>
<td>Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.</td>
</tr>
</tbody>
</table>

Usage Notes

- The procedure enables a trace for a given combination of Service, Module and Action name. The specification is strictly hierarchical: Service Name or Service Name/Module, or Service Name, Module, and Action name must be specified. Omitting a qualifier behaves like a wild-card, so that not specifying an Action means all Actions. Using the ALL_ACTIONS constant achieves the same purpose.
- This tracing is useful when an application Module and optionally known Action is experiencing poor service levels.
- By default, tracing is enabled globally for the database. The instance_name parameter is provided to restrict tracing to named instances that are known, for example, to exhibit poor service levels.
- Tracing information is present in multiple trace files and you must use the trcsess tool to collect it into a single file.
• Specifying `NULL` for the `module_name` parameter means that statistics will be accumulated for the sessions which do not set the `MODULE` attribute.

**Examples**

To enable tracing for a Service named `APPS1`:

```sql
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1', DBMS_MONITOR.ALL_MODULES, DBMS_MONITOR.ALL_ACTIONS, TRUE, FALSE, NULL);
```

To enable tracing for a given combination of Service and `MODULE` (all `ACTIONS`):

```sql
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1', 'PAYROLL', DBMS_MONITOR.ALL_ACTIONS, TRUE, FALSE, NULL);
```

### 105.1.11 SESSION_TRACE_DISABLE Procedure

This procedure will disable the trace for a given database session at the local instance.

**Syntax**

```sql
DBMS_MONITOR.SESSION_TRACE_DISABLE(
    session_id IN BINARY_INTEGER DEFAULT NULL,
    serial_num IN BINARY_INTEGER DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>Database Session Identifier for which SQL trace is disabled</td>
</tr>
<tr>
<td>serial_num</td>
<td>Serial number for this session</td>
</tr>
</tbody>
</table>

**Usage Notes**

If `serial_num` is `NULL` but `session_id` is specified, a session with a given `session_id` is no longer traced irrespective of its serial number. If both `session_id` and `serial_num` are `NULL`, the current user session is no longer traced. It is illegal to specify `NULL` `session_id` and non-`NULL` `serial_num`. In addition, the `NULL` values are default and can be omitted.

**Examples**

To enable tracing for a client with a given client session ID:

```sql
EXECUTE DBMS_MONITOR.SESSION.Trace_ENABLE(7,4634, TRUE, FALSE);
```

To disable tracing specified in the previous step:

```sql
EXECUTE DBMS_MONITOR.SESSION_TRACE_DISABLE(7,4634);
```
105.1.12 SESSION_TRACE_ENABLE Procedure

This procedure enables a SQL trace for the given Session ID on the local instance.

Syntax

```sql
DBMS_MONITOR.SESSION_TRACE_ENABLE(
    session_id   IN  BINARY_INTEGER DEFAULT NULL,
    serial_num   IN  BINARY_INTEGER DEFAULT NULL,
    waits        IN  BOOLEAN DEFAULT TRUE,
    binds        IN  BOOLEAN DEFAULT FALSE,
    plan_stat    IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>Client Identifier for which SQL trace is enabled. If omitted (or NULL), the user's own session is assumed.</td>
</tr>
<tr>
<td>serial_num</td>
<td>Serial number for this session. If omitted (or NULL), only the session ID is used to determine a session.</td>
</tr>
<tr>
<td>waits</td>
<td>If TRUE, wait information is present in the trace</td>
</tr>
<tr>
<td>binds</td>
<td>If TRUE, bind information is present in the trace</td>
</tr>
<tr>
<td>plan_stat</td>
<td>Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.</td>
</tr>
</tbody>
</table>

Usage Notes

The procedure enables a trace for a given database session, and is still useful for client/server applications. The trace is enabled only on the instance to which the caller is connected, since database sessions do not span instances. This tracing is strictly local to an instance.

If `serial_num` is NULL but `session_id` is specified, a session with a given `session_id` is traced irrespective of its serial number. If both `session_id` and `serial_num` are NULL, the current user session is traced. It is illegal to specify NULL `session_id` and non-NULL `serial_num`. In addition, the NULL values are default and can be omitted.

Examples

To enable tracing for a client with a given client session ID:

```sql
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(7, 4634, TRUE, FALSE);
```

To disable tracing specified in the previous step:

```sql
EXECUTE DBMS_MONITOR.SESSION_TRACE_DISABLE(7, 4634);
```

Either

```sql
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(5);
```
or
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(5, NULL);
traces the session with session ID of 5, while either
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE();
or
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(NULL, NULL);
traces the current user session. Also,
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(NULL, NULL, TRUE, TRUE);
traces the current user session including waits and binds. The same can be also expressed using keyword syntax:
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(binds=>TRUE);
106

DBMS_MVIEW

DBMS_MVIEW enables you to understand capabilities for materialized views and potential materialized views, including their rewrite availability. It also enables you to refresh materialized views that are not part of the same refresh group and purge logs.

Note:

DBMS_MVIEW is a synonym for DBMS_SNAPSHOT.

See Also:

Oracle Database Data Warehousing Guide for more information about using materialized views in a data warehousing environment

This chapter contains the following topics:

- Operational Notes
- Security Model
- Rules and Limits
- Summary of DBMS_MVIEW Subprograms

106.1 DBMS_MVIEW Operational Notes

If a query is less than 256 characters long, you can invoke EXPLAIN_REWRITE using the EXECUTE command from SQL*Plus. Otherwise, the recommended method is to use a PL/SQL BEGIN..END block, as shown in the examples in /rdbms/demo/smxrw.sql.

106.2 DBMS_MVIEW Security Model

The DBMS_MVIEW package consists of a number of materialized view-related subprograms, each of which has different functionality and privilege requirements.

The privilege model is generally based on the invoker's right. Each package subprogram is executed by first checking the privileges against the invoker. If all the required privileges are met, the subprogram will be executed. Otherwise, an insufficient privileges error will be thrown.
106.3 DBMS_MVIEW Rules and Limits

The DBMS_MVIEW.EXPLAIN_REWRITE procedure cannot accept queries longer than 32627 characters. These restrictions also apply when passing the defining query of a materialized view to the DBMS_MVIEW.EXPLAIN_MVIEW procedure.

106.4 Summary of DBMS_MVIEW Subprograms

This table lists the DBMS_MVIEW subprograms and briefly describes them.

Table 106-1   DBMS_MVIEW Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_TABLE_REORGANIZATION Procedure</td>
<td>Performs a process to preserve materialized view data needed for refresh</td>
</tr>
<tr>
<td>END_TABLE_REORGANIZATION Procedure</td>
<td>Ensures that the materialized view data for the master table is valid and that the master table is in the proper state</td>
</tr>
<tr>
<td>ESTIMATE_MVIEW_SIZE Procedure</td>
<td>Estimates the size of a materialized view that you might create, in bytes and rows</td>
</tr>
<tr>
<td>EXPLAIN_MVIEW Procedure</td>
<td>Explains what is possible with a materialized view or potential materialized view</td>
</tr>
<tr>
<td>EXPLAIN_REWRITE Procedure</td>
<td>Explains why a query failed to rewrite or why the optimizer chose to rewrite a query with a particular materialized view or materialized views</td>
</tr>
<tr>
<td>I_AM_A_REFRESH Function</td>
<td>Returns the value of the I_AM_REFRESH package state</td>
</tr>
<tr>
<td>PMARKER Function</td>
<td>Returns a partition marker from a rowid, and is used for Partition Change Tracking (PCT)</td>
</tr>
<tr>
<td>PURGE_DIRECT_LOAD_LOG Procedure</td>
<td>Purges rows from the direct loader log after they are no longer needed by any materialized views (used with data warehousing)</td>
</tr>
<tr>
<td>PURGE_LOG Procedure</td>
<td>Purges rows from the materialized view log</td>
</tr>
<tr>
<td>PURGE_MVIEW_FROM_LOG Procedure</td>
<td>Purges rows from the materialized view log</td>
</tr>
<tr>
<td>REFRESH Procedures</td>
<td>Refreshes one or more materialized views that are not members of the same refresh group</td>
</tr>
<tr>
<td>REFRESH_ALL_MVIEWS Procedure</td>
<td>Refreshes all materialized views that do not reflect changes to their master table or master materialized view</td>
</tr>
<tr>
<td>REFRESH_DEPENDENT Procedures</td>
<td>Refreshes all table-based materialized views that depend on a specified master table or master materialized view, or list of master tables or master materialized views</td>
</tr>
<tr>
<td>REGISTER_MVIEW Procedure</td>
<td>Enables the administration of individual materialized views</td>
</tr>
<tr>
<td>UNREGISTER_MVIEW Procedure</td>
<td>Enables the administration of individual materialized views once invoked at a master site or master materialized view site to unregister a materialized view</td>
</tr>
</tbody>
</table>
106.4.1 BEGIN_TABLE_REORGANIZATION Procedure

This procedure performs a process to preserve materialized view data needed for refresh. It must be called before a master table is reorganized.

Syntax

```sql
DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION (
    tabowner  IN   VARCHAR2,
    tabname   IN   VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabowner</td>
<td>Owner of the table being reorganized</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table being reorganized</td>
</tr>
</tbody>
</table>

106.4.2 END_TABLE_REORGANIZATION Procedure

This procedure ensures that the materialized view data for the master table is valid and that the master table is in the proper state. It must be called after a master table is reorganized.

Syntax

```sql
DBMS_MVIEW.END_TABLE_REORGANIZATION (
    tabowner  IN   VARCHAR2,
    tabname   IN   VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabowner</td>
<td>Owner of the table being reorganized</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table being reorganized</td>
</tr>
</tbody>
</table>

106.4.3 ESTIMATE_MVIEW_SIZE Procedure

This procedure estimates the size of a materialized view that you might create, in bytes and number of rows.

Syntax

```sql
DBMS_MVIEW.ESTIMATE_MVIEW_SIZE (  
    stmt_id IN VARCHAR2,    
    select_clause IN VARCHAR2,    
    num_rows OUT NUMBER,    
    num_bytes OUT NUMBER);  
```
106.4.4 EXPLAIN_MVIEW Procedure

This procedure enables you to learn what is possible with a materialized view or potential materialized view. For example, you can determine if a materialized view is fast refreshable and what types of query rewrite you can perform with a particular materialized view.

Using this procedure is straightforward. You simply call DBMS_MVIEW.EXPLAIN_MVIEW, passing in as parameters the schema and materialized view name for an existing materialized view. Alternatively, you can specify the SELECT string or CREATE MATERIALIZED VIEW statement for a potential materialized view. The materialized view or potential materialized view is then analyzed and the results are written into either a table called MV_CAPABILITIES_TABLE, which is the default, or to an array called MSG_ARRAY.

The procedure is overloaded:

- The first version is for explaining an existing or potential materialized view with output to MV_CAPABILITIES_TABLE.
- The second version is for explaining an existing or potential materialized view with output to a VARRAY.

Syntax

DBMS_MVIEW.EXPLAIN_MVIEW (  
  mv IN VARCHAR2,  
  statement_id IN VARCHAR2:= NULL);

DBMS_MVIEW.EXPLAIN_MVIEW (  
  mv IN VARCHAR2,  
  msg_array OUT SYS.ExplainMVArrayType);

Parameters

Table 106-5  EXPLAIN_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv</td>
<td>The name of an existing materialized view (optionally qualified with the owner name separated by a &quot;.&quot;) or a SELECT statement or a CREATE MATERIALIZED VIEW statement for a potential materialized view.</td>
</tr>
</tbody>
</table>
Table 106-5 (Cont.) EXPLAIN_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement_id</td>
<td>A client-supplied unique identifier to associate output rows with specific invocations of EXPLAIN_MVIEW</td>
</tr>
<tr>
<td>msg_array</td>
<td>The PL/SQL VARRAY that receives the output. Use this parameter to direct EXPLAIN_MVIEW’s output to a PL/SQL VARRAY rather than MV_CAPABILITIES_TABLE.</td>
</tr>
</tbody>
</table>

Usage Notes

You must run the utlxmv.sql script to create MV_CAPABILITIES_TABLE in the current schema prior to calling EXPLAIN_MVIEW except when you direct output to a VARRAY. The script is found in the ADMIN directory.

106.4.5 EXPLAIN_REWRITE Procedure

This procedure enables you to learn why a query failed to rewrite, or, if it rewrites, which materialized views will be used.

Using the results from the procedure, you can take the appropriate action needed to make a query rewrite if at all possible. The query specified in the EXPLAIN_REWRITE statement is never actually executed.

A demo file, xrwutl.sql, is available to help format the output from EXPLAIN_REWRITE.

Syntax

You can obtain the output from DBMS_MVIEW.EXPLAIN_REWRITE in two ways. The first is to use a table, while the second is to create a VARRAY. The following shows the basic syntax for using an output table:

```sql
DBMS_MVIEW.EXPLAIN_REWRITE (query VARCHAR2, mv VARCHAR2(30), statement_id VARCHAR2(30));
```

You can create an output table called REWRITE_TABLE by executing the utlxrw.sql script.

The query parameter is a text string representing the SQL query. The parameter, mv, is a fully qualified materialized view name in the form of schema.mv. This is an optional parameter. When it is not specified, EXPLAIN_REWRITE returns any relevant messages regarding all the materialized views considered for rewriting the given query. When schema is omitted and only mv is specified, EXPLAIN_REWRITE looks for the materialized view in the current schema.

If you want to direct the output of EXPLAIN_REWRITE to a VARRAY instead of a table, you should call the procedure as follows:

```sql
DBMS_MVIEW.EXPLAIN_REWRITE (query [VARCHAR2 | CLOB], mv VARCHAR2(30), output_array SYS.RewriteArrayType);
```
Note that if the query is less than 256 characters long, EXPLAIN_REWRITE can be easily invoked with the execute command from SQL*Plus. Otherwise, the recommended method is to use a PL/SQL BEGIN... END block, as shown in the examples in /rdbms/demo/smxrw*.

You can also use EXPLAIN_REWRITE with multiple materialized views, in which case the syntax will be the same as with a single materialized view, except that the materialized views are specified by a comma-delimited string. For example, to find out whether a given set of materialized views mv1, mv2, and mv3 could be used to rewrite the query, query_txt, and, if not, why not, use EXPLAIN_REWRITE as follows:

DBMS_MVIEW.EXPLAIN_REWRITE(query_txt, 'mv1, mv2, mv3')

See Oracle Database Data Warehousing Guide for more information on using the EXPLAIN_REWRITE procedure.

Parameters

Table 106-6  EXPLAIN_REWRITE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>SQL SELECT statement to be explained</td>
</tr>
<tr>
<td>mv</td>
<td>The fully qualified name of an existing materialized view in the form of SCHEMA.MV. For multiple materialized views, you can provide a comma-delimited list of names.</td>
</tr>
<tr>
<td>statement_id</td>
<td>A client-supplied unique identifier to distinguish output messages</td>
</tr>
<tr>
<td>msg_array</td>
<td>The PL/SQL VARRAY that receives the output. Use this parameter to direct EXPLAIN_REWRITE's output to a PL/SQL VARRAY.</td>
</tr>
</tbody>
</table>

Usage Notes

To obtain the output into a table, you must run the utlxrw.sql script before calling EXPLAIN_REWRITE. This script creates a table named REWRITE_TABLE in the current schema.

106.4.6 I_AM_A_REFRESH Function

This function returns the value of the I_AM_REFRESH package state.

Syntax

```
DBMS_MVIEW.I_AM_A_REFRESH
RETURN BOOLEAN;
```

Return Values

A return value of true indicates that all local replication triggers for materialized views are effectively disabled in this session because each replication trigger first checks this state. A return value of false indicates that these triggers are enabled.
106.4.7 PMARKER Function

This function returns a partition marker from a rowid. It is used for Partition Change Tracking (PCT).

Syntax

```sql
DBMS_MVIEW.PMARKER(
    rid IN ROWID)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rid</td>
<td>The rowid of a row entry in a master table</td>
</tr>
</tbody>
</table>

106.4.8 PURGE_DIRECT_LOAD_LOG Procedure

This procedure removes entries from the direct loader log after they are no longer needed for any known materialized view. This procedure usually is used in environments using Oracle's data warehousing technology.

Syntax

```sql
DBMS_MVIEW.PURGE_DIRECT_LOAD_LOG();
```

106.4.9 PURGE_LOG Procedure

This procedure purges rows from the materialized view log.

Syntax

```sql
DBMS_MVIEW.PURGE_LOG (
    master IN VARCHAR2,
    num IN BINARY_INTEGER := 1,
    flag IN VARCHAR2 := 'NOP');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>Name of the master table or master materialized view.</td>
</tr>
</tbody>
</table>
Table 106-8  (Cont.) PURGE_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>Number of least recently refreshed materialized views whose rows you want to remove from materialized view log. For example, the following statement deletes rows needed to refresh the two least recently refreshed materialized views: DBMS_MVIEW.PURGE_LOG('master_table', 2); To delete all rows in the materialized view log, indicate a high number of materialized views to disregard, as in this example: DBMS_MVIEW.PURGE_LOG('master_table',9999); This statement completely purges the materialized view log that corresponds to master_table if fewer than 9999 materialized views are based on master_table. A simple materialized view whose rows have been purged from the materialized view log must be completely refreshed the next time it is refreshed.</td>
</tr>
<tr>
<td>flag</td>
<td>Specify delete to guarantee that rows are deleted from the materialized view log for at least one materialized view. This parameter can override the setting for the parameter num. For example, the following statement deletes rows from the materialized view log that has dependency rows in the least recently refreshed materialized view: DBMS_MVIEW.PURGE_LOG('master_table',1,'delete');</td>
</tr>
</tbody>
</table>

106.4.10 PURGE_MVIEW_FROM_LOG Procedure

This procedure is called on the master site or master materialized view site to delete the rows in materialized view refresh related data dictionary tables maintained at the master for the specified materialized view identified by mview_id or the combination of mviewowner, mviewname, and mviewsite.

If the materialized view specified is the oldest materialized view to have refreshed from any of the master tables or master materialized views, then the materialized view log is also purged. This procedure does not unregister the materialized view.

Syntax

DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (mview_id IN BINARY_INTEGER);

DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (mviewowner IN VARCHAR2,
mviewname IN VARCHAR2,
mviewsite IN VARCHAR2);

Note:

This procedure is overloaded. The parameter mview_id is mutually exclusive with the three remaining parameters: mviewowner, mviewname, and mviewsite.
Parameters

Table 106-9   PURGE_MVIEW_FROM_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mview_id</td>
<td>If you want to execute this procedure based on the identification of the target materialized view, specify the materialized view identification using the mview_id parameter. Query the DBA_BASE_TABLE_MVIEWS view at the materialized view log site for a listing of materialized view IDs. Executing this procedure based on the materialized view identification is useful if the target materialized view is not listed in the list of registered materialized views (DBA_REGISTERED_MVIEWS).</td>
</tr>
<tr>
<td>mviewowner</td>
<td>If you do not specify an mview_id, enter the owner of the target materialized view using the mviewowner parameter. Query the DBA_REGISTERED_MVIEWS view at the materialized view log site to view the materialized view owners.</td>
</tr>
<tr>
<td>mviewname</td>
<td>If you do not specify an mview_id, enter the name of the target materialized view using the mviewname parameter. Query the DBA_REGISTERED_MVIEWS view at the materialized view log site to view the materialized view names.</td>
</tr>
<tr>
<td>mviewsite</td>
<td>If you do not specify an mview_id, enter the site of the target materialized view using the mviewsite parameter. Query the DBA_REGISTERED_MVIEWS view at the materialized view log site to view the materialized view sites.</td>
</tr>
</tbody>
</table>

Usage Notes

If there is an error while purging one of the materialized view logs, the successful purge operations of the previous materialized view logs are not rolled back. This is to minimize the size of the materialized view logs. In case of an error, this procedure can be invoked again until all the materialized view logs are purged.

106.4.11 REFRESH Procedures

This procedure refreshes a list of materialized views.

Syntax

DBMS_MVIEW.REFRESH ( |
    { list IN VARCHAR2,|
    | tab IN DBMS_UTILITY.UNCL_ARRAY,}|
    method IN VARCHAR2 := NULL,|
    rollback_seg IN VARCHAR2 := NULL,|
    push_deferred_rpc IN BOOLEAN := true,|
    refresh_after_errors IN BOOLEAN := false,|
    purge_option IN BINARY_INTEGER := 1,|
    parallelism IN BINARY_INTEGER := 0,|
    heap_size IN BINARY_INTEGER := 0,|
    atomic_refresh IN BOOLEAN := true,|
    nested IN BOOLEAN := false,|
    out_of_place IN BOOLEAN := false); |
**Note:**

This procedure is overloaded. The list and tab parameters are mutually exclusive.

## Parameters

### Table 106-10   REFRESH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Comma-delimited list of materialized views that you want to refresh. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your local database. Alternatively, you may pass in a PL/SQL index-by table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a materialized view.</td>
</tr>
</tbody>
</table>
| method | A string of refresh methods indicating how to refresh the listed materialized views. An f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. P or p refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables. If a materialized view does not have a corresponding refresh method (that is, if more materialized views are specified than refresh methods), then that materialized view is refreshed according to its default refresh method. For example, consider the following EXECUTE statement within SQL*Plus:
```
DBMS_MVIEW.REFRESH
   ('countries_mv,regions_mv,hr.employees_mv','cf');
```
This statement performs a complete refresh of the countries_mv materialized view, a fast refresh of the regions_mv materialized view, and a default refresh of the hr.employees materialized view. |
| rollback_seg | Name of the materialized view site rollback segment to use while refreshing materialized views |
| push_deferred_rpc | Used by updatable materialized views only. Set this parameter to true if you want to push changes from the materialized view to its associated master tables or master materialized views before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost. |
| refresh_after_errors | If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view. |
Table 106-10  (Cont.) REFRESH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>purge_option</td>
<td>If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 0 means do not purge, 1 means lazy purge, and 2 means aggressive purge. In most cases, lazy purge is the optimal setting. Set purge to aggressive to trim the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set this parameter to 0 and occasionally execute PUSH with this parameter set to 2 to reduce the queue.</td>
</tr>
<tr>
<td>parallelism</td>
<td>0 specifies serial propagation.</td>
</tr>
<tr>
<td></td>
<td>1 specifies parallel propagation using only one parallel process.</td>
</tr>
<tr>
<td>heap_size</td>
<td>Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Note: Do not set this parameter unless directed to do so by Oracle Support Services.</td>
</tr>
<tr>
<td>atomic_refresh</td>
<td>If this parameter is set to true, then the list of materialized views is refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the materialized views is refreshed non-atomically in separate transactions. As part of complete refresh, if truncate is used (non-atomic refresh), unique index rebuild is executed. INDEX REBUILD automatically computes statistics. Thus, statistics are updated for truncated tables.</td>
</tr>
<tr>
<td>nested</td>
<td>If true, then perform nested refresh operations for the specified set of materialized views. Nested refresh operations refresh all the depending materialized views and the specified set of materialized views based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.</td>
</tr>
<tr>
<td>out_of_place</td>
<td>If true, then it performs an out-of-place refresh. The default is false. This parameter uses the four methods of refresh (F, P, C, ?). So, for example, if you specify F and out_of_place = true, then an out-of-place fast refresh will be attempted. Similarly, if you specify P and out_of_place = true, then out-of-place PCT refresh will be attempted.</td>
</tr>
</tbody>
</table>

106.4.12 REFRESH_ALL_MVIEWS Procedure

This procedure refreshes all materialized views that have certain properties

All materialized views with the following properties are refreshed:

- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view DBA_MVIEWS.
This procedure is intended for use with data warehouses.

Syntax

```
DBMS_MVIEW.REFRESH_ALL_MVIEWS (
    number_of_failures OUT BINARY_INTEGER,
    method            IN   VARCHAR2       := NULL,
    rollback_seg      IN   VARCHAR2       := NULL,
    refresh_after_errors IN   BOOLEAN      := false,
    atomic_refresh    IN   BOOLEAN      := true,
    out_of_place      IN   BOOLEAN      := false);
```

Parameters

Table 106-11  REFRESH_ALL_MVIEWS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number_of_failures</td>
<td>Returns the number of failures that occurred during processing</td>
</tr>
<tr>
<td>method</td>
<td>A single refresh method indicating the type of refresh to perform for each materialized view that is refreshed. F or f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. If no method is specified, a materialized view is refreshed according to its default refresh method. F or p refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables.</td>
</tr>
<tr>
<td>rollback_seg</td>
<td>Name of the materialized view site rollback segment to use while refreshing materialized views</td>
</tr>
<tr>
<td>refresh_after_errors</td>
<td>If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view’s master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.</td>
</tr>
<tr>
<td>atomic_refresh</td>
<td>If this parameter is set to true, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the materialized views is refreshed non-atomically in separate transactions.</td>
</tr>
<tr>
<td>out_of_place</td>
<td>If true, then it performs an out-of-place refresh. The default is false. This parameter uses the four methods of refresh (F, P, C, ?). So, for example, if you specify P and out_of_place = true, then an out-of-place fast refresh will be attempted. Similarly, if you specify P and out_of_place = true, then out-of-place PCT refresh will be attempted.</td>
</tr>
</tbody>
</table>

106.4.13 REFRESH_DEPENDENT Procedures

This procedure refreshes all materialized views that have certain properties.

Materialized views with the following properties are refreshed:
• The materialized view depends on a master table or master materialized view in the list of specified masters.
• The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
• The materialized view and all of the master tables or master materialized views on which it depends are local.
• The materialized view is in the view DBA_MVIEWS.

This procedure is intended for use with data warehouses.

**Syntax**

```sql
DBMS_MVIEW.REFRESH_DEPENDENT ( 
    number_of_failures OUT    BINARY_INTEGER,
    { list | tab } IN     VARCHAR2,
    method IN     VARCHAR2    := NULL,
    rollback_seg IN     VARCHAR2    := NULL,
    refresh_after_errors IN     BOOLEAN     := false,
    atomic_refresh IN     BOOLEAN     := true,
    nested IN     BOOLEAN     := false,
    out_of_place IN     BOOLEAN     := false);
```

**Note:**

This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

**Parameters**

**Table 106-12  REFRESH_DEPENDENT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number_of_failures</td>
<td>Returns the number of failures that occurred during processing</td>
</tr>
<tr>
<td>list</td>
<td>tab</td>
</tr>
</tbody>
</table>
Table 106-12  (Cont.) REFRESH_DEPENDENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| method          | A string of refresh methods indicating how to refresh the dependent materialized views. All of the materialized views that depend on a particular table are refreshed according to the refresh method associated with that table. F or f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. F or P refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables. If a table does not have a corresponding refresh method (that is, if more tables are specified than refresh methods), then any materialized view that depends on that table is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus:  

```
DBMS_MVIEW.REFRESH_DEPENDENT
('employees, departments, hr.regions', 'cf');
```

performs a complete refresh of the materialized views that depend on the `employees` table, a fast refresh of the materialized views that depend on the `departments` table, and a default refresh of the materialized views that depend on the `hr.regions` table. |
| rollback_seg    | Name of the materialized view site rollback segment to use while refreshing materialized views                                                                                      |
| refresh_after_errors | If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view. |
| atomic_refresh  | If this parameter is set to true, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to false, then each of the materialized views is refreshed non-atomically in separate transactions. |
| nested          | If true, then perform nested refresh operations for the specified set of tables. Nested refresh operations refresh all the depending materialized views of the specified set of tables based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables. |
| out_of_place    | If true, then it performs an out-of-place refresh. The default is false. This parameter uses the four methods of refresh (F, P, C, ?). So, for example, if you specify F and out_of_place = true, then an out-of-place fast refresh will be attempted. Similarly, if you specify P and out_of_place = true, then out-of-place PCT refresh will be attempted. |
### 106.4.14 REGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to register a materialized view.

Note that, typically, a materialized view is registered automatically during materialized view creation. You should only run this procedure to manually register a materialized view if the automatic registration failed or if the registration information was deleted.

#### Syntax

```sql
DBMS_MVIEW.REGISTER_MVIEW (  
mviewowner  IN   VARCHAR2,  
mviewname   IN   VARCHAR2,  
mviewsite   IN   VARCHAR2,  
mview_id    IN   DATE | BINARY_INTEGER,  
flag        IN   BINARY_INTEGER,  
qry_txt     IN   VARCHAR2,  
rep_type    IN   BINARY_INTEGER := DBMS_MVIEW.REG_UNKNOWN);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mviewowner</td>
<td>Owner of the materialized view.</td>
</tr>
<tr>
<td>mviewname</td>
<td>Name of the materialized view.</td>
</tr>
<tr>
<td>mviewsite</td>
<td>Name of the materialized view site for a materialized view registering at an Oracle database version 8.x and higher master site or master materialized view site. This name should not contain any double quotes.</td>
</tr>
<tr>
<td>mview_id</td>
<td>The identification number of the materialized view. Specify an Oracle database version 8.x and higher materialized view as a <code>BINARY_INTEGER</code>. Specify an Oracle database version 7 materialized view registering at an Oracle database version 8.x and higher master sites or master materialized view sites as a <code>DATE</code>.</td>
</tr>
<tr>
<td>flag</td>
<td>A constant that describes the properties of the materialized view being registered. Valid constants that can be assigned include the following: <code>DBMS_MVIEW.REG_ROWID_MVIEW</code> for a rowid materialized view, <code>DBMS_MVIEW.REG_PRIMARY_KEY_MVIEW</code> for a primary key materialized view, <code>DBMS_MVIEW.REG_OBJECT_ID_MVIEW</code> for an object id materialized view, <code>DBMS_MVIEW.REG_FAST_REFRESHABLE_MVIEW</code> for a materialized view that can be fast refreshed, <code>DBMS_MVIEW.REG_UPDATABLE_MVIEW</code> for a materialized view that is updatable. A materialized view can have more than one of these properties. In this case, use the plus sign (+) to specify more than one property. For example, if a primary key materialized view can be fast refreshed, you can enter the following for this parameter: <code>DBMS_MVIEW.REG_PRIMARY_KEY_MVIEW + DBMS_MVIEW.REG_FAST_REFRESHABLE_MVIEW</code>. You can determine the properties of a materialized view by querying the <code>ALL_MVIEWS</code> data dictionary view.</td>
</tr>
<tr>
<td>qry_txt</td>
<td>The first 32,000 bytes of the materialized view definition query.</td>
</tr>
</tbody>
</table>
Table 106-13  (Cont.) REGISTER_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rep_type</td>
<td>Version of the materialized view. Valid constants that can be assigned include the following:</td>
</tr>
<tr>
<td></td>
<td>DBMS_MVIEW.REG_V7_SNAPSHOT if the materialized view is at an Oracle database version 7 site</td>
</tr>
<tr>
<td></td>
<td>DBMS_MVIEW.REG_V8_SNAPSHOT if the materialized view is at an Oracle database version 8.x or higher site</td>
</tr>
<tr>
<td>reg_repapi_snapshot</td>
<td>if the materialized view is at an Oracle database version 8.x or higher site</td>
</tr>
<tr>
<td></td>
<td>DBMS_MVIEW.REG_UNKNOWN (the default) if you do not know whether the materialized view is at an Oracle database version 7 site or an Oracle database version 8.x (or higher) site</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is invoked at the master site or master materialized view site by a remote materialized view site using a remote procedure call. If REGISTER_MVIEW is called multiple times with the same mviewowner, mviewname, and mviewsite, then the most recent values for mview_id, flag, and qry_txt are stored. If a query exceeds the maximum VARCHAR2 size, then qry_txt contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the value of mview_id must be looked up in the materialized view data dictionary views by the person who calls the procedure.

106.4.15 UNREGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to unregister a materialized view.

Syntax

```
DBMS_MVIEW.UNREGISTER_MVIEW (
    mviewowner IN VARCHAR2,
    mviewname IN VARCHAR2,
    mviewsite IN VARCHAR2);
```

Parameters

Table 106-14  UNREGISTER_MVIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mviewowner</td>
<td>Owner of the materialized view</td>
</tr>
<tr>
<td>mviewname</td>
<td>Name of the materialized view</td>
</tr>
<tr>
<td>mviewsite</td>
<td>Name of the materialized view site</td>
</tr>
</tbody>
</table>
DBMS_MVIEW_STATS package provides an interface to manage the collection and retention of statistics for materialized view refresh operations.

See Also:

Oracle Database Data Warehousing Guide for information about managing and using materialized view refresh statistics

This chapter contains the following topics:

• DBMS_MVIEW_STATS Overview
• DBMS_MVIEW_STATS Security Model
• Summary of DBMS_MVIEW_STATS Subprograms

107.1 DBMS_MVIEW_STATS Overview

You can use the procedures contained in the DBMS_MVIEW_STATS package to manage the collection and retention of statistics for materialized view refresh operations. This includes the level and granularity at which these statistics are collected and the duration for which they are retained in the database.

You can also set database level system defaults for the parameters that control statistics collection.

107.2 DBMS_MVIEW_STATS Security Model

Refer to the Usage Notes section in each subprogram for information about the privileges required to use the subprogram.

107.3 Summary of DBMS_MVIEW_STATS Subprograms

This table lists the DBMS_MVIEW_STATS subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PURGE_REFRESH_STATS Procedure</td>
<td>Purges the statistics of materialized view refresh operations that are older than the specified retention period.</td>
</tr>
</tbody>
</table>
Table 107-1  (Cont.) DBMS_MVIEW_STATS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_MVREF_STATS_PARAMS Procedure</td>
<td>Sets the values of parameters that define the collection level and retention period for materialized view refresh statistics. You can set the values either at the database level or for individual materialized views.</td>
</tr>
<tr>
<td>SET_SYSTEM_DEFAULT Procedure</td>
<td>Sets the system default value of a refresh statistics parameter. The two refresh statistics parameters are collection level and the retention period.</td>
</tr>
</tbody>
</table>

107.3.1 PURGE_REFRESH_STATS Procedure

This procedure purges refresh statistics that are older than the specified retention period for the specified materialized views.

This procedure forces a purge of refresh statistics without altering the retention period defined for the specified materialized views.

Syntax

```sql
DBMS_MVIEW_STATS.PURGE_REFRESH_STATISTICS (  
  mv_list               IN     VARACHAR2,
  retention_period      IN     NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv_list</td>
<td>The fully-qualified name of an existing materialized view in the form of <code>schema_name.mv_name</code>. Use a comma-separated list to specify multiple materialized views. Specify NULL to purge materialized view refresh statistics for all materialized views in the database.</td>
</tr>
<tr>
<td>retention_period</td>
<td>The number of days for which refresh statistics must be preserved in the data dictionary. Statistics for materialized view refresh operations that are older than the retention period are purged from the data dictionary. The retention period specified in this procedure overrides the retention period that may have been set previously either at the database level or for specified materialized views. Specify NULL to use the purging policy defined by the automatic statistics purge. Specify -1 to purge all refresh statistics.</td>
</tr>
</tbody>
</table>

Usage Notes

To invoke this procedure, you need either the SYSDBA privilege or privileges on every materialized view that is specified in `mv_list`. 
107.3.2 SET_MVREF_STATS_PARAMS Procedure

This procedure sets the collection level and retention period for materialized view refresh statistics. You can set these properties either for individual materialized views or for all materialized views in the database.

Syntax

```sql
DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMS (  
  mv_list            IN    VARACHAR2,  
  collection_level   IN    VARCHAR2 DEFAULT NULL,  
  retention_period   IN    NUMBER DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv_list</td>
<td>The fully-qualified name of an existing materialized view in the form of schema_name.mv_name. Use a comma-separated list to specify multiple materialized views. Specify NULL to set properties for all existing materialized views in the database.</td>
</tr>
<tr>
<td>collection_level</td>
<td>Specifies the level of detail used when collecting refresh statistics for the materialized views specified in mv_list. Set one of the following values for collection_level:</td>
</tr>
<tr>
<td></td>
<td>- NONE: No materialized view refresh statistics are collected.</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL: Only basic refresh statistics are collected and stored for the materialized views specified in mv_list.</td>
</tr>
<tr>
<td></td>
<td>- ADVANCED: Detailed refresh statistics are collected and stored for materialized view specified in mv_list.</td>
</tr>
<tr>
<td></td>
<td>If this parameter is set to NULL, then the system default value for collection_level (set using SET_SYSTEM_DEFAULT) is used.</td>
</tr>
<tr>
<td>retention_period</td>
<td>Specifies the retention period, in days, for the refresh statistics of the materialized views specified in mv_list. Statistics that are older than the retention period are automatically purged from the data dictionary.</td>
</tr>
<tr>
<td></td>
<td>Valid values are between 1 and 1365000.</td>
</tr>
<tr>
<td></td>
<td>If this parameter is set to NULL, then the system default value for retention_period (set using SET_SYSTEM_DEFAULT) is used.</td>
</tr>
<tr>
<td></td>
<td>Set retention_period to -1 to specify that refresh statistics for the materialized views in mv_list must never be purged.</td>
</tr>
</tbody>
</table>

Usage Notes

To set the collection level or retention period of one or more materialized views, you must have privileges on those materialized views. To set the collection level or retention period for all materialized views in the database, you must have either the SYSDBA privilege or privileges on every materialized view in the database.

To set the system-level default values for statistics collection level and retention period, use the SET_SYSTEM_DEFAULT procedure.
Use the `DBA_MVREF_STATS_PARAMS` view to determine the currently-set retention period and collection level for materialized view statistics collection.

To disable refresh statistics collection for all materialized views in the database, use the following:

```sql
DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMS (NULL, 'NONE', NULL);
```

Note that the parameters set using `SET_MVREF_STATS_PARAMS` only affect materialized views that exist in the database at the time the procedure is run. Any new materialized views created after this procedure is run will use the system default values for `collection_level` and `retention_period`.

### 107.3.3 SET_SYSTEM_DEFAULT Procedure

This procedure sets system-wide defaults that manage the collection and retention of materialized view refresh statistics. All newly-created materialized views use these defaults until the parameters are reset explicitly using the `SET_MVREF_STATS_PARAMS` procedure.

#### Syntax

```sql
DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULT (    
    parameter_name        IN   VARCHAR2,    
    value             IN   VARCHAR2 DEFAULT NULL);
```

#### Parameters

**Table 107-4   SET_SYSTEM_DEFAULT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter_name</td>
<td>The name of the materialized view refresh statistics parameter whose system default value is being set. The parameters that can be set are:</td>
</tr>
<tr>
<td></td>
<td>• COLLECTION_LEVEL: Specifies the level of detail for collecting materialized view refresh statistics.</td>
</tr>
<tr>
<td></td>
<td>• RETENTION_PERIOD: Specifies the duration, in days, for which refresh statistics are retained in the data dictionary</td>
</tr>
<tr>
<td>value</td>
<td>The value of the materialized view refresh statistics parameter. The valid values for COLLECTION_LEVEL are:</td>
</tr>
<tr>
<td></td>
<td>• NONE: No refresh statistics are collected for the refresh operation.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: Only basic refresh statistics are collected for the refresh operation. This is the default setting.</td>
</tr>
<tr>
<td></td>
<td>• ADVANCED: Detailed refresh statistics are collected for the refresh operation.</td>
</tr>
<tr>
<td></td>
<td>The valid values for RETENTION_PERIOD are:</td>
</tr>
<tr>
<td></td>
<td>• -1</td>
</tr>
<tr>
<td></td>
<td>• Numbers between 1 and 1365000</td>
</tr>
<tr>
<td></td>
<td>The default value for retention_period is 31. If you specify NULL for any of the parameters, then the system default setting for that parameter is used.</td>
</tr>
</tbody>
</table>
Usage Notes

You must have **SYSDBA** privilege to invoke this procedure.

Use the **DBA_MVREF_STATS_SYS_DEFAULTS** view to display the current default settings for materialized view refresh statistics collection.
The `DBMS_NETWORK_ACL_ADMIN` package provides the interface to administer the network Access Control List (ACL).

The chapter contains the following topics:

- Overview
- Deprecated Subprograms
- Security Model
- Constants
- Exceptions
- Examples
- Summary of `DBMS_NETWORK_ACL_ADMIN` Subprograms

**See Also:**

For more information, see "Managing Fine-grained Access to External Network Services" in *Oracle Database Security Guide*

### 108.1 DBMS_NETWORK_ACL_ADMIN Overview

The `DBMS_NETWORK_ACL_ADMIN` package provides the interface to administer the network access control lists (ACL).

ACLs are used to control access by users to external network services and resources from the database through PL/SQL network utility packages including `UTL_TCP`, `UTL_HTTP`, `UTL_SMTP`, and `UTL_INADDR`.

### 108.2 DBMS_NETWORK_ACL_ADMIN Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with release Oracle Database 12c:

- `ADD_PRIVILEGE` Procedure
- `ASSIGN_ACL` Procedure
- `ASSIGN_WALLET_ACL` Procedure
- `CHECK_PRIVILEGE` Function
• CHECK_PRIVILEGE_ACLID Function
• CREATE_ACL Procedure
• DELETE_PRIVILEGE Procedure
• DROP_ACL Procedure
• UNASSIGN_ACL Procedure
• UNASSIGN_WALLET_ACL Procedure

108.3 DBMS_NETWORK_ACL_ADMIN Security Model

The EXECUTE privilege on the DBMS_NETWORK_ACL_ADMIN package is granted to the DBA role and to the EXECUTE_CATALOG_ROLE by default.

108.4 DBMS_NETWORK_ACL_ADMIN Constants

The DBMS_NETWORK_ACL_ADMIN package defines constants to use specifying parameter values.

These are shown in the following table.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_ADDR_MASK</td>
<td>VARCHAR2(80)</td>
<td>'([[:digit:]]+.){3}[[[:digit:]]]['</td>
<td>IP address mask: xxx.xxx.xxx.xxx</td>
</tr>
<tr>
<td>IP_SUBNET_MASK</td>
<td>VARCHAR2(80)</td>
<td>'([[:digit:]]+.){0,3}[*'</td>
<td>IP subnet mask: xxx.xxx...*</td>
</tr>
<tr>
<td>HOSTNAME_MASK</td>
<td>VARCHAR2(80)</td>
<td>'^[^.:/*]+([^.:/*]+)+['</td>
<td>Hostname mask: ???....??....??</td>
</tr>
<tr>
<td>DOMAIN_MASK</td>
<td>VARCHAR2(80)</td>
<td>''([^.:/*]+)['</td>
<td>Domain mask: *.???.???.??</td>
</tr>
</tbody>
</table>

108.5 DBMS_NETWORK_ACL_ADMIN Exceptions

The following table lists the exceptions raised by the DBMS_NETWORK_ACL_ADMIN package.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE_ALREADY_EXISTS</td>
<td>24243</td>
<td>ACE already exists</td>
</tr>
<tr>
<td>EMPTY_ACL</td>
<td>24246</td>
<td>Empty ACL</td>
</tr>
<tr>
<td>ACL_NOT_FOUND</td>
<td>46114</td>
<td>ACL not found</td>
</tr>
<tr>
<td>ACL_ALREADY_EXISTS</td>
<td>46212</td>
<td>ACL already exists</td>
</tr>
<tr>
<td>INVALID_ACL_PATH</td>
<td>46059</td>
<td>Invalid ACL path</td>
</tr>
</tbody>
</table>
Table 108-2  (Cont.) DBMS_NETWORK_ACL_ADMIN Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_HOST</td>
<td>24244</td>
<td>Invalid host</td>
</tr>
<tr>
<td>INVALID_PRIVILEGE</td>
<td>24245</td>
<td>Invalid privilege</td>
</tr>
<tr>
<td>INVALID_WALLET_PATH</td>
<td>29248</td>
<td>Invalid wallet path</td>
</tr>
<tr>
<td>BAD_ARGUMENT</td>
<td>29261</td>
<td>Bad argument</td>
</tr>
<tr>
<td>UNRESOLVED_PRINCIPAL</td>
<td>46238</td>
<td>Unresolved principal</td>
</tr>
<tr>
<td>PRIVILEGE_NOT_GRANTED</td>
<td>01927</td>
<td>Privilege not granted</td>
</tr>
</tbody>
</table>

108.6 DBMS_NETWORK_ACL_ADMIN Examples

Grant the connect and resolve privileges for host www.us.example.com to SCOTT.

Example 1

DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => 'www.us.example.com',
    ace => xs$ace_type(privilege_list => xs$name_list('connect', 'resolve'),
        principal_name => 'scott',
        principal_type => xs_acl.ptype_db));

Example 2

Revoke the resolve privilege for host www.us.example.com from SCOTT.

dbms_network_acl_admin.remove_host_ace(
    host => 'www.us.example.com',
    ace => xs$ace_type(privilege_list => xs$name_list('resolve'),
        principal_name => 'scott',
        principal_type => xs_acl.ptype_db));

Example 3

Grant the use_client_certificates and use_passwords privileges for wallet file:/example/wallets/hr_wallet to SCOTT.

dbms_network_acl_admin.append_wallet_ace(
    wallet_path => 'file:/example/wallets/hr_wallet',
    ace => xs$ace_type(privilege_list =>
        xs$name_list('use_client_certificates', 'use_passwords'),
        principal_name => 'scott',
        principal_type => xs_acl.ptype_db));

Example 4

Revoke the use_passwords privilege for wallet file:/example/wallets/hr_wallet from SCOTT.

dbms_network_acl_admin.remove_wallet_ace(
    wallet_path => 'file:/example/wallets/hr_wallet',
    ace => xs$ace_type(privilege_list => xs$name_list('use_passwords'),
        principal_name => 'scott',
        principal_type => xs_acl.ptype_db));
Example 5

The `CONTAINS_HOST` in the `DBMS_NETWORK_ACLUTILITY` package determines if a host is contained in a domain. It can be used in conjunction with the `DBA_HOST_ACE` view to determine the users and their privilege assignments to access a network host. For example, for access to `www.us.example.com`:

```
SELECT HOST, LOWER_PORT, UPPER_PORT, ACE_ORDER, PRINCIPAL, PRINCIPAL_TYPE, GRANT_TYPE, INVERTED_PRINCIPAL, PRIVILEGE, START_DATE, END_DATE
FROM (SELECT ACES.*,
    DBMS_NETWORK_ACLUTILITY.CONTAINS_HOST('www.us.example.com', HOST) PRECEDENCE
    FROM DBA_HOST_ACES ACES)
WHERE PRECEDENCE IS NOT NULL
ORDER BY PRECEDENCE DESC, LOWER_PORT NULLS LAST, UPPER_PORT NULLS LAST, ACE_ORDER;
```

<table>
<thead>
<tr>
<th>HOST</th>
<th>LOWER_PORT</th>
<th>UPPER_PORT</th>
<th>PRIVILEGE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.us.example.com">www.us.example.com</a></td>
<td>80</td>
<td>80</td>
<td>CONNECT</td>
<td>GRANTED</td>
</tr>
<tr>
<td><a href="http://www.us.example.com">www.us.example.com</a></td>
<td>80</td>
<td>80</td>
<td>RESOLVE</td>
<td>GRANTED</td>
</tr>
</tbody>
</table>

Example 6

For example, for `HQ_DBA`'s own permission to access to `www.us.example.com`:

```
SELECT HOST, LOWER_PORT, UPPER_PORT, PRIVILEGE, STATUS
FROM (SELECT ACES.*,
    DBMS_NETWORK_ACLUTILITY.CONTAINS_HOST('www.us.example.com', HOST) PRECEDENCE
    FROM USER_HOST_ACES ACES)
WHERE PRECEDENCE IS NOT NULL
ORDER BY PRECEDENCE DESC, LOWER_PORT NULLS LAST, UPPER_PORT NULLS LAST;
```

<table>
<thead>
<tr>
<th>HOST</th>
<th>LOWER_PORT</th>
<th>UPPER_PORT</th>
<th>PRIVILEGE</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td></td>
<td></td>
<td>CONNECT</td>
<td>GRANTED</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td>RESOLVE</td>
<td></td>
<td>GRANTED</td>
</tr>
</tbody>
</table>
### 108.7 Summary of DBMS_NETWORK_ACL_ADMIN Subprograms

This table lists and briefly describes the DBMS_NETWORK_ACL_ADMIN package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_PRIVILEGE Procedure</td>
<td>[DEPRECATED] Adds a privilege to grant or deny the network access to the user in an access control list (ACL)</td>
</tr>
<tr>
<td>APPEND_HOST_ACE Procedure</td>
<td>Appends an access control entry (ACE) to the access control list (ACL) of a network host.</td>
</tr>
<tr>
<td>APPEND_HOST_ACL Procedure</td>
<td>Appends access control entries (ACE) of an access control list (ACL) to the ACL of a network host</td>
</tr>
<tr>
<td>APPEND_WALLET_ACE Procedure</td>
<td>Appends an access control entry (ACE) to the access control list (ACL) of a wallet</td>
</tr>
<tr>
<td>APPEND_WALLET_ACL Procedure</td>
<td>Appends access control entries (ACE) of an access control list (ACL) to the ACL of a wallet</td>
</tr>
<tr>
<td>ASSIGN_ACL Procedure</td>
<td>[DEPRECATED] Assigns an access control list (ACL) to a host computer, domain, or IP subnet, and if specified, the TCP port range.</td>
</tr>
<tr>
<td>ASSIGN_WALLET_ACL Procedure</td>
<td>[DEPRECATED] Assigns an access control list (ACL) to a wallet</td>
</tr>
<tr>
<td>CHECK_PRIVILEGE Function</td>
<td>[DEPRECATED] Checks if a privilege is granted or denied the user in an access control list (ACL)</td>
</tr>
<tr>
<td>CHECK_PRIVILEGE_ACLID Function</td>
<td>[DEPRECATED] Checks if a privilege is granted to or denied from the user in an ACL by specifying the object ID of the access control list</td>
</tr>
<tr>
<td>CREATE_ACL Procedure</td>
<td>[DEPRECATED] Creates an access control list (ACL) with an initial privilege setting</td>
</tr>
<tr>
<td>DELETE_PRIVILEGE Procedure</td>
<td>[DEPRECATED] Deletes a privilege in an access control list (ACL)</td>
</tr>
<tr>
<td>DROP_ACL Procedure</td>
<td>[DEPRECATED] Drops an access control list (ACL)</td>
</tr>
<tr>
<td>REMOVE_HOST_ACE Procedure</td>
<td>Removes privileges from access control entries (ACE) in the access control list (ACL) of a network host matching the given ACE</td>
</tr>
<tr>
<td>REMOVE_WALLET_ACE Procedure</td>
<td>Removes privileges from access control entries (ACE) in the access control list (ACL) of a wallet matching the given ACE</td>
</tr>
<tr>
<td>SET_HOST_ACL Procedure</td>
<td>Sets the access control list (ACL) of a network host which controls access to the host from the database</td>
</tr>
<tr>
<td>SET_WALLET_ACL Procedure</td>
<td>Sets the access control list (ACL) of a wallet which controls access to the wallet from the database</td>
</tr>
<tr>
<td>UNASSIGN_ACL Procedure</td>
<td>[DEPRECATED] Unassigns the access control list (ACL) currently assigned to a network host</td>
</tr>
<tr>
<td>UNASSIGN_WALLET_ACL Procedure</td>
<td>[DEPRECATED] Unassigns the access control list (ACL) currently assigned to a wallet</td>
</tr>
</tbody>
</table>
108.7.1 ADD_PRIVILEGE Procedure

This procedure adds a privilege to grant or deny the network access to the user. The access control entry (ACE) is created if it does not exist.

Note:

This procedure is deprecated in Oracle Database 12c. While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the APPEND_HOST_ACE Procedure and the APPEND_WALLET_ACE Procedure.

Syntax

DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
    acl IN VARCHAR2,
    principal IN VARCHAR2,
    is_grant IN BOOLEAN,
    privilege IN VARCHAR2,
    position IN PLS_INTEGER DEFAULT NULL,
    start_date IN TIMESTAMP WITH TIMESTAMP DEFAULT NULL,
    end_date IN TIMESTAMP WITH TIMESTAMP DEFAULT NULL);

Parameters

Table 108-4 ADD_PRIVILEGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;</td>
</tr>
<tr>
<td>principal</td>
<td>Principal (database user or role) to whom the privilege is granted or denied. Case sensitive.</td>
</tr>
<tr>
<td>is_grant</td>
<td>Privilege is granted or denied.</td>
</tr>
<tr>
<td>privilege</td>
<td>Network privilege to be granted or denied</td>
</tr>
<tr>
<td>position</td>
<td>Position (1-based) of the ACE. If a non-NUL value is given, the privilege will be added in a new ACE at the given position and there should not be another ACE for the principal with the same is_grant (grant or deny). If a NULL value is given, the privilege will be added to the ACE matching the principal and the is_grant if one exists, or to the end of the ACL if the matching ACE does not exist.</td>
</tr>
<tr>
<td>start_date</td>
<td>Start date of the access control entry (ACE). When specified, the ACE will be valid only on and after the specified date. The start_date will be ignored if the privilege is added to an existing ACE.</td>
</tr>
<tr>
<td>end_date</td>
<td>End date of the access control entry (ACE). When specified, the ACE expires after the specified date. The end_date must be greater than or equal to the start_date. The end_date will be ignored if the privilege is added to an existing ACE.</td>
</tr>
</tbody>
</table>
Usage Notes

To remove the permission, use the DELETE_PRIVILEGE Procedure.

Examples

BEGIN
    DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE(
        acl => 'us-example-com-permissions.xml',
        principal => 'ST_USERS',
        is_grant => TRUE,
        privilege => 'connect'
    )
END;

108.7.2 APPEND_HOST_ACE Procedure

This procedure appends an access control entry (ACE) to the access control list (ACL) of a network host. The ACL controls access to the given host from the database and the ACE specifies the privileges granted to or denied from the specified principal.

Syntax

DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE (
    host IN VARCHAR2,
    lower_port IN PLS_INTEGER DEFAULT NULL,
    upper_port IN PLS_INTEGER DEFAULT NULL,
    ace IN XS$ACE_TYPE);

Parameters

Table 108-5  APPEND_HOST_ACE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The host, which can be the name or the IP address of the host. You can use a wildcard to specify a domain or a IP subnet. The host or domain name is case-insensitive.</td>
</tr>
<tr>
<td>lower_port</td>
<td>Lower bound of an optional TCP port range</td>
</tr>
<tr>
<td>upper_port</td>
<td>Upper bound of an optional TCP port range. If NULL, lower_port is assumed.</td>
</tr>
<tr>
<td>ace</td>
<td>The ACE</td>
</tr>
</tbody>
</table>

Usage Notes

- Duplicate privileges in the matching ACE in the host ACL will be skipped.
- To remove the ACE, use the REMOVE_HOST_ACE Procedure.
- A host's ACL takes precedence over its domains' ACLs. For a given host, say www.us.example.com, the following domains are listed in decreasing precedence:
  - www.us.example.com
  - *.us.example.com
  - *.example.com
  - *.com
An IP address' ACL takes precedence over its subnets' ACLs. For a given IP address, say 192.168.0.100, the following subnets are listed in decreasing precedence:

- 192.168.0.100
- 192.168.0.*
- 192.168.*
- 192.*
- *

An ACE with a "resolve" privilege can be appended only to a host's ACL without a port range.

When ACEs with "connect" privileges are appended to a host's ACLs with and without a port range, the one appended to the host with a port range takes precedence.

When specifying a TCP port range of a host, it cannot overlap with other existing port ranges of the host.

If the ACL is shared with another host or wallet, a copy of the ACL will be made before the ACL is modified.

See Also:

Oracle Database Real Application Security Administrator's and Developer's Guide for more information about the XS$ACE_TYPE object type

108.7.3 APPEND_HOST_ACL Procedure

This procedure appends access control entries (ACE) of an access control list (ACL) to the ACL of a network host.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACL (host IN VARCHAR2,
    lower_port IN PLS_INTEGER DEFAULT NULL,
    upper_port IN PLS_INTEGER DEFAULT NULL,
    acl IN VARCHAR2);
```

Parameters

Table 108-6 APPEND_HOST_ACL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The host, which can be the name or the IP address of the host. You can use a wildcard to specify a domain or a IP subnet. The host or domain name is case-insensitive.</td>
</tr>
<tr>
<td>lower_port</td>
<td>Lower bound of an optional TCP port range</td>
</tr>
</tbody>
</table>
### Table 108-6 (Cont.) APPEND_HOST_ACL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper_port</td>
<td>Upper bound of an optional TCP port range. If <strong>NULL</strong>, <code>lower_port</code> is assumed.</td>
</tr>
<tr>
<td>acl</td>
<td>The ACL from which to append</td>
</tr>
</tbody>
</table>

### Usage Notes

- Duplicate privileges in the matching ACE in the host ACL will be skipped.
- To remove the ACE, use the **REMOVE_HOST_ACE Procedure**.
- A host's ACL takes precedence over its domains' ACLs. For a given host, say `www.us.example.com`, the following domains are listed in decreasing precedence:
  - `www.us.example.com`
  - `*.us.example.com`
  - `*.example.com`
  - `*.com`
  - `*`
- An IP address' ACL takes precedence over its subnets' ACLs. For a given IP address, say `192.168.0.100`, the following subnets are listed in decreasing precedence:
  - `192.168.0.100`
  - `192.168.0.*`
  - `192.168.*`
  - `192.*`
  - `*`
- An ACE with a "resolve" privilege can be appended only to a host's ACL without a port range.
- When ACEs with "connect" privileges are appended to a host's ACLs with and without a port range, the one appended to the host with a port range takes precedence.
- When specifying a TCP port range of a host, it cannot overlap with other existing port ranges of the host.- If the ACL is shared with another host or wallet, a copy of the ACL will be made before the ACL is modified.

### 108.7.4 APPEND_WALLET_ACE Procedure

This procedure appends an access control entry (ACE) to the access control list (ACL) of a wallet. The ACL controls access to the given wallet from the database and the ACE specifies the privileges granted to or denied from the specified principal.

#### Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE (wallet_path IN VARCHAR2,
                                          ace IN XS$ACE_TYPE);
```
Parameters

Table 108-7  APPEND_WALLET_ACE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_path</td>
<td>Directory path of the wallet. The path is case-sensitive of the format file:directory-path.</td>
</tr>
<tr>
<td>ace</td>
<td>The ACE</td>
</tr>
</tbody>
</table>

Usage Notes

• Duplicate privileges in the matching ACE in the host ACL will be skipped.
• To remove the ACE, use the REMOVE_WALLET_ACE Procedure.
• If the ACL is shared with another host or wallet, a copy of the ACL is made before the ACL is modified.

See Also:

Oracle Database Real Application Security Administrator's and Developer's Guide for more information about the $XSACE_TYPE object type

108.7.5 APPEND_WALLET_ACL Procedure

This procedure appends access control entries (ACE) of an access control list (ACL) to the ACL of a wallet.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACL (
    wallet_path    IN VARCHAR2,
    acl            IN VARCHAR2);
```

Parameters

Table 108-8  APPEND_WALLET_ACL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_path</td>
<td>Directory path of the wallet. The path is case-sensitive of the format file:directory-path.</td>
</tr>
<tr>
<td>ace</td>
<td>The ACL from which to append</td>
</tr>
</tbody>
</table>

Usage Notes

• Duplicate privileges in the matching ACE in the host ACL will be skipped.
• To remove the ACE, use REMOVE_WALLET_ACE.
• If the ACL is shared with another host or wallet, a copy of the ACL is made before the ACL is modified.
108.7.6 ASSIGN_ACL Procedure

This procedure assigns an access control list (ACL) to a host computer, domain, or IP subnet, and if specified, the TCP port range.

Note:

This procedure is deprecated in Oracle Database 12c. While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the APPEND_HOST_ACE Procedure and the APPEND_WALLET_ACE Procedure.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL (  
    acl         IN VARCHAR2,  
    host        IN VARCHAR2,  
    lower_port  IN PLS_INTEGER DEFAULT NULL,  
    upper_port  IN PLS_INTEGER DEFAULT NULL);
```

Parameters

Table 108-9 ASSIGN_ACL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;.</td>
</tr>
<tr>
<td>host</td>
<td>Host to which the ACL is to be assigned. The host can be the name or the IP address of the host. A wildcard can be used to specify a domain or a IP subnet. The host or domain name is case-insensitive.</td>
</tr>
<tr>
<td>lower_port</td>
<td>Lower bound of a TCP port range if not NULL</td>
</tr>
<tr>
<td>upper_port</td>
<td>Upper bound of a TCP port range. If NULL, lower_port is assumed.</td>
</tr>
</tbody>
</table>

Usage Notes

- Only one ACL can be assigned to any host computer, domain, or IP subnet, and if specified, the TCP port range. When you assign a new access control list to a network target, Oracle Database unassigns the previous access control list that was assigned to the same target. However, Oracle Database does not drop the access control list. You can drop the access control list by using the DROP_ACL Procedure. To remove an access control list assignment, use the UNASSIGN_ACL Procedure.

- The ACL assigned to a domain takes a lower precedence than the other ACLs assigned sub-domains, which take a lower precedence than the ACLs assigned to the individual hosts. So for a given host, for example, "www.us.example.com", the following domains are listed in decreasing precedences:
  - www.us.example.com
  - *.us.example.com
  - *.example.com
In the same way, the ACL assigned to a subnet takes a lower precedence than the other ACLs assigned smaller subnets, which take a lower precedence than the ACLs assigned to the individual IP addresses. So for a given IP address, for example, "192.168.0.100", the following subnets are listed in decreasing precedences:
- 192.168.0.100
- 192.168.0.*
- 192.168.*
- 192.*
- *

- The port range is applicable only to the "connect" privilege assignments in the ACL. The "resolve" privilege assignments in an ACL have effects only when the ACL is assigned to a host without a port range.

  For the "connect" privilege assignments, an ACL assigned to the host without a port range takes a lower precedence than other ACLs assigned to the same host with a port range.

- When specifying a TCP port range, both lower_port and upper_port must not be NULL and upper_port must be greater than or equal to lower_port. The port range must not overlap with any other port ranges for the same host assigned already.

- To remove the assignment, use UNASSIGN_ACL Procedure.

Examples
BEGIN
  DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL(
    acl         => 'us-example-com-permissions.xml',
    host        => '*.us.example.com',
    lower_port  => 80);
END;

108.7.7 ASSIGN_WALLET_ACL Procedure

This procedure assigns an access control list (ACL) to a wallet.

Note:
This procedure is deprecated in Oracle Database 12c. While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the APPEND_HOST_ACE Procedure and the APPEND_WALLET_ACE Procedure.

Syntax
UTL_HTTP.ASSIGN_WALLET_ACL (
  acl      IN VARCHAR2,
  wallet_path IN VARCHAR2);
Parameters

Table 108-10 ASSIGN_WALLET_ACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;</td>
</tr>
<tr>
<td>wallet_path</td>
<td>Directory path of the wallet to which the ACL is to be assigned. The path is case-sensitive and of the format file:directory-path.</td>
</tr>
</tbody>
</table>

Usage Notes

To remove the assignment, use the UNASSIGN_WALLET_ACL Procedure.

Examples

BEGIN
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
    acl => 'wallet-acl.xml',
    description => 'Wallet ACL',
    principal => 'SCOTT',
    is_grant => TRUE,
    privilege => 'use-client-certificates');

  DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE(
    acl => 'wallet-acl.xml',
    principal => 'SCOTT',
    is_grant => TRUE,
    privilege => 'use-passwords');

  DBMS_NETWORK_ACL_ADMIN.ASSIGN_WALLET_ACL(
    acl => 'wallet-acl.xml',
    wallet_path => 'file:/example/wallets/test_wallet');
END;

108.7.8 CHECK_PRIVILEGE Function

This function checks if a privilege is granted or denied the user in an ACL.

**Note:**

This procedure is deprecated in Oracle Database 12c. The procedure remains available in the package only for reasons of backward compatibility.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE (
  acl IN VARCHAR2,
  user IN VARCHAR2,
  privilege IN VARCHAR2)
RETURN NUMBER;
```
Parameters

Table 108-11  CHECK_PRIVILEGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;.</td>
</tr>
<tr>
<td>user</td>
<td>User to check against. If the user is NULL, the invoker is assumed. The username is case-sensitive as in the USERNAME column of the ALL_USERS view.</td>
</tr>
<tr>
<td>privilege</td>
<td>Network privilege to check</td>
</tr>
</tbody>
</table>

Return Values

Returns 1 when the privilege is granted; 0 when the privilege is denied; NULL when the privilege is neither granted or denied.

Examples

```sql
SELECT DECODE(
    DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE(
        'us-example-com-permissions.xml', 'SCOTT', 'resolve'),
    1, 'GRANTED', 0, 'DENIED', NULL) PRIVILEGE
FROM DUAL;
```

108.7.9 CHECK_PRIVILEGE_ACLID Function

This function checks if a privilege is granted to or denied from the user in an ACL by specifying the object ID of the access control list.

**Note:**

This procedure is deprecated in Oracle Database 12c. The procedure remains available in the package only for reasons of backward compatibility.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID (
    aclid           IN RAW,
    user            IN VARCHAR2 DEFAULT NULL,
    privilege       IN VARCHAR2,
) RETURN NUMBER;
```

Parameters

Table 108-12  CHECK_PRIVILEGE_ACLID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclid</td>
<td>Object ID of the ACL</td>
</tr>
</tbody>
</table>
Table 108-12  (Cont.) CHECK_PRIVILEGE_ACLID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>User to check against. If the user is NULL, the invoker is assumed. The username is case-sensitive as in the USERNAME column of the ALL_USERS view.</td>
</tr>
<tr>
<td>privilege</td>
<td>Network privilege to check</td>
</tr>
</tbody>
</table>

Return Values

Returns 1 when the privilege is granted; 0 when the privilege is denied; NULL when the privilege is neither granted or denied.

108.7.10 CREATE_ACL Procedure

This deprecated procedure creates an access control list (ACL) with an initial privilege setting. An ACL must have at least one privilege setting. The ACL has no access control effect unless it is assigned to the network target.

Note:

This procedure is deprecated in Oracle Database 12c. While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the APPEND_HOST_ACE Procedure and the APPEND_WALLET_ACE Procedure.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (  
acl            IN VARCHAR2,  
description    IN VARCHAR2,  
principal      IN VARCHAR2,  
is_grant       IN BOOLEAN,  
privilege      IN VARCHAR2,  
start_date     IN TIMESTAMP DEFAULT NULL,  
end_date       IN TIMESTAMP DEFAULT NULL );
```

Parameters

Table 108-13  CREATE_ACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;.</td>
</tr>
<tr>
<td>description</td>
<td>Description attribute in the ACL</td>
</tr>
<tr>
<td>principal</td>
<td>Principal (database user or role) to whom the privilege is granted or denied. Case sensitive.</td>
</tr>
<tr>
<td>is_grant</td>
<td>Privilege is granted or not (denied)</td>
</tr>
</tbody>
</table>
Table 108-13 (Cont.) CREATE_ACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>Network privilege to be granted or denied - 'connect</td>
</tr>
<tr>
<td>start_date</td>
<td>Start date of the access control entry (ACE). When specified, the ACE is valid only on and after the specified date.</td>
</tr>
<tr>
<td>end_date</td>
<td>End date of the access control entry (ACE). When specified, the ACE expires after the specified date. The end_date must be greater than or equal to the start_date.</td>
</tr>
</tbody>
</table>

Usage Notes

To drop the access control list, use the DROP_ACL Procedure.

Examples

BEGIN
    DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
        acl => 'us-example-com-permissions.xml',
        description => 'Network permissions for *.us.example.com',
        principal => 'SCOTT',
        is_grant => TRUE,
        privilege => 'connect');
END;

108.7.11 DELETE_PRIVILEGE Procedure

This deprecated procedure deletes a privilege in an access control list.

Note:

This procedure is deprecated in Oracle Database 12c. While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the REMOVE_HOST_ACE Procedure and the REMOVE_WALLET_ACE Procedure.

Syntax

DBMS_NETWORK_ACL_ADMIN.DELETE_PRIVILEGE (  
    acl IN VARCHAR2,  
    principal IN VARCHAR2,  
    is_grant IN BOOLEAN DEFAULT NULL,  
    privilege IN VARCHAR2 DEFAULT NULL);
Parameters

Table 108-14  DELETE_PRIVILEGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;.</td>
</tr>
<tr>
<td>principal</td>
<td>Principal (database user or role) for whom all the ACE will be deleted</td>
</tr>
<tr>
<td>is_grant</td>
<td>Privilege is granted or not (denied). If a NULL value is given, the deletion is applicable to both granted or denied privileges.</td>
</tr>
<tr>
<td>privilege</td>
<td>Network privilege to be deleted. If a NULL value is given, the deletion is applicable to all privileges.</td>
</tr>
</tbody>
</table>

Examples

BEGIN
    DBMS_NETWORK_ACL_ADMIN.DELETE_PRIVILEGE(
        acl        => 'us-example-com-permissions.xml',
        principal => 'ST_USERS')
END;

108.7.12 DROP_ACL Procedure

This deprecated procedure drops an access control list (ACL).

⚠️ Note:

This procedure is deprecated in Oracle Database 12c. The procedure remains available in the package only for reasons of backward compatibility.

Syntax

DBMS_NETWORK_ACL_ADMIN.DROP_ACL ( acl IN VARCHAR2);

Parameters

Table 108-15  DROP_ACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;.</td>
</tr>
</tbody>
</table>

Examples

BEGIN
    DBMS_NETWORK_ACL_ADMIN.DROP_ACL( 
        acl => 'us-example-com-permissions.xml');
END;
108.7.13 REMOVE_HOST_ACE Procedure

This procedure removes privileges from access control entries (ACE) in the access control list (ACL) of a network host matching the given ACE.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.REMOVE_HOST_ACE (  
    host               IN VARCHAR2,  
    lower_port         IN PLS_INTEGER DEFAULT NULL,  
    upper_port         IN PLS_INTEGER DEFAULT NULL,  
    ace                IN XS$ACE_TYPE,  
    remove_empty_acl   IN BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The host, which can be the name or the IP address of the host. You can use a wildcard to specify a domain or a IP subnet. The host or domain name is case-insensitive.</td>
</tr>
<tr>
<td>lower_port</td>
<td>Lower bound of an optional TCP port range</td>
</tr>
<tr>
<td>upper_port</td>
<td>Upper bound of an optional TCP port range. If NULL, <code>lower_port</code> is assumed.</td>
</tr>
<tr>
<td>ace</td>
<td>The ACE</td>
</tr>
<tr>
<td>remove_empty_acl</td>
<td>Whether to remove the ACL when it becomes empty when the ACE is removed</td>
</tr>
</tbody>
</table>

Usage Notes

If the ACL is shared with another host or wallet, a copy of the ACL is made before the ACL is modified.

108.7.14 REMOVE_WALLET_ACE Procedure

This procedure removes privileges from access control entries (ACE) in the access control list (ACL) of a wallet matching the given ACE.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.REMOVE_WALLET_ACE (  
    wallet_path        IN VARCHAR2,  
    ace                IN XS$ACE_TYPE,  
    remove_empty_acl   IN BOOLEAN DEFAULT FALSE);
```
Parameters

Table 108-17 REMOVE_WALLET_ACE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_path</td>
<td>Directory path of the wallet. The path is case-sensitive of the format file:directory-path.</td>
</tr>
<tr>
<td>ace</td>
<td>The ACE</td>
</tr>
<tr>
<td>remove_empty_acl</td>
<td>Whether to remove the ACL when it becomes empty when the ACE is removed</td>
</tr>
</tbody>
</table>

Usage Notes

If the ACL is shared with another host or wallet, a copy of the ACL is made before the ACL is modified.

108.7.15 SET_HOST_ACL Procedure

This procedure sets the access control list (ACL) of a network host which controls access to the host from the database.

Syntax

```
DBMS_NETWORK_ACL_ADMIN.SET_HOST_ACL ( 
    host           IN VARCHAR2, 
    lower_port     IN PLS_INTEGER DEFAULT NULL, 
    upper_port     IN PLS_INTEGER DEFAULT NULL, 
    acl            IN VARCHAR2); 
```

Parameters

Table 108-18 SET_HOST_ACL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The host, which can be the name or the IP address of the host. You can use a wildcard to specify a domain or a IP subnet. The host or domain name is case-insensitive.</td>
</tr>
<tr>
<td>lower_port</td>
<td>Lower bound of an optional TCP port range</td>
</tr>
<tr>
<td>upper_port</td>
<td>Upper bound of an optional TCP port range. If NULL, lower_port is assumed.</td>
</tr>
<tr>
<td>acl</td>
<td>The ACL. NULL to unset the host's ACL.</td>
</tr>
</tbody>
</table>

Usage Notes

A host's ACL is created and set on-demand when an access control entry (ACE) is appended to the host's ACL. Users are discouraged from setting a host's ACL manually.
108.7.16 SET_WALLET_ACL Procedure

This procedure sets the access control list (ACL) of a wallet which controls access to the wallet from the database.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.SET_WALLET_ACL (
    wallet_path    IN VARCHAR2,
    acl            IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_path</td>
<td>Directory path of the wallet. The path is case-sensitive of the format file:directory-path.</td>
</tr>
<tr>
<td>acl</td>
<td>The ACL. NULL to unset the host's ACL.</td>
</tr>
</tbody>
</table>

Usage Notes

A wallet's ACL is created and set on-demand when an access control entry (ACE) is appended to the wallet's ACL. Users are discouraged from setting a wallet's ACL manually.

108.7.17 UNASSIGN_ACL Procedure

This deprecated procedure unassigns the access control list (ACL) currently assigned to a network host.

**Note:**

This procedure is deprecated in Oracle Database 12c. While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the REMOVE_HOST_ACE Procedure and the REMOVE_WALLET_ACE Procedure.

Syntax

```sql
DBMS_NETWORK_ACL_ADMIN.UNASSIGN_ACL (
    acl         IN VARCHAR2 DEFAULT NULL,
    host        IN VARCHAR2 DEFAULT NULL,
    lower_port  IN PLS_INTEGER DEFAULT NULL,
    upper_port  IN PLS_INTEGER DEFAULT NULL);
```
Parameters

Table 108-20   UNASSIGN_ACL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;. If ACL is NULL, any ACL assigned to the host is unassigned.</td>
</tr>
<tr>
<td>host</td>
<td>Host from which the ACL is to be removed. The host can be the name or the IP address of the host. A wildcard can be used to specify a domain or a IP subnet. The host or domain name is case-insensitive. If host is NULL, the ACL will be unassigned from any host. If both host and acl are NULL, all ACLs assigned to any hosts are unassigned.</td>
</tr>
<tr>
<td>lower_port</td>
<td>Lower bound of a TCP port range if not NULL</td>
</tr>
<tr>
<td>upper_port</td>
<td>Upper bound of a TCP port range. If NULL, lower_port is assumed.</td>
</tr>
</tbody>
</table>

Examples

BEGIN
    DBMS_NETWORK_ACL_ADMIN.UNASSIGN_ACL(
        host        => '*.us.example.com',
        lower_port  => 80);
END;

108.7.18 UNASSIGN_WALLET_ACL Procedure

This deprecated procedure unassigns the access control list (ACL) currently assigned to a wallet.

Note:

This procedure is deprecated in Oracle Database 12c. While the procedure remains available in the package for reasons of backward compatibility, Oracle recommends using the REMOVE_HOST_ACE Procedure and the REMOVE_WALLET_ACE Procedure.

Syntax

    UTL_HTTP.UNASSIGN_WALLET_ACL ( 
        acl IN VARCHAR2 DEFAULT NULL,
        wallet_path IN VARCHAR2 DEFAULT NULL);

Parameters

Table 108-21   UNASSIGN_WALLET_ACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>Name of the ACL. Relative path will be relative to &quot;/sys/acls&quot;. If acl is NULL, any ACL assigned to the wallet is unassigned</td>
</tr>
</tbody>
</table>
Table 108-21  (Cont.) UNASSIGN_WALLET_ACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_path</td>
<td>Directory path of the wallet to which the ACL is assigned. The path is</td>
</tr>
<tr>
<td></td>
<td>case-sensitive and of the format file:directory-path. If both acl and</td>
</tr>
<tr>
<td></td>
<td>wallet_path are NULL, all ACLs assigned to any wallets are unassigned.</td>
</tr>
</tbody>
</table>

Examples

BEGIN
    DBMS_NETWORK_ACL_ADMIN.UNASSIGN_WALLET_ACL(
        acl => 'wallet-acl.xml',
        wallet_path => 'file:/example/wallets/test_wallet');
END;
The `DBMS_NETWORK_ACL_UTIL` package provides the utility functions to facilitate the evaluation of access control list (ACL) assignments governing TCP connections to network hosts.

### See Also:
For more information, see "Managing Fine-grained Access to External Network Services" in *Oracle Database Security Guide*

The chapter contains the following topics:
- Security Model
- Examples
- Summary of `DBMS_NETWORK_ACL_UTIL` Subprograms

#### 109.1 DBMS_NETWORK_ACL_UTIL Security Model

EXECUTE on the `DBMS_NETWORK_ACL_UTIL` package is granted to PUBLIC.

#### 109.2 DBMS_NETWORK_ACL_UTIL Examples

The `CONTAINS_HOST` Function in this package indicates if a domain or subnet contains a given host or IP address.

It can be used in conjunction with the `CHECK_PRIVILEGE_ACLID` Function in the `DBMS_NETWORK_ACL_ADMIN` package to determine the privilege assignments affecting a user's permission to access a network host. The return value of the `CONTAINS_HOST` Function in can also be used to order the ACL assignments by their precedence.

**Example 1**

For example, for SCOTT's permission to connect to www.hr.example.com:

```sql
SELECT host, lower_port, upper_port, acl,
       DECODE(
           DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid, 'SCOTT', 'connect'),
           1, 'GRANTED', 0, 'DENIED', NULL) privilege
FROM (SELECT host, acl, aclid, lower_port, upper_port,
       DBMS_NETWORK_ACL_UTIL.CONTAINS_HOST('www.hr.example.com', host)
       precedence
FROM dba_network_acls)
WHERE precedence > 0
ORDER BY precedence DESC, lower_port NULLS LAST;
```
Example 2

For example, for SCOTT’s permission to do domain name resolution for www.hr.example.com:

```sql
SELECT host, acl,
  DECODE(
    DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid, 'SCOTT', 'resolve'),
    1, 'GRANTED', 0, 'DENIED', null) privilege
FROM (SELECT host, acl, aclid,
    DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST('www.hr.example.com', host)
  FROM dba_network_acls
  WHERE lower_port IS NULL AND upper_port IS NULL)
WHERE precedence > 0
ORDER BY precedence DESC;
```

Note that the "resolve" privilege takes effect only in ACLs assigned without any port range (when `lower_port` and `upper_port` are NULL). For this reason, the example does not include `lower_port` and `upper_port` columns in the query.

Related Topics

- **CONTAINS_HOST Function**
  This function determines if the given host is equal to or contained in the given host, domain, or subnet. It handles different representation of the same IP address or subnet. For example, an IPv4-mapped IPv6 address is considered equal to the IPv4-native address it represents. It does not perform domain name resolution when evaluating the host or domain.

### 109.3 Summary of DBMS_NETWORK_ACLUtility Subprograms

This table lists and briefly describes the `DBMS_NETWORK_ACLUtility` package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONTAINS_HOST Function</strong></td>
<td>Determines if the given host is equal to or contained in the given host, domain, or subnet</td>
</tr>
</tbody>
</table>
Table 109-1  (Cont.) DBMS_NETWORK_ACL_UTILITY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN_LEVEL Function</td>
<td>Returns the domain level of the given host name, domain, or subnet</td>
</tr>
<tr>
<td>DOMAINS Function</td>
<td>For a given host, this function returns the domains whose ACL assigned is used to determine if a user has the privilege to access the given host or not.</td>
</tr>
<tr>
<td>EQUALS_HOST Function</td>
<td>Determines if the two given hosts, domains, or subnets are equal</td>
</tr>
</tbody>
</table>

109.3.1 CONTAINS_HOST Function

This function determines if the given host is equal to or contained in the given host, domain, or subnet. It handles different representation of the same IP address or subnet. For example, an IPv4-mapped IPv6 address is considered equal to the IPv4-native address it represents. It does not perform domain name resolution when evaluating the host or domain.

Syntax

```sql
DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST ( 
    host    IN    VARCHAR2, 
    domain IN    VARCHAR2) 
RETURN NUMBER;
```

Parameters

Table 109-2  CONTAINS_HOST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>Network host</td>
</tr>
<tr>
<td>domain</td>
<td>Network host, domain, or subnet</td>
</tr>
</tbody>
</table>

Return Values

Returns a non-NULL value if the given host is equal to or contained in the related host, domain, or subnet:

- If `domain` is a hostname, returns the level of its domain + 1
- If `domain` is a domain name, returns the domain level
- If `domain` is an IP address or subnet, return the number of significant address bits of the IP address or subnet
- If `domain` is the wildcard "\*", returns 0

The non-NULL value returned indicates the precedence of the domain or subnet for ACL assignment. The higher the value, the higher is the precedence. NULL will be returned if the host is not equal to or contained in the given host, domain or subnet.
Examples

SELECT host, acl, precedence
FROM (select host, acl,
    DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST('192.0.2.3', host)
    precedence
FROM dba_network_acls)
WHERE precedence > 0
ORDER BY precedence DESC;

<table>
<thead>
<tr>
<th>HOST</th>
<th>ACL</th>
<th>PRECEDENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.3</td>
<td>/sys/acls/hr-www.xml</td>
<td>32</td>
</tr>
<tr>
<td>::ffff:192.0.2.0/120</td>
<td>/sys/acls/hr-domain.xml</td>
<td>24</td>
</tr>
<tr>
<td>::ffff:192.0.0.0/104</td>
<td>/sys/acls/corp-domain.xml</td>
<td>8</td>
</tr>
</tbody>
</table>

109.3.2 DOMAIN_LEVEL Function

This function returns the domain level of the given host name, domain, or subnet.

Syntax

DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL (host IN VARCHAR2) RETURN NUMBER;

Parameters

Table 109-3  DOMAIN_LEVEL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>Network host, domain, or subnet</td>
</tr>
</tbody>
</table>

Return Values

The domain level of the given host, domain, or subnet.

Usage Notes

Note that this function cannot handle IPv6 addresses and subnets, and subnets in CIDR notation.

Examples

SELECT host, acl, domain_level
FROM (select host, acl,
    DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL(host) domain_level
FROM dba_network_acls)
order by domain_level desc;

<table>
<thead>
<tr>
<th>HOST</th>
<th>ACL</th>
<th>DOMAIN_LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.hr.example.com">www.hr.example.com</a></td>
<td>/sys/acls/hr-www.xml</td>
<td>4</td>
</tr>
<tr>
<td>*.hr.example.com</td>
<td>/sys/acls/hr-domain.xml</td>
<td>3</td>
</tr>
<tr>
<td>*.example.com</td>
<td>/sys/acls/corp-domain.xml</td>
<td>2</td>
</tr>
</tbody>
</table>
109.3.3 DOMAINS Function

For a given host, this function returns the domains whose ACL assigned determines if a user has the privilege to access the given host or not. When the IP address of the host is given, return the subnets instead.

Syntax

```
DBMS_NETWORK_ACL_UTILITY.DOMAINS (
    host  IN VARCHAR2)
RETURN DOMAIN_TABLE PIPELINED;
```

Parameters

**Table 109-4  DOMAINS Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>Network host</td>
</tr>
</tbody>
</table>

Return Values

The domains or subnets for the given host.

Usage Notes

Note that this function cannot handle IPv6 addresses. Nor can it generate subnets of arbitrary number of prefix bits for an IPv4 address.

Examples

```
select * from table(dbms_network_acl_utility.domains('www.hr.example.com'));
```

```
DOMAINS
--------------------------
www.hr.example.com
*.hr.example.com
*.example.com
*.com
*
```

109.3.4 EQUALS_HOST Function

This function determines if the two given hosts, domains, or subnets are equal. It handles different representation of the same IP address or subnet. For example, an IPv4-mapped IPv6 address is considered equal to the IPv4-native address it represents. It does not perform domain name resolution when comparing the two hosts or domains.

Syntax

```
DBMS_NETWORK_ACL_UTILITY.EQUALS_HOST (
    host1  IN VARCHAR2,
    host2  IN VARCHAR2)
RETURN NUMBER;
```
Parameters

Table 109-5  EQUALS_HOST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host1</td>
<td>Network host, domain, or subnet to compare</td>
</tr>
<tr>
<td>host2</td>
<td>Network host, domain, or subnet to compare</td>
</tr>
</tbody>
</table>

Return Values

1 if the two hosts, domains, or subnets are equal. 0 otherwise.

Examples

```
SELECT host, acl
FROM dba_network_acls
WHERE DBMS_NETWORK_ACL_UTILITY.EQUALS_HOST('192.0.2.*', host) = 1;
```

<table>
<thead>
<tr>
<th>HOST</th>
<th>ACL</th>
</tr>
</thead>
<tbody>
<tr>
<td>::ffff:192.0.2.0/120</td>
<td>/sys/acls/hr-domain.xml</td>
</tr>
</tbody>
</table>
DBMS_ODCI package contains a single user function related to the use of Data Cartridges.

This chapter contains the following topic:

- Summary of DBMS_ODCI Subprograms

See Also:

- Oracle Database Data Cartridge Developer's Guide

### 110.1 Summary of DBMS_ODCI Subprograms

The DBMS_ODCI package has one subprogram, the `ESTIMATE_CPU_UNITS` function.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ESTIMATE_CPU_UNITS</code> Function</td>
<td>Returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds)</td>
</tr>
</tbody>
</table>

### 110.1.1 ESTIMATE_CPU_UNITS Function

This function returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds). This information can be used to associate the CPU cost with a user-defined function for the extensible optimizer.

The function takes as input the elapsed time of the user function, measures CPU units by multiplying the elapsed time by the processor speed of the machine, and returns the approximate number of CPU instructions that should be associated with the user function. For a multiprocessor machine, `ESTIMATE_CPU_UNITS` considers the speed of a single processor.

**Syntax**

```sql
DBMS_ODCI.ESTIMATE_CPU_UNITS(
    elapsed_time       NUMBER)
RETURN NUMBER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elapsed_time</td>
<td>The elapsed time in seconds that it takes to execute a function.</td>
</tr>
</tbody>
</table>
Usage Notes

When associating CPU cost with a user-defined function, use the full number of CPU units rather than the number of thousands of CPU units returned by `ESTIMATE_CPU_UNITS`; multiply the number returned by `ESTIMATE_CPU_UNITS` by 1,000.
The DBMS_OUTLN package, synonymous with OUTLN_PKG, contains the functional interface for subprograms associated with the management of stored outlines.

**See Also:**
For more information about using the DBMS_OUTLN package, see Oracle Database SQL Tuning Guide.

**Note:**
Stored outlines will be desupported in a future release in favor of SQL plan management. In Oracle Database 11g Release 1 (11.1), stored outlines continue to function as in past releases. However, Oracle strongly recommends that you use SQL plan management for new applications. SQL plan management creates SQL plan baselines, which offer superior SQL performance and stability compared with stored outlines. If you have existing stored outlines, please consider migrating them to SQL plan baselines by using the LOAD_PLANS_FROM_CURSOR_CACHE Functions or LOAD_PLANS_FROM_SQLSET Function of the DBMS_SPM package. When the migration is complete, you should disable or remove the stored outlines.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_OUTLN Subprograms

111.1 DBMS_OUTLN Overview

A stored outline is the stored data that pertains to an execution plan for a given SQL statement. It enables the optimizer to repeatedly re-create execution plans that are equivalent to the plan originally generated along with the outline. The data stored in an outline consists, in part, of a set of hints that are used to achieve plan stability.

Stored outlines will be de-supported in a future release in favor of SQL plan management. As of 11g R1, stored outlines continue to function as in past releases, but Oracle strongly recommends that you use SQL plan management for new applications. SQL plan management creates SQL plan baselines, which offer superior SQL performance and stability compared with stored outlines. If you have existing stored outlines, please consider migrating them to SQL plan baselines by using the LOAD_PLANS_FROM_CURSOR_CACHE Functions or the LOAD_PLANS_FROM_SQLSET Function of the DBMS_SPM package. When the migration is complete, you should disable or remove the stored outlines.
111.2 DBMS_OUTLN Security Model

DBMS_OUTLN contains management procedures that should be available to appropriate users only. EXECUTE privilege is not extended to the general user community unless the DBA explicitly does so.

PL/SQL functions that are available for outline management purposes can be executed only by users with EXECUTE privilege on the procedure (or package).

111.3 Summary of DBMS_OUTLN Subprograms

This table lists the DBMS_OUTLN subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR_USED Procedure</td>
<td>Clears the outline 'used' flag</td>
</tr>
<tr>
<td>CREATE_OUTLINE Procedure</td>
<td>Generates outlines from the shared cursor identified by hash value and child number</td>
</tr>
<tr>
<td>DROP_BY_CAT Procedure</td>
<td>Drops outlines that belong to a specified category</td>
</tr>
<tr>
<td>DROP_UNUSED Procedure</td>
<td>Drops outlines that have never been applied in the compilation of a SQL statement</td>
</tr>
<tr>
<td>EXACT_TEXT_SIGNATURES Procedure</td>
<td>Updates outline signatures to those that compute based on exact text matching</td>
</tr>
<tr>
<td>UPDATE_BY_CAT Procedure</td>
<td>Changes the category of outlines in one category to a new category</td>
</tr>
<tr>
<td>UPDATE_SIGNATURES Procedure</td>
<td>Updates outline signatures to the current version's signature</td>
</tr>
</tbody>
</table>

111.3.1 CLEAR_USED Procedure

This procedure clears the outline 'used' flag.

Syntax

```sql
DBMS_OUTLN.CLEAR_USED (name   IN   VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the outline.</td>
</tr>
</tbody>
</table>
111.3.2 CREATE_OUTLINE Procedure

This procedure generates an outline by reparsing the SQL statement from the shared cursor identified by hash value and child number.

Syntax

```
DBMS_OUTLN.CREATE_OUTLINE (
    hash_value    IN NUMBER,
    child_number  IN NUMBER,
    category      IN VARCHAR2 DEFAULT 'DEFAULT');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash_value</td>
<td>Hash value identifying the target shared cursor.</td>
</tr>
<tr>
<td>child_number</td>
<td>Child number of the target shared cursor.</td>
</tr>
<tr>
<td>category</td>
<td>Category in which to create outline (optional).</td>
</tr>
</tbody>
</table>

111.3.3 DROP_BY_CAT Procedure

This procedure drops outlines that belong to a particular category. While outlines are put into the DEFAULT category unless otherwise specified, users have the option of grouping their outlines into groups called categories.

Syntax

```
DBMS_OUTLN.DROP_BY_CAT (cat VARCHAR2);`
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>Category of outlines to drop.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure purges a category of outlines in a single call.

Examples

This example drops all outlines in the DEFAULT category:

```
DBMS_OUTLN.DROP_BY_CAT('DEFAULT');
```
111.3.4 DROP_UNUSED Procedure

This procedure drops outlines that have never been applied in the compilation of a SQL statement.

Syntax

DBMS_OUTLN.DROP_UNUSED;

Usage Notes

You can use DROP_UNUSED for outlines generated by an application for one-time use SQL statements created as a result of dynamic SQL. These outlines are never used and take up valuable disk space.

111.3.5 EXACT_TEXT_SIGNATURES Procedure

This procedure updates outline signatures to those that compute based on exact text matching.

Syntax

DBMS_OUTLN.EXACT_TEXT_SIGNATURES;

Usage Notes

This procedure is relevant only for downgrading an outline to 8.1.6 or earlier.

111.3.6 UPDATE_BY_CAT Procedure

This procedure changes the category of all outlines in one category to a new category.

Syntax

DBMS_OUTLN.UPDATE_BY_CAT (oldcat VARCHAR2 default 'DEFAULT', newcat VARCHAR2 default 'DEFAULT');

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldcat</td>
<td>The current category of outlines.</td>
</tr>
<tr>
<td>newcat</td>
<td>The new category of outlines.</td>
</tr>
</tbody>
</table>

111.3.7 UPDATE_SIGNATURES Procedure

This procedure updates outline signatures to the current version's signature.

Syntax

DBMS_OUTLN.UPDATE_SIGNATURES;
Usage Notes

You should execute this procedure if you have imported outlines generated in an earlier release to ensure that the signatures are compatible with the current release's computation algorithm.
The `DBMS_OUTPUT` package enables you to send messages from stored procedures, packages, and triggers. The package is especially useful for displaying PL/SQL debugging information.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Exceptions
- Rules and Limits
- Examples
- Data Structures
- Summary of DBMS_OUTPUT Subprograms

### 112.1 DBMS_OUTPUT Overview

The package is typically used for debugging, or for displaying messages and reports to SQL*DBA or SQL*Plus (such as are produced by applying the SQL command `DESCRIBE` to procedures).

The `PUT` Procedure and `PUT_LINE` Procedure in this package enable you to place information in a buffer that can be read by another trigger, procedure, or package. In a separate PL/SQL procedure or anonymous block, you can display the buffered information by calling the `GET_LINE` Procedure and `GET_LINES` Procedure.

If the package is disabled, all calls to subprograms are ignored. In this way, you can design your application so that subprograms are available only when a client is able to process the information.

### 112.2 DBMS_OUTPUT Security Model

The `dbmsotpt.sql` script must be run as user SYS. This creates the public synonym `DBMS_OUTPUT`, and `EXECUTE` permission on this package is granted to public.

### 112.3 DBMS_OUTPUT Operational Notes

The following operational notes apply to `DBMS_OUTPUT`.

- If you do not call `GET_LINE`, or if you do not display the messages on your screen in SQL*Plus, the buffered messages are ignored.
- SQL*Plus calls `GET_LINES` after issuing a SQL statement or anonymous PL/SQL calls.
• Typing `SET SERVEROUTPUT ON` in SQL*Plus has the effect of invoking
  
  `DBMS_OUTPUT.ENABLE (buffer_size => NULL);`

  with no limit on the output.

• You should generally avoid having application code invoke either the `DISABLE Procedure` or `ENABLE Procedure` because this could subvert the attempt of an external tool like SQL*Plus to control whether or not to display output.

  **Note:**

  Messages sent using `DBMS_OUTPUT` are not actually sent until the sending subprogram or trigger completes. There is no mechanism to flush output during the execution of a procedure.

### 112.4 DBMS_OUTPUT Exceptions

`DBMS_OUTPUT` subprograms raise the application error `ORA-20000` and return errors.

The output procedures can return the following errors:

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORU-10027</td>
<td>Buffer overflow</td>
</tr>
<tr>
<td>ORU-10028</td>
<td>Line length overflow</td>
</tr>
</tbody>
</table>

### 112.5 DBMS_OUTPUT Rules and Limits

The following are limits on `DBMS_OUTPUT` line and buffer size.

- The maximum line size is 32767 bytes.
- The default buffer size is 20000 bytes. The minimum size is 2000 bytes and the maximum is unlimited.

### 112.6 DBMS_OUTPUT Examples

This topic contains three examples of using `DBMS_OUTPUT`.

**Example 1: Using a Trigger to Produce Output**

You can use a trigger to print out some output from the debugging process. For example, you could code the trigger to invoke:

```sql
DBMS_OUTPUT.PUT_LINE('I got here:"'||:new.col||'" is the new value');
```

If you have enabled the `DBMS_OUTPUT` package, then the text produced by this `PUT_LINE` would be buffered, and you could, after executing the statement (presumably
some INSERT, DELETE, or UPDATE that caused the trigger to fire), retrieve the line of information. For example:

BEGIN
    DBMS_OUTPUT.GET_LINE(:buffer, :status);
END;

You could then optionally display the buffer on the screen. You repeat calls to GET_LINE until status comes back as nonzero. For better performance, you should use calls to GET_LINESProcedure which can return an array of lines.

Example 2: Debugging Stored Procedures and Triggers

The DBMS_OUTPUT package is commonly used to debug stored procedures and triggers. This package can also be used to enable you to retrieve information about an object and format this output, as shown in "Example 3: Retrieving Information About an Object".

This function queries the employee table and returns the total salary for a specified department. The function includes several calls to the PUT_LINE procedure:

CREATE FUNCTION dept_salary (dnum NUMBER) RETURN NUMBER IS
    CURSOR emp_cursor IS
        SELECT sal, comm FROM emp WHERE deptno = dnum;
    total_wages    NUMBER(11, 2) := 0;
    counter        NUMBER(10) := 1;
BEGIN
    FOR emp_record IN emp_cursor LOOP
        emp_record.comm := NVL(emp_record.comm, 0);
        total_wages := total_wages + emp_record.sal
                       + emp_record.comm;
        DBMS_OUTPUT.PUT_LINE('Loop number = ' || counter ||
            '; Wages = '|| TO_CHAR(total_wages)); /* Debug line */
        counter := counter + 1; /* Increment debug counter */
    END LOOP;
    /* Debug line */
    DBMS_OUTPUT.PUT_LINE('Total wages = ' || TO_CHAR(total_wages));
    RETURN total_wages;
END dept_salary;

Assume the EMP table contains the following rows:

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1002</td>
<td>1500</td>
<td>500</td>
<td>20</td>
</tr>
<tr>
<td>1203</td>
<td>1000</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>1289</td>
<td>1000</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>1347</td>
<td>1000</td>
<td>250</td>
<td>20</td>
</tr>
</tbody>
</table>

Assume the user executes the following statements in SQL*Plus:

SET SERVEROUTPUT ON
VARIABLE salary NUMBER;
EXECUTE :salary := dept_salary(20);

The user would then see the following information displayed in the output pane:

Loop number = 1; Wages = 2000
Loop number = 2; Wages = 3250
Example 3: Retrieving Information About an Object

In this example, the user has used the EXPLAIN PLAN command to retrieve information about the execution plan for a statement and has stored it in PLAN_TABLE. The user has also assigned a statement ID to this statement. The example EXPLAIN_OUT procedure retrieves the information from this table and formats the output in a nested manner that more closely depicts the order of steps undergone in processing the SQL statement.

```sql
CREATE OR REPLACE PROCEDURE explain_out
  (statement_id IN VARCHAR2) AS
  CURSOR explain_rows IS
    SELECT level, id, position, operation, options,
           object_name
    FROM plan_table
    WHERE statement_id = explain_out.statement_id
    CONNECT BY PRIOR id = parent_id
          AND statement_id = explain_out.statement_id
    START WITH id = 0
    ORDER BY id;

  BEGIN
    FOR line IN explain_rows LOOP
      IF line.id = 0 THEN
        DBMS_OUTPUT.PUT_LINE ('Plan for statement ' || statement_id || ', estimated cost = ' || line.position);
      END IF;
      DBMS_OUTPUT.PUT_LINE (lpad(' ',2*(line.level-1)) || line.operation || ' ' || line.options || ' ' || line.object_name);
    END LOOP;
  END;
```

Total wages = 3250

PL/SQL procedure successfully executed.
112.7 DBMS_OUTPUT Data Structures

The DBMS_OUTPUT package declares 2 collection types for use with the GET_LINES Procedure.

TABLE Types
CHARARR Table Type

OBJECT Types
DBMSOUTPUT_LINESARRAY Object Type

Related Topics
• GET_LINES Procedure
  This procedure retrieves an array of lines from the buffer.

112.7.1 CHARARR Table Type

This package type is to be used with the GET_LINES Procedure to obtain text submitted through the PUT Procedure and PUT_LINE Procedure.

Syntax

```sql
TYPE CHARARR IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

Related Topics
• GET_LINES Procedure
  This procedure retrieves an array of lines from the buffer.
• PUT Procedure
  This procedure places a partial line in the buffer.
• PUT_LINE Procedure
  This procedure places a line in the buffer.

112.7.2 DBMS_OUTPUT DBMSOUTPUT_LINESARRAY Object Type

This type, defined outside the package, is to be used with the GET_LINES Procedure to obtain text submitted through the PUT Procedure and PUT_LINE Procedure.

Syntax

```sql
TYPE DBMSOUTPUT_LINESARRAY IS
  VARRAY(2147483647) OF VARCHAR2(32767);
```

Related Topics
• GET_LINES Procedure
  This procedure retrieves an array of lines from the buffer.
• **PUT Procedure**
  This procedure places a partial line in the buffer.

• **PUT_LINE Procedure**
  This procedure places a line in the buffer.

112.8 Summary of DBMS_OUTPUT Subprograms

This table lists the **DBMS_OUTPUT** subprograms and briefly describes them.

**Table 112-2  DBMS_OUTPUT Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE Procedure</td>
<td>Disables message output</td>
</tr>
<tr>
<td>ENABLE Procedure</td>
<td>Enables message output</td>
</tr>
<tr>
<td>GET_LINE Procedure</td>
<td>Retrieves one line from buffer</td>
</tr>
<tr>
<td>GET_LINES Procedure</td>
<td>Retrieves an array of lines from buffer</td>
</tr>
<tr>
<td>NEW_LINE Procedure</td>
<td>Terminates a line created with <strong>PUT</strong></td>
</tr>
<tr>
<td>PUT Procedure</td>
<td>Places a partial line in the buffer</td>
</tr>
<tr>
<td>PUT_LINE Procedure</td>
<td>Places line in buffer</td>
</tr>
</tbody>
</table>

**Note:**
The **PUT Procedure** that take a number are obsolete and, while currently supported, are included in this release for legacy reasons only.

112.8.1 DISABLE Procedure

This procedure disables calls to **PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES**, and purges the buffer of any remaining information.

As with the **ENABLE Procedure**, you do not need to call this procedure if you are using the **SERVEROUTPUT** option of SQL*Plus.

**Syntax**

```plaintext
DBMS_OUTPUT.DISABLE;
```

**Pragmas**

```plaintext
pragma restrict_references(disable,WNDS,RNDS);
```

112.8.2 ENABLE Procedure

This procedure enables calls to **PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES**. Calls to these procedures are ignored if the **DBMS_OUTPUT** package is not activated.
Syntax

```sql
DBMS_OUTPUT.ENABLE (    buffer_size IN INTEGER DEFAULT 20000);
```

Pragmas

```sql
pragma restrict_references(enable,WNDS,RNDS);
```

Parameters

Table 112-3  ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_size</td>
<td>Upper limit, in bytes, the amount of buffered information. Setting buffer_size to NULL specifies that there should be no limit.</td>
</tr>
</tbody>
</table>

Usage Notes

- It is not necessary to call this procedure when you use the `SET SERVEROUTPUT` option of SQL*Plus.
- If there are multiple calls to `ENABLE`, then `buffer_size` is the last of the values specified. The maximum size is 1,000,000, and the minimum is 2,000 when the user specifies `buffer_size` (NOT NULL).
- `NULL` is expected to be the usual choice. The default is 20,000 for backwards compatibility with earlier database versions that did not support unlimited buffering.

### 112.8.3 GET_LINE Procedure

This procedure retrieves a single line of buffered information.

Syntax

```sql
DBMS_OUTPUT.GET_LINE (    line OUT VARCHAR2,    status OUT INTEGER);
```

Parameters

Table 112-4  GET_LINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>Returns a single line of buffered information, excluding a final newline character. You should declare the actual for this parameter as VARCHAR2 (32767) to avoid the risk of &quot;ORA-06502: PL/SQL: numeric or value error: character string buffer too small&quot;.</td>
</tr>
<tr>
<td>status</td>
<td>If the call completes successfully, then the status returns as 0. If there are no more lines in the buffer, then the status is 1.</td>
</tr>
</tbody>
</table>
Usage Notes

• You can choose to retrieve from the buffer a single line or an array of lines. Call the `GET_LINE` procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the `GET_LINES` procedure to retrieve an array of lines from the buffer.

• You can choose to automatically display this information if you are using SQL*Plus by using the special `SET SERVEROUTPUT ON` command.

• After calling `GET_LINE` or `GET_LINES`, any lines not retrieved before the next call to `PUT`, `PUT_LINE`, or `NEW_LINE` are discarded to avoid confusing them with the next message.

112.8.4 GET_LINES Procedure

This procedure retrieves an array of lines from the buffer.

Syntax

```sql
DBMS_OUTPUT.GET_LINES (lines OUT CHARARR,
numlines IN OUT INTEGER);
```

```sql
DBMS_OUTPUT.GET_LINES (lines OUT DBMSOUTPUT_LINESARRAY,
numlines IN OUT INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lines</td>
<td>Returns an array of lines of buffered information. The maximum length of each line in the array is 32767 bytes. It is recommended that you use the VARRAY overload version in a 3GL host program to execute the procedure from a PL/SQL anonymous block.</td>
</tr>
<tr>
<td>numlines</td>
<td>Number of lines you want to retrieve from the buffer. After retrieving the specified number of lines, the procedure returns the number of lines actually retrieved. If this number is less than the number of lines requested, then there are no more lines in the buffer.</td>
</tr>
</tbody>
</table>

Usage Notes

• You can choose to retrieve from the buffer a single line or an array of lines. Call the `GET_LINE` procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the `GET_LINES` procedure to retrieve an array of lines from the buffer.

• You can choose to automatically display this information if you are using SQL*Plus by using the special `SET SERVEROUTPUT ON` command.
• After calling GET_LINE or GET_LINES, any lines not retrieved before the next call to PUT, PUT_LINE, or NEW_LINE are discarded to avoid confusing them with the next message.

112.8.5 NEW_LINE Procedure

This procedure puts an end-of-line marker.

The GET_LINE Procedure and the GET_LINES Procedure return "lines" as delimited by "newlines". Every call to the PUT_LINE Procedure or NEW_LINE Procedure generates a line that is returned by GET_LINE(s).

Syntax

DBMS_OUTPUT.NEW_LINE;

112.8.6 PUT Procedure

This procedure places a partial line in the buffer.

**Note:**
The PUT procedure that takes a NUMBER is obsolete and, while currently supported, is included in this release for legacy reasons only.

Syntax

DBMS_OUTPUT.PUT (  
  item IN VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Item to buffer.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000, ORU-10027:</td>
<td>Buffer overflow, limit of &lt;buf_limit&gt; bytes.</td>
</tr>
<tr>
<td>ORA-20000, ORU-10028:</td>
<td>Line length overflow, limit of 32767 bytes for each line.</td>
</tr>
</tbody>
</table>

Usage Notes

• You can build a line of information piece by piece by making multiple calls to PUT, or place an entire line of information into the buffer by calling PUT_LINE.
When you call **PUT_LINE** the item you specify is automatically followed by an end-of-line marker. If you make calls to **PUT** to build a line, then you must add your own end-of-line marker by calling **NEW_LINE**. **GET_LINE** and **GET_LINES** do not return lines that have not been terminated with a newline character.

If your lines exceed the line limit, you receive an error message.

Output that you create using **PUT** or **PUT_LINE** is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

For example, SQL*Plus does not display **DBMS_OUTPUT** messages until the PL/SQL program completes. There is no mechanism for flushing the **DBMS_OUTPUT** buffers within the PL/SQL program.

```sql
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE ('hello');
  3  DBMS_LOCK.SLEEP (10);
  4  END;
```

112.8.7 **PUT_LINE** Procedure

This procedure places a line in the buffer.

**Note:**

The **PUT_LINE** procedure that takes a **NUMBER** is obsolete and, while currently supported, is included in this release for legacy reasons only.

**Syntax**

```sql
DBMS_OUTPUT.PUT_LINE (item IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Item to buffer.</td>
</tr>
</tbody>
</table>

**Exceptions**

**Table 112-9** **PUT_LINE** Procedure Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000, ORU-10027:</td>
<td>Buffer overflow, limit of &lt;buf_limit&gt; bytes.</td>
</tr>
<tr>
<td>ORA-20000, ORU-10028:</td>
<td>Line length overflow, limit of 32767 bytes for each line.</td>
</tr>
</tbody>
</table>
Usage Notes

- You can build a line of information piece by piece by making multiple calls to PUT, or place an entire line of information into the buffer by calling PUT_LINE.

- When you call PUT_LINE the item you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW_LINE. GET_LINE and GET_LINES do not return lines that have not been terminated with a newline character.

- If your lines exceeds the line limit, you receive an error message.

- Output that you create using PUT or PUT_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

For example, SQL*Plus does not display DBMS_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS_OUTPUT buffers within the PL/SQL program. For example:

```sql
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2   DBMS_OUTPUT.PUT_LINE ('hello');
  3   DBMS_LOCK.SLEEP (10);
  4 END;
```
The **DBMS_PARALLEL_EXECUTE** package enables incremental update of table data in parallel.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Constants**
- **Views**
- **Exceptions**
- **Examples**
- **Summary of DBMS_PARALLEL_EXECUTE Subprograms**

† See Also:

- *Oracle Database Development Guide*
- *Oracle Database Reference*

### 113.1 DBMS_PARALLEL_EXECUTE Overview

This package lets you incrementally update table data in parallel, in two high-level steps.

1. Group sets of rows in the table into smaller-sized chunks.
2. Run a user-specified statement on these chunks in parallel, and commit when finished processing each chunk.

This package introduces the notion of *parallel execution task*. This task groups the various steps associated with the parallel execution of a PL/SQL block, which is typically updating table data.

All of the package subroutines (except the `GENERATE_TASK_NAME` Function and the `TASK_STATUS` Procedure) perform a commit.

### 113.2 DBMS_PARALLEL_EXECUTE Security Model

**DBMS_PARALLEL_EXECUTE** is a `SYS`-owned package which is granted to `PUBLIC`.

Users who have the `ADM_PARALLEL_EXECUTE_TASK` role can perform administrative routines (qualified by the prefix `ADM`) and access the DBA view.

Apart from the administrative routines, all the subprograms refer to tasks owned by the current user.
To execute chunks in parallel, you must have `CREATE JOB` system privilege.

The `CHUNK_BY_SQL`, `RUN_TASK`, and `RESUME_TASK` subprograms require a query, and are executed using `DBMS_SQL`. Invokers of the `DBMS_SQL` interface must ensure that no query contains SQL injection.

### 113.3 DBMS_PARALLEL_EXECUTE Constants

The `DBMS_PARALLEL_EXECUTE` package uses the constants described in these two tables.

#### Table 113-1  DBMS_PARALLEL_EXECUTE Constants - Chunk Status Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNED</td>
<td>NUMBER</td>
<td>1</td>
<td>Chunk has been assigned for processing</td>
</tr>
<tr>
<td>PROCESSED</td>
<td>NUMBER</td>
<td>2</td>
<td>Chunk has been processed successfully</td>
</tr>
<tr>
<td>PROCESSED_WITH_ERROR</td>
<td>NUMBER</td>
<td>3</td>
<td>Chunk has been processed, but an error occurred during processing</td>
</tr>
<tr>
<td>UNASSIGNED</td>
<td>NUMBER</td>
<td>0</td>
<td>Chunk is unassigned</td>
</tr>
</tbody>
</table>

#### Table 113-2  DBMS_PARALLEL_EXECUTE Constants - Task Status Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHUNKED</td>
<td>NUMBER</td>
<td>4</td>
<td>Table associated with the task has been chunked, but none of the chunk has been assigned for processing</td>
</tr>
<tr>
<td>CHUNKING</td>
<td>NUMBER</td>
<td>2</td>
<td>Table associated with the task is being chunked</td>
</tr>
<tr>
<td>CHUNKING_FAILED</td>
<td>NUMBER</td>
<td>3</td>
<td>Chunking failed</td>
</tr>
<tr>
<td>CRASHED</td>
<td>NUMBER</td>
<td>8</td>
<td>Only applicable if parallel execution is used, this occurs if a job slave crashes or if the database crashes during <code>EXECUTE</code>, leaving a chunk in <code>ASSIGNED</code> or <code>UNASSIGNED</code> state.</td>
</tr>
<tr>
<td>CREATED</td>
<td>NUMBER</td>
<td>1</td>
<td>The task has been created by the <code>CREATE_TASK Procedure</code></td>
</tr>
<tr>
<td>FINISHED</td>
<td>NUMBER</td>
<td>6</td>
<td>All chunks processed without error</td>
</tr>
<tr>
<td>FINISHED_WITH_ERROR</td>
<td>NUMBER</td>
<td>7</td>
<td>All chunks processed, but with errors in some cases</td>
</tr>
<tr>
<td>PROCESSING</td>
<td>NUMBER</td>
<td>5</td>
<td>Part of the chunk assigned for processing, or which has been processed</td>
</tr>
</tbody>
</table>

### 113.4 DBMS_PARALLEL_EXECUTE Views

The `DBMS_PARALLEL_EXECUTE` package uses the following views.

- `DBA_PARALLEL_EXECUTE_CHUNKS`
- `DBA_PARALLEL_EXECUTE_TASKS`
113.5 DBMS_PARALLEL_EXECUTE Exceptions

The following table lists the exceptions raised by DBMS_PARALLEL_EXECUTE.

Table 113-3  Exceptions Raised by DBMS_PARALLEL_EXECUTE

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHUNK_NOT_FOUND</td>
<td>29499</td>
<td>Specified chunk does not exist</td>
</tr>
<tr>
<td>DUPLICATE_TASK_NAME</td>
<td>29497</td>
<td>Same task name has been used by an existing task</td>
</tr>
<tr>
<td>INVALID_STATE_FOR_CHUNK</td>
<td>29492</td>
<td>Attempts to chunk a table that is not in CREATED or CHUNKING_FAILED state</td>
</tr>
<tr>
<td>INVALID_STATE_FOR_REDSUME</td>
<td>29495</td>
<td>Attempts to resume execution, but the task is not in FINISHED_WITH_ERROR or CRASHED state</td>
</tr>
<tr>
<td>INVALID_STATE_FOR_RUN</td>
<td>29494</td>
<td>Attempts to execute the task that is not in CHUNKED state</td>
</tr>
<tr>
<td>INVALID_STATUS</td>
<td>29493</td>
<td>Attempts to set an invalid value to the chunk status</td>
</tr>
<tr>
<td>INVALID_TABLE</td>
<td>29491</td>
<td>Attempts to chunk a table by rowid in cases in which the table is not a physical table, or the table is an IOT</td>
</tr>
<tr>
<td>MISSING_ROLE</td>
<td>29490</td>
<td>User does not have the necessary ADM_PARALLEL_EXECUTE role</td>
</tr>
<tr>
<td>TASK_NOT_FOUND</td>
<td>29498</td>
<td>Specified task name does not exist</td>
</tr>
</tbody>
</table>

113.6 DBMS_PARALLEL_EXECUTE Examples

The following examples run on the Human Resources (HR) schema of the Oracle Database Sample Schemas. They require that the HR schema be created with the JOB SYSTEM privilege.

**Chunk by ROWID**

This example shows the most common usage of this package. After calling the RUN_TASK Procedure, it checks for errors and reruns in the case of error.

```sql
DECLARE
  l_sql_stmt VARCHAR2(1000);
  l_try NUMBER;
  l_status NUMBER;
BEGIN

  -- Create the TASK
  DBMS_PARALLEL_EXECUTE.CREATE_TASK ('mytask');

  -- Chunk the table by ROWID
  DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID('mytask', 'HR', 'EMPLOYEES', true, 100);

  -- Execute the DML in parallel
  l_sql_stmt := 'update EMPLOYEES e
    SET e.salary = e.salary + 10
    WHERE /* other conditions */';

END;
```
WHERE rowid BETWEEN :start_id AND :end_id;
DBMS_PARALLEL_EXECUTE.RUN_TASK('mytask', l_sql_stmt, DBMS_SQL.NATIVE,
   parallel_level => 10);

-- If there is an error, RESUME it for at most 2 times.
L_try := 0;
L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
WHILE(l_try < 2 and L_status != DBMS_PARALLEL_EXECUTE.FINISHED)
LOOP
   L_try := l_try + 1;
   DBMS_PARALLEL_EXECUTE.RESUME_TASK('mytask');
   L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
END LOOP;

-- Done with processing; drop the task
DBMS_PARALLEL_EXECUTE.DROP_TASK('mytask');

END;
/

Chunk by User-Provided SQL

A user can specify a chunk algorithm by using the CREATE_CHUNKS_BY_SQL Procedure. This example shows that rows with the same manager_id are grouped together and processed in one chunk.

DECLARE
   l_chunk_sql VARCHAR2(1000);
   l_sql_stmt VARCHAR2(1000);
   l_try NUMBER;
   l_status NUMBER;
BEGIN

   -- Create the TASK
   DBMS_PARALLEL_EXECUTE.CREATE_TASK ('mytask');

   -- Chunk the table by MANAGER_ID
   l_chunk_sql := 'SELECT distinct manager_id, manager_id FROM employees';
   DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_SQL('mytask', l_chunk_sql, false);

   -- Execute the DML in parallel
   -- the WHERE clause contain a condition on manager_id, which is the chunk column. In this case, grouping rows is by manager_id.
   l_sql_stmt := 'update EMPLOYEES e
      SET e.salary = e.salary + 10
      WHERE manager_id between :start_id and :end_id';
   DBMS_PARALLEL_EXECUTE.RUN_TASK('mytask', l_sql_stmt, DBMS_SQL.NATIVE,
      parallel_level => 10);

   -- If there is error, RESUME it for at most 2 times.
   L_try := 0;
   L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
   WHILE(l_try < 2 and L_status != DBMS_PARALLEL_EXECUTE.FINISHED)
      LOOP
         L_try := l_try + 1;
         DBMS_PARALLEL_EXECUTE.RESUME_TASK('mytask');
         L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
      END LOOP;

   -- Done with processing; drop the task
   DBMS_PARALLEL_EXECUTE.DROP_TASK('mytask');

END;
Executing Chunks in an User-defined Framework

You can execute chunks in a self-defined framework without using the RUN_TASK Procedure. This example shows how to use GET_ROWID_CHUNK Procedure, EXECUTE IMMEDIATE, SET_CHUNK_STATUS Procedure to execute the chunks.

DECLARE
  l_sql_stmt varchar2(1000);
  l_try number;
  l_status number;
  l_chunk_id number;
  l_start_rowid rowid;
  l_end_rowid rowid;
  l_any_rows boolean;
CURSOR c1 IS SELECT chunk_id
  FROM user_parallel_execute_chunks
  WHERE task_name = 'mytask'
  AND STATUS IN ('DBMS_PARALLEL_EXECUTE.PROCESSED_WITH_ERROR',
                  'DBMS_PARALLEL_EXECUTE.ASSIGNED');
BEGIN
  -- Create the Objects, task, and chunk by ROWID
  DBMS_PARALLEL_EXECUTE.CREATE_TASK ('mytask');
  DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID('mytask', 'HR', 'EMPLOYEES', true, 100);
  l_sql_stmt := 'update EMPLOYEES e
                  SET e.salary = e.salary + 10
                  WHERE rowid BETWEEN :start_id AND :end_id';
  -- Execute the DML in his own framework
  --
  -- Process each chunk and commit.
  -- After processing one chunk, repeat this process until
  -- all the chunks are processed.
  --
  <<main_processing>>
  LOOP
    -- Get a chunk to process; if there is nothing to process, then exit the
    -- loop;
    --
    DBMS_PARALLEL_EXECUTE.GET_ROWID_CHUNK('mytask',
                                           l_chunk_id,
                                           l_start_rowid,
                                           l_end_rowid,
                                           l_any_rows);
    IF (l_any_rows = false) THEN EXIT; END IF;
    --
    -- The chunk is specified by start_id and end_id.
    -- Bind the start_id and end_id and then execute it
    --
    -- If no error occured, set the chunk status to PROCESSED.
    --
    -- Catch any exception. If an exception occured, store the error num/msg
    -- into the chunk table and then continue to process the next chunk.
    --
BEGIN
  EXECUTE IMMEDIATE l_sql_stmt using l_start_rowid, l_end_rowid;
  DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS('mytask', l_chunk_id,
      DBMS_PARALLEL_EXECUTE.PROCESSED);
EXCEPTION WHEN OTHERS THEN
  DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS('mytask', l_chunk_id,
      DBMS_PARALLEL_EXECUTE.PROCESSED_WITH_ERROR, SQLCODE, SQLERRM);
END;

--
-- Finished processing one chunk; Commit here
--
    COMMIT;
END LOOP;

113.7 Summary of DBMS_PARALLEL_EXECUTE Subprograms

This table lists the DBMS_PARALLEL_EXECUTE subprograms and briefly describes them.

Table 113-4  DBMS_PARALLEL_EXECUTE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_DROP_CHUNKS Procedure</td>
<td>Drops all chunks of the specified task owned by the specified owner</td>
</tr>
<tr>
<td>ADM_DROP_TASK Procedure</td>
<td>Drops the task of the given user and all related chunks</td>
</tr>
<tr>
<td>ADM_TASK_STATUS Function</td>
<td>Returns the task status</td>
</tr>
<tr>
<td>ADM_STOP_TASK Procedure</td>
<td>Stops the task of the given owner and related job slaves</td>
</tr>
<tr>
<td>CREATE_TASK Procedure</td>
<td>Creates a task for the current user</td>
</tr>
<tr>
<td>CREATE_CHUNKS_BY_NUMBER_COLUMN Procedure</td>
<td>Chunks the table associated with the given task by the specified column.</td>
</tr>
<tr>
<td>CREATE_CHUNKS_BY_ROWID Procedure</td>
<td>Chunks the table associated with the given task by ROWID</td>
</tr>
<tr>
<td>CREATE_CHUNKS_BY_SQL Procedure</td>
<td>Chunks the table associated with the given task by means of a user-provided SELECT statement</td>
</tr>
<tr>
<td>DROP_TASK Procedure</td>
<td>Drops the task and all related chunks</td>
</tr>
<tr>
<td>DROP_CHUNKS Procedure</td>
<td>Drops the task's chunks</td>
</tr>
<tr>
<td>GENERATE_TASK_NAME Function</td>
<td>Returns a unique name for a task</td>
</tr>
<tr>
<td>GET_NUMBER_COL_CHUNK Procedure</td>
<td>Picks an unassigned NUMBER chunk and changes it to ASSIGNED</td>
</tr>
<tr>
<td>GET_ROWID_CHUNK Procedure</td>
<td>Picks an unassigned ROWID chunk and changes it to ASSIGNED</td>
</tr>
<tr>
<td>PURGE_PROCESSED_CHUNKS Procedure</td>
<td>Deletes all the processed chunks whose status is PROCESSED or PROCESSED_WITH_ERROR</td>
</tr>
<tr>
<td>RESUME_TASK Procedures</td>
<td>Retries the given the task if the RUN_TASK Procedure finished with an error, or resumes the task if a crash occurred.</td>
</tr>
</tbody>
</table>
Table 113-4 (Cont.) DBMS_PARALLEL_EXECUTE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_TASK Procedure</td>
<td>Executes the specified SQL statement on the chunks in parallel</td>
</tr>
<tr>
<td>SET_CHUNK_STATUS Procedure</td>
<td>Sets the status of the chunk</td>
</tr>
<tr>
<td>STOP_TASK Procedure</td>
<td>Stops the task and related job slaves</td>
</tr>
<tr>
<td>TASK_STATUS Procedure</td>
<td>Returns the task status</td>
</tr>
</tbody>
</table>

113.7.1 ADM_DROP_CHUNKS Procedure

This procedure drops all chunks of the specified task owned by the specified owner.

Syntax

```sql
DBMS_PARALLEL_EXECUTE.ADM_DROP_CHUNKS (  
    task_owner      IN  VARCHAR2,  
    task_name       IN  VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_owner</td>
<td>Owner of the task</td>
</tr>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

113.7.2 ADM_DROP_TASK Procedure

This procedure drops the task of the specified user and all related chunks.

Syntax

```sql
DBMS_PARALLEL_EXECUTE.ADM_DROP_TASK (  
    task_owner      IN  VARCHAR2,  
    task_name       IN  VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_owner</td>
<td>Owner of the task</td>
</tr>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>
113.7.3 ADM_TASK_STATUS Function

This function returns the task status.

Syntax

```
DBMS_PARALLEL_EXECUTE.ADM_TASK_STATUS  (
   task_owner      IN  VARCHAR2,
   task_name       IN  VARCHAR2)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_owner</td>
<td>Owner of the task</td>
</tr>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

113.7.4 ADM_STOP_TASK Procedure

This procedure stops the task of the specified owner and related job slaves.

Syntax

```
DBMS_PARALLEL_EXECUTE.ADM_STOP_TASK  (
   task_owner      IN  VARCHAR2,
   task_name       IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_owner</td>
<td>Owner of the task</td>
</tr>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

113.7.5 CREATE_TASK Procedure

This procedure creates a task for the current user. The pairing of `task_name` and `current_user` must be unique.

Syntax

```
DBMS_PARALLEL_EXECUTE.CREATE_TASK  (
   task_name        IN   VARCHAR2,
   comment          IN   VARCHAR2 DEFAULT NULL);
```
Parameters

### Table 113-9  CREATE_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task. The task_name can be any string in which related length must be less than or equal to 128 bytes.</td>
</tr>
<tr>
<td>comment</td>
<td>Comment field. The comment must be less than 4000 bytes.</td>
</tr>
</tbody>
</table>

### 113.7.6 CREATE_CHUNKS_BY_NUMBER_COL Procedure

This procedure chunks the table (associated with the specified task) by the specified column. The specified column must be a `NUMBER` column. This procedure takes the `MIN` and `MAX` value of the column, and then divides the range evenly according to `chunk_size`.

The chunks are:

<table>
<thead>
<tr>
<th>START_ID</th>
<th>END_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_id_val</td>
<td>min_id_val+1*chunk_size-1</td>
</tr>
<tr>
<td>min_id_val+1*chunk_size</td>
<td>min_id_val+2*chunk_size-1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>min_id_val+i*chunk_size</td>
<td>max_id_val</td>
</tr>
</tbody>
</table>

### Syntax

```sql
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_NUMBER_COL (
    task_name       IN  VARCHAR2,
    table_owner     IN  VARCHAR2,
    table_name      IN  VARCHAR2,
    table_column    IN  VARCHAR2,
    chunk_size      IN  NUMBER);
```

Parameters

### Table 113-10  CREATE_CHUNKS_BY_NUMBER_COL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>table_owner</td>
<td>Owner of the table</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table</td>
</tr>
<tr>
<td>table_column</td>
<td>Name of the <code>NUMBER</code> column</td>
</tr>
<tr>
<td>chunk_size</td>
<td>Range of each chunk</td>
</tr>
</tbody>
</table>

### 113.7.7 CREATE_CHUNKS_BY_ROWID Procedure

This procedure chunks the table (associated with the specified task) by `ROWID`.

`num_row` and `num_block` are approximate guidance for the size of each chunk. The table to be chunked must be a physical table with physical `ROWID` having views and table functions. Index-organized tables are not allowed.
Syntax

```sql
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID (
  task_name       IN  VARCHAR2,
  table_owner     IN  VARCHAR2,
  table_name      IN  VARCHAR2,
  by_row          IN  BOOLEAN,
  chunk_size      IN  NUMBER);
```

Parameters

**Table 113-11  CREATE_CHUNKS_BY_ROWID Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>table_owner</td>
<td>Owner of the table</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table</td>
</tr>
<tr>
<td>by_row</td>
<td>TRUE if chunk_size refers to the number of rows, otherwise, chunk_size refers to the number of blocks</td>
</tr>
<tr>
<td>chunk_size</td>
<td>Approximate number of rows/blocks to process for each commit cycle</td>
</tr>
</tbody>
</table>

**113.7.8 CREATE_CHUNKS_BY_SQL Procedure**

This procedure chunks the table (associated with the specified task) by means of a user-provided `SELECT` statement.

The `SELECT` statement that returns the range of each chunk must have two columns: `start_id` and `end_id`. If the task is to chunk by ROWID, then the two columns must be of ROWID type. If the task is to chunk the table by NUMBER column, then the two columns must be of NUMBER type. The procedure provides the flexibility to users who want to deploy user-defined chunk algorithms.

Syntax

```sql
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_SQL (
  task_name       IN  VARCHAR2,
  sql_statement   IN  CLOB,
  by_rowid        IN  BOOLEAN);
```

Parameters

**Table 113-12  CREATE_CHUNKS_BY_SQL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>sql_statement</td>
<td>SQL that returns the chunk ranges</td>
</tr>
<tr>
<td>by_rowid</td>
<td>TRUE if the table is chunked by rowids</td>
</tr>
</tbody>
</table>
113.7.9 DROP_TASK Procedure

This procedure drops the task and all related chunks.

Syntax

```sql
DBMS_PARALLEL_EXECUTE.DROP_TASK (  
  task_name       IN VARCHAR2);
```

Parameters

Table 113-13  DROP_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

113.7.10 DROP_CHUNKS Procedure

This procedure drops the task's chunks.

Syntax

```sql
DBMS_PARALLEL_EXECUTE.DROP_CHUNKS (  
  task_name       IN VARCHAR2);
```

Parameters

Table 113-14  DROP_CHUNKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

113.7.11 GENERATE_TASK_NAME Function

This function returns a unique name for a task.

The name is of the form `prefixN` where `N` is a number from a sequence. If no prefix is specified, the generated name is, by default, `TASK$_1, TASK$_2, TASK$_3, and so on. If 'SCOTT' is specified as the prefix, the name is `SCOTT1, SCOTT2, and so on.

Syntax

```sql
DBMS_PARALLEL_EXECUTE.GENERATE_TASK_NAME (  
  prefix       IN      VARCHAR2 DEFAULT 'TASK$')  
RETURN VARCHAR2;
```
### 113.7.12 GET_NUMBER_COL CHUNK Procedure

This procedure picks an unassigned `NUMBER` chunk and changes it to `ASSIGNED`. If there are no more chunks to assign, `any_rows` is set to `FALSE`. Otherwise, the `chunk_id`, `start`, and `end_id` of the chunk are returned as `OUT` parameters.

The chunk information in `DBMS_PARALLEL_EXECUTE_CHUNKS$` is updated as follows: `STATUS` becomes `ASSIGNED`; `START_TIMESTAMP` records the current time; `END_TIMESTAMP` is cleared.

**Syntax**

```sql
DBMS_PARALLEL_EXECUTE.GET_NUMBER_COL_CHUNK (
    task_name       IN VARCHAR2,
    chunk_id        OUT NUMBER,
    start_rowid     OUT ROWID,
    end_id          OUT ROWID,
    any_rows        OUT BOOLEAN);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>chunk_id</td>
<td>Chunk id of the chunk</td>
</tr>
<tr>
<td>start_rowid</td>
<td>Start rowid in the returned range</td>
</tr>
<tr>
<td>end_id</td>
<td>End rowid in the returned range</td>
</tr>
<tr>
<td>any_rows</td>
<td>Indicating if there could be any rows to process in the range</td>
</tr>
</tbody>
</table>

**Usage Notes**

If the task is chunked by `ROWID`, then use `get_rowid_range`. If the task is chunked by `NUMBER` column, then use `get_number_col_range`. If you make the wrong function call, the returning `chunk_id` and `any_rows` have valid values but `start_id` and `end_id` are `NULL`. 

---

**Table 113-15  GENERATE_TASK_NAME Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>The prefix to use when generating the task name</td>
</tr>
</tbody>
</table>

---

**Table 113-16  GET_NUMBER_COL_CHUNK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>chunk_id</td>
<td>Chunk id of the chunk</td>
</tr>
<tr>
<td>start_rowid</td>
<td>Start rowid in the returned range</td>
</tr>
<tr>
<td>end_id</td>
<td>End rowid in the returned range</td>
</tr>
<tr>
<td>any_rows</td>
<td>Indicating if there could be any rows to process in the range</td>
</tr>
</tbody>
</table>
113.7.13 GET_ROWID_CHUNK Procedure

This procedure picks an unassigned ROWID chunk and changes it to ASSIGNED.

If there are no more chunks to assign, any_rows is set to FALSE. Otherwise, the chunk_id, start, and end_id of the chunk are returned as OUT parameters. The chunk info in DBMS_PARALLEL_EXECUTE_CHUNKS$ is updated as follows: STATUS becomes ASSIGNED; START_TIMESTAMP records the current time; END_TIMESTAMP is cleared.

See Also:

Views

Syntax

DBMS_PARALLEL_EXECUTE.GET_ROWID_CHUNK (  
  task_name       IN VARCHAR2,  
  chunk_id        OUT NUMBER,  
  start_rowid     OUT ROWID,  
  end_id          OUT ROWID,  
  any_rows        OUT BOOLEAN);  

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>chunk_id</td>
<td>Chunk_id of the chunk</td>
</tr>
<tr>
<td>start_rowid</td>
<td>Start rowid in the returned range</td>
</tr>
<tr>
<td>end_id</td>
<td>End rowid in the returned range</td>
</tr>
<tr>
<td>any_rows</td>
<td>Indicates that the range could include rows to process</td>
</tr>
</tbody>
</table>

Usage Notes

If the task is chunked by ROWID, then use get_rowid_range. If the task is chunked by NUMBER column, then use get_number_col_range. If you make the wrong function call, the returning chunk_id and any_rows will still have valid values but start_id and end_id are NULL.

113.7.14 PURGE_PROCESSED_CHUNKS Procedure

This procedure deletes all the processed chunks whose status is PROCESSED or PROCESSED_WITH_ERROR.

Syntax

DBMS_PARALLEL_EXECUTE.PURGE_PROCESSED_CHUNKS (  
  task_name       IN VARCHAR2);
Parameters

Table 113-18 PURGE_PROCESSED_CHUNKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

113.7.15 RESUME_TASK Procedures

This procedure retries the specified task if the RUN_TASK Procedure finished with an error, or resumes the task if a crash occurred.

You can only invoke this procedure if the task is in a CRASHED or FINISHED_WITH_ERROR state.

For a crashed serial execution, the state remains in PROCESSING. The FORCE option allows you to resume any task in PROCESSING state. However, it is your responsibility to determine that a crash has occurred.

The procedure resumes processing the chunks which have not been processed. Also, chunks which are in PROCESSED_WITH_ERROR or ASSIGNED (due to crash) state are processed because those chunks did not commit.

This procedure takes the same argument as the RUN_TASK Procedure. The overload which takes task name as the only input argument re-uses the arguments provided in the previous invoking of the RUN_TASK Procedure or RESUME_TASK Procedures.

See Also:
Table 113-2

Syntax

DBMS_PARALLEL_EXECUTE.RESUME_TASK (    task_name                     IN  VARCHAR2,    sql_stmt                      IN  CLOB,    language_flag                 IN  NUMBER,    edition                       IN  VARCHAR2  DEFAULT NULL,    apply_crossedition_trigger    IN  VARCHAR2  DEFAULT NULL,    fire_apply_trigger            IN  BOOLEAN  DEFAULT TRUE,    parallel_level                IN  NUMBER  DEFAULT 0,    job_class                     IN  VARCHAR2  DEFAULT 'DEFAULT_JOB_CLASS',    force                         IN  BOOLEAN  DEFAULT FALSE);    DBMS_PARALLEL_EXECUTE.RESUME_TASK (    task_name                     IN  VARCHAR2,    force                         IN  BOOLEAN  DEFAULT FALSE);
### Parameters

#### Table 113-19  RESUME_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>sql_stmt</td>
<td>SQL statement; must have :start_id and :end_id placeholders</td>
</tr>
<tr>
<td>language_flag</td>
<td>Determines how Oracle handles the SQL statement. The following options are recognized:&lt;br&gt;· V6 (or 0) specifies version 6 behavior&lt;br&gt;· NATIVE (or 1) specifies normal behavior for the database to which the program is connected&lt;br&gt;· V7 (or 2) specifies Oracle database version 7 behavior</td>
</tr>
<tr>
<td>edition</td>
<td>Specifies the edition in which to run the statement. Default is the current edition.</td>
</tr>
<tr>
<td>apply_crossedition_trigger</td>
<td>Specifies the unqualified name of a forward crossedition trigger that is to be applied to the specified SQL. The name is resolved using the edition and current_schema setting in which the statement is to be executed. The trigger must be owned by the user who executes the statement.</td>
</tr>
<tr>
<td>fire_apply_trigger</td>
<td>Indicates whether the specified apply_crossedition_trigger is itself to be executed, or only to used as be a guide in selecting other triggers</td>
</tr>
<tr>
<td>parallel_level</td>
<td>Number of parallel jobs; zero if run in serial; NULL uses the default parallelism</td>
</tr>
<tr>
<td>job_class</td>
<td>If running in parallel, the jobs all belong to the specified job class</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, do not raise an error if the status is PROCESSING.</td>
</tr>
</tbody>
</table>

### Examples

Suppose the chunk table contains the following chunk ranges:

<table>
<thead>
<tr>
<th>START_ID</th>
<th>END_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>30</td>
</tr>
</tbody>
</table>

And the specified SQL statement is:

```sql
UPDATE employees
    SET salary = salary + 10
    WHERE e.employee_id BETWEEN :start_id AND :end_id
```

This procedure executes the following statements in parallel:

```sql
UPDATE employees
    SET salary = salary + 10 WHERE employee_id BETWEEN 1 and 10;
    COMMIT;

UPDATE employees
    SET salary = salary + 10 WHERE employee_id between 11 and 20;
    COMMIT;
```
UPDATE employees
  SET salary = salary + 10  WHERE employee_id between 21 and 30;
COMMIT;

Related Topics
• **RUN_TASK Procedure**
  This procedure executes the specified statement (sql_stmt) on the chunks in parallel.

### 113.7.16 RUN_TASK Procedure

This procedure executes the specified statement (sql_stmt) on the chunks in parallel.

It commits after processing each chunk.

The specified statement must have two placeholders called `start_id` and `end_id`, respectively, which represent the range of the chunk to be processed. The type of each placeholder must be `ROWID` where `ROWID`-based chunking was used, or `NUMBER` where `NUMBER`-based chunking was used.

**Syntax**

```sql
DBMS_PARALLEL_EXECUTE.RUN_TASK (  
  task_name                     IN  VARCHAR2,  
  sql_stmt                      IN  CLOB,  
  language_flag                 IN  NUMBER,  
  edition                       IN  VARCHAR2  DEFAULT NULL,  
  apply_crossedition_trigger    IN  VARCHAR2  DEFAULT NULL,  
  fire_apply_trigger            IN  BOOLEAN   DEFAULT TRUE,  
  parallel_level                IN  NUMBER    DEFAULT 0,  
  job_class                     IN  VARCHAR2  DEFAULT 'DEFAULT_JOB_CLASS');  
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>sql_stmt</td>
<td>SQL statement; must have <code>:start_id</code> and <code>:end_id</code> placeholders</td>
</tr>
<tr>
<td>language_flag</td>
<td>Determines how Oracle handles the SQL statement. The following options are recognized:</td>
</tr>
<tr>
<td></td>
<td>• V6 (or 0) specifies version 6 behavior</td>
</tr>
<tr>
<td></td>
<td>• NATIVE (or 1) specifies normal behavior for the database to which the program is connected</td>
</tr>
<tr>
<td></td>
<td>• V7 (or 2) specifies Oracle database version 7 behavior</td>
</tr>
<tr>
<td>edition</td>
<td>Specifies the edition in which to run the statement. Default is the current edition.</td>
</tr>
<tr>
<td>apply_crossedition_trigger</td>
<td>Specifies the unqualified name of a forward crossedition trigger that is to be applied to the specified SQL. The name is resolved using the edition and current_schema setting in which the statement is to be executed. The trigger must be owned by the user executes the statement.</td>
</tr>
</tbody>
</table>
### Table 113-20  (Cont.) RUN_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fire_apply_trigger</td>
<td>Indicates whether the specified apply_crossedition_trigger is itself to be executed, or only a guide to be used in selecting other triggers.</td>
</tr>
<tr>
<td>parallel_level</td>
<td>Number of parallel jobs; zero if run in serial; NULL uses the default parallelism.</td>
</tr>
<tr>
<td>job_class</td>
<td>If running in parallel, the jobs belong to the specified job class.</td>
</tr>
</tbody>
</table>

### Usage Notes

- The SQL statement is executed as the current user.
- Since this subprogram is subject to reexecution on error, you need to take great care in submitting a statement to `RUN_TASK` that is not idempotent.
- Chunks can be executed in parallel by `DBMS_SCHEDULER` job slaves. Therefore, parallel execution requires `CREATE JOB` system privilege. The job slaves is created under the current user. The default number of job slaves is computed as the product of Oracle parameters `cpu_count` and `parallel_threads_per_cpu`. On a Real Application Clusters installation, the number of job slaves is the sum of individual settings on each node in the cluster. This procedure returns only when all the chunks are processed. In parallel cases, this procedure returns only when all the job slaves finished.

### Examples

Suppose the chunk table contains the following chunk ranges:

<table>
<thead>
<tr>
<th>START_ID</th>
<th>END_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>30</td>
</tr>
</tbody>
</table>

And the specified SQL statement is:

```
UPDATE employees
SET salary = salary + 10
WHERE e.employee_id BETWEEN :start_id AND :end_id
```

This procedure executes the following statements in parallel:

```sql
UPDATE employees
SET salary = salary + 10 WHERE employee_id BETWEEN 1 and 10;
COMMIT;

UPDATE employees
SET salary = salary + 10 WHERE employee_id between 11 and 20;
COMMIT;

UPDATE employees
SET salary = salary + 10 WHERE employee_id between 21 and 30;
COMMIT;
```
113.7.17 SET_CHUNK_STATUS Procedure

This procedure sets the status of the chunk.

The START_TIMESTAMP and END_TIMESTAMP of the chunk is updated according to the new status:

<table>
<thead>
<tr>
<th>Value of the new Status</th>
<th>Side Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNASSIGNED</td>
<td>START_TIMESTAMP and END_TIMESTAMP will be cleared</td>
</tr>
<tr>
<td>ASSIGNED</td>
<td>START_TIMESTAMP will be the current time and END_TIMESTAMP will be cleared.</td>
</tr>
<tr>
<td>PROCESSED or PROCESSED_WITH_ERROR</td>
<td>The current time will be recorded in END_TIMESTAMP</td>
</tr>
</tbody>
</table>

See Also:

Views

Syntax

DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS (
    task_name       IN VARCHAR2,
    chunk_id        OUT NUMBER,
    status          IN  NUMBER,
    err_num         IN  NUMBER   DEFAULT NULL,
    err_msg         IN  VARCHAR2 DEFAULT NULL);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
<tr>
<td>chunk_id</td>
<td>Chunk_id of the chunk</td>
</tr>
<tr>
<td>status</td>
<td>Status of the chunk: UNASSIGNED, ASSIGNED, PROCESSED, PROCESSED_WITH_ERROR</td>
</tr>
<tr>
<td>err_num</td>
<td>Error code returned during the processing of the chunk</td>
</tr>
<tr>
<td>err_msg</td>
<td>Error message returned during the processing of the chunk</td>
</tr>
</tbody>
</table>

113.7.18 STOP_TASK Procedure

This procedure stops the task and related job slaves.

Syntax

DBMS_PARALLEL_EXECUTE.STOP_TASK ( 
    task_name       IN VARCHAR2);
Parameters

Table 113-22  STOP_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>

113.7.19 TASK_STATUS Procedure

This procedure returns the task status.

Syntax

```
DBMS_PARALLEL_EXECUTE.TASK_STATUS (
  task_name IN VARCHAR2);
```

Parameters

Table 113-23  TASK_STATUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task</td>
</tr>
</tbody>
</table>
The **DBMS_PART** package provides an interface for maintenance and management operations on partitioned objects.

**See Also:**
- *Oracle Database Reference* for related views

This chapter contains the following topics:
- Security Model
- Operational Notes
- Summary of DBMS_PART Subprograms

### 114.1 DBMS_PART Security Model

**DBMS_PART** is an invoker's rights package, running with the privileges of the user.

### 114.2 DBMS_PART Operational Notes

The following operational notes apply to DBMS_PART.
- **DBMS_PART** ignores all the errors that it runs into during the cleanup process.
- To display the message *PL/SQL procedure executed successfully* requires at least one cleanup operation to be successful.

### 114.3 Summary of DBMS_PART Subprograms

This table briefly describes the subprograms of **DBMS_PART** package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEANUP_GIDX Procedure</td>
<td>Gathers the list of global indexes where optimized asynchronous index</td>
</tr>
<tr>
<td></td>
<td>maintenance has taken place to clean up entries pointing to data segments</td>
</tr>
<tr>
<td></td>
<td>that no longer exist</td>
</tr>
<tr>
<td>CLEANUP_GIDX_JOB Procedure</td>
<td></td>
</tr>
<tr>
<td>CLEANUP_ONLINE_OP Procedure</td>
<td>Cleans up failed online move operations</td>
</tr>
</tbody>
</table>
114.3.1 CLEANUP_GIDX Procedure

As a consequence of prior partition maintenance operations with asynchronous global index maintenance, global indexes can contain entries pointing to data segments that no longer exist. These stale index rows will not cause any correctness issues or corruptions during any operation on the table or index, whether these are queries, DMLs, DDLs or analyze. This procedure will identify and cleanup these global indexes to ensure efficiency in terms of storage and performance.

Syntax

```
DBMS_PART.CLEANUP_GIDX (    
    schema_name_in    IN   VARCHAR2 DEFAULT NULL, 
    table_name_in     IN   VARCHAR2 DEFAULT NULL, 
    parallel          IN   VARCHAR2 DEFAULT NULL, 
    options           IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 114-2 CLEANUP_GIDX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name_in</td>
<td>Non-NULL processes only indexes on tables in the given schema</td>
</tr>
<tr>
<td>table_name_in</td>
<td>Non-NULL processes only indexes on the given table in the given schema</td>
</tr>
<tr>
<td>parallel</td>
<td>The parallel degree to use for the ALTER INDEX DDLs.</td>
</tr>
<tr>
<td>options</td>
<td>The following options are supported:</td>
</tr>
<tr>
<td></td>
<td>· CLEANUP_ORPHANS: implies that 'cleanup only' mechanism is used.</td>
</tr>
<tr>
<td></td>
<td>· COALESCE: implies that 'coalesce cleanup' mechanism is used.</td>
</tr>
</tbody>
</table>

114.3.2 CLEANUP_GIDX_JOB Procedure

This procedure will identify and cleanup these global indexes to ensure efficiency in terms of storage and performance.

Syntax

```
DBMS_PART.CLEANUP_GIDX_JOB (    
    parallel          IN   VARCHAR2 DEFAULT NULL, 
    options           IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 114-3 CLEANUP_GIDX_JOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>The parallel degree to use for the ALTER INDEX DDLs.</td>
</tr>
</tbody>
</table>
### 114.3.3 CLEANUP_ONLINE_OP Procedure

There are many possible points of failure when performing `ALTER TABLE ... MOVE PARTITION ... ONLINE` operations. This procedure pro-actively cleans up such failed online move operations instead of waiting for the background process (`SMON`) to do so.

**Syntax**

```sql
DBMS_PART.CLEANUP_ONLINE_OP (
    schema_name IN VARCHAR2 DEFAULT NULL,
    table_name  IN VARCHAR2 DEFAULT NULL,
    partition_name IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

**Table 114-4 CLEANUP_ONLINE_OP Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of schema</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of schema</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of partition</td>
</tr>
</tbody>
</table>

**Usage Notes**

- If `schema_name`, `table_name` and `partition_name` are specified, this cleans up the failed online move operation for the specified partition.
- If `schema_name` and `table_name` are specified, this cleans up all failed online move operations for all the partitions of the specified table.
- If only `schema_name` is specified, this cleans up all failed online move operations in the schema.
- If no arguments are provided, we cleans up all the failed online move operations in the system.
- All other cases raise `ORA-20000` to inform the user of invalid inputs as arguments.
The DBMS_PCLXUTIL package provides intra-partition parallelism for creating partition-wise local indexes. DBMS_PCLXUTIL circumvents the limitation that, for local index creation, the degree of parallelism is restricted to the number of partitions as only one parallel execution server process for each partition is used.

See Also:
There are several rules concerning partitions and indexes. For more information, see Oracle Database Concepts and Oracle Database Administrator's Guide.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Rules and Limits
- Summary of DBMS_PCLXUTIL Subprograms

115.1 DBMS_PCLXUTIL Overview

DBMS_PCLXUTIL uses the DBMS_JOB package to provide a greater degree of parallelism for creating a local index for a partitioned table. This is achieved by asynchronous inter-partition parallelism using the background processes (with DBMS_SCHEDULER), in combination with intra-partition parallelism using the parallel execution server.

DBMS_PCLXUTIL works with both range and range-hash composite partitioning.

The DBMS_PCLXUTIL package can be used during the following DBA tasks:

1. Local index creation
   The procedure BUILD_PART_INDEX assumes that the dictionary information for the local index already exists. This can be done by issuing the create index SQL command with the UNUSABLE option.

   CREATE INDEX <idx_name> on <tab_name>(...) local(...) unusable;

   This causes the dictionary entries to be created without "building" the index itself, the time consuming part of creating an index. Now, invoking the procedure BUILD_PART_INDEX causes a concurrent build of local indexes with the specified degree of parallelism.

   EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
For composite partitions, the procedure automatically builds local indexes for all subpartitions of the composite table.

2. Local index maintenance

By marking desired partitions usable or unusable, the `BUILD_PART_INDEX` procedure also enables selective rebuilding of local indexes. The `force_opt` parameter provides a way to override this and build local indexes for all partitions.

```sql
ALTER INDEX <idx_name> local(...) unusable;
```

Rebuild only the desired (sub)partitions (that are marked unusable):

```sql
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
```

Rebuild all (sub)partitions using `force_opt = TRUE`:

```sql
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,TRUE);
```

A progress report is produced, and the output appears on screen when the program is ended (because the `DBMS_OUTPUT` package writes messages to a buffer first, and flushes the buffer to the screen only upon termination of the program).

### 115.2 DBMS_PCLXUTIL Security Model

This utility can be run only as table owner, and not as any other user.

### 115.3 DBMS_PCLXUTIL Operational Notes

`DBMS_PCLXUTIL` submits a job for each partition. It is the responsibility of the user/dba to control the number of concurrent jobs by setting the `INIT.ORA` parameter `JOB_QUEUE_PROCESSES` correctly. There is minimal error checking for correct syntax. Any errors are reported in the job queue process trace files.

### 115.4 DBMS_PCLXUTIL Rules and Limits

Because `DBMS_PCLXUTIL` uses the `DBMS_JOB` package, you must be aware of the following limitations pertaining to `DBMS_JOB`:

- You must decide appropriate values for the `job_queue_processes` initialization parameter. Clearly, if the job processes are not started before calling `BUILD_PART_INDEX()`, then the package will not function properly. The background processes are specified by the following `init.ora` parameters:

  ```
  job_queue_processes=n     #the number of background processes = n
  ```

- Failure conditions are reported only in the trace files (a `DBMS_JOB` limitation), making it impossible to give interactive feedback to the user. This package prints a failure message, removes unfinished jobs from the queue, and requests the user to take a look at the `j*.trc` trace files.
115.5 Summary of DBMS_PCLXUTIL Subprograms

The DBMS_PCLXUTIL package has one subprogram, the BUILD_PART_INDEX procedure.

Table 115-1  DBMS_PCLXUTIL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_PART_INDEX Procedure</td>
<td>Provides intra-partition parallelism for creating partition-wise local indexes</td>
</tr>
</tbody>
</table>

115.5.1 BUILD_PART_INDEX Procedure

This procedure provides intra-partition parallelism for creating partition-wise local indexes.

Syntax

```sql
DBMS_PCLXUTIL.BUILD_PART_INDEX (
    jobs_per_batch  IN NUMBER   DEFAULT 1,
    procs_per_job   IN NUMBER   DEFAULT 1,
    tab_name        IN VARCHAR2 DEFAULT NULL,
    idx_name        IN VARCHAR2 DEFAULT NULL,
    force_opt       IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 115-2  BUILD_PART_INDEX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobs_per_batch</td>
<td>The number of concurrent partition-wise &quot;local index builds&quot;.</td>
</tr>
<tr>
<td>procs_per_job</td>
<td>The number of parallel execution servers to be utilized for each local index build (1 &lt;= procs_per_job &lt;= max_slaves).</td>
</tr>
<tr>
<td>tab_name</td>
<td>The name of the partitioned table (an exception is raised if the table does not exist or not partitioned).</td>
</tr>
<tr>
<td>idx_name</td>
<td>The name given to the local index (an exception is raised if a local index is not created on the table tab_name).</td>
</tr>
<tr>
<td>force_opt</td>
<td>If TRUE, then force rebuild of all partitioned indexes; otherwise, rebuild only the partitions marked 'UNUSABLE'.</td>
</tr>
</tbody>
</table>

Usage Notes

This utility can be run only as table owner, and not as any other user.
Examples

Suppose a table PROJECT is created with two partitions PROJ001 and PROJ002, along with a local index IDX.

A call to the procedure BUILD_PART_INDEX(2,4,'PROJECT','IDX',TRUE) produces the following output:

SQLPLUS> EXECUTE dbms_pclxutil.build_part_index(2,4,'PROJECT','IDX',TRUE);
Statement processed.
INFO: Job #21 created for partition PROJ002 with 4 slaves
INFO: Job #22 created for partition PROJ001 with 4 slaves
The `DBMS_PDB` package provides an interface to examine and manipulate data about pluggable databases (PDBs) in a multitenant container database (CDB). It also contains an interface to specify which database objects are application common objects. You can migrate a PDB from one CDB to another CDB. After the migration is complete, all backups of the PDB before migration are available and usable in the destination CDB.

### See Also:

- *Oracle Database Administrator's Guide* for information about creating and managing PDBs and CDBs
- *Oracle XML DB Developer's Guide* for information about configuring protocol ports and DNS mappings
- *Oracle Database SQL Language Reference* for information about creating PDBs
- *Oracle Database Security Guide* regarding how to create audit policies in a multitenant environment

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Summary of DBMS_PDB Subprograms**

### 116.1 DBMS_PDB Overview

A multitenant container database (CDB) is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs). The `DBMS_PDB` package provides an interface to examine and manipulate data about pluggable databases (PDBs). The subprograms in this package can also set a database object to one of the following types of application common objects in an application container: data-linked object, extended data-linked object, or metadata-linked object.

A data-linked application common object stores data in the application root that can be accessed and modified by all of the application PDBs in the application container. For an extended data-linked object, each application PDB can create its own specific data while sharing the common data in the application root. Therefore, with an extended data-linked object, only the data stored in the application root is common for all application PDBs. A metadata-linked application common object stores the metadata for the specific object, such as a table, in the application root, so that the containers in the application container have the same structure for the object but different data. This package also contains a procedure that specifies that a database object is not an application common object.
Typically, the subprograms in this package are used in the following cases:

- An application that is installed in a PDB is migrated to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.
- An application is installed in an application root using an installation program that does not allow setting application common objects.

See Also:

Oracle Database Administrator’s Guide for information about migrating an application to an application container.

116.2 DBMS_PDB Security Model

Users must have the EXECUTE privilege to run the procedures of DBMS_PDB package.

116.3 Summary of DBMS_PDB Subprograms

This table lists and describes DBMS_PDB package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK_PLUG_COMPATIBILITY Function</td>
<td>Uses an XML file describing a pluggable database (PDB) to determine whether it may be plugged into a given multitenant container database (CDB).</td>
</tr>
<tr>
<td>DESCRIBE Procedure</td>
<td>Generates an XML file describing the specified pluggable database (PDB).</td>
</tr>
<tr>
<td>EXPORTRMANBACKUP Procedure</td>
<td>Needs to be called only when a non-CDB is migrated as PDB. This procedure needs to be executed in non-CDB database. For PDB relocation, it is automatically called during unplugin.</td>
</tr>
<tr>
<td>RECOVER Procedure</td>
<td>Generates an XML file describing a pluggable database by using data files belonging to the pluggable database (PDB).</td>
</tr>
<tr>
<td>REMOVE_LINK Procedure</td>
<td>Specifies that a database object is not an application common object.</td>
</tr>
<tr>
<td>SET_DATA_LINKED Procedure</td>
<td>Sets a database object to a data-linked application common object.</td>
</tr>
<tr>
<td>SET_EXT_DATA_LINKED Procedure</td>
<td>Sets a database object to an extended data-linked application common object.</td>
</tr>
<tr>
<td>SET_METADATA_LINKED Procedure</td>
<td>Sets a database object to a metadata-linked application common object.</td>
</tr>
<tr>
<td>SET_PROFILE_EXPLICIT Procedure</td>
<td>Sets a profile as an application common profile in an application container.</td>
</tr>
<tr>
<td>SET_ROLE_EXPLICIT Procedure</td>
<td>Sets a role as an application common role in an application container.</td>
</tr>
</tbody>
</table>
Table 116-1 (Cont.) DBMS_PDB Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_USER_EXPLICIT</td>
<td>This procedure sets a local user as an application common user in an application container.</td>
</tr>
</tbody>
</table>

116.3.1 CHECK_PLUG_COMPATIBILITY Function

This function uses an XML file describing a pluggable database (PDB) to determine whether it may be plugged into a given multitenant container database (CDB).

Syntax

```sql
DBMS_PDB.CHECK_PLUG_COMPATIBILITY (  
    pdb_descr_file    IN   VARCHAR2,
    pdb_name          IN   VARCHAR2 DEFAULT NULL)
RETURN BOOLEAN;
```

Parameters

Table 116-2 CHECK_PLUG_COMPATIBILITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdb_descr_file</td>
<td>Path of the XML file that will contain description of a PDB</td>
</tr>
<tr>
<td>pdb_name</td>
<td>Name which will be given to the PDB represented by pdb_descr_file when plugged into a given CDB. If not specified, the name will be extracted from pdb_descr_file.</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the PDB described by pdb_descr_file is compatible with the given CDB, FALSE otherwise. If this function returns FALSE, then query the PDB_PLUG_IN_VIOLATIONS data dictionary view to find information about the errors that are found.

See Also:

Oracle Database Reference for information about the PDB_PLUG_IN_VIOLATIONS view

116.3.2 DESCRIBE Procedure

This procedure generates an XML file describing the specified pluggable database (PDB). This file can then be passed to the CHECK_PLUG_COMPATIBILITY Function to determine if
the PDB described by the XML file may be plugged into a given multitenant container database (CDB).

Syntax

```
DBMS_PDB.DESCRIBE (pdb_descr_file IN VARCHAR2, pdb_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 116-3  
**DESCRIBE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdb_descr_file</td>
<td>Path of the XML file that will contain description of a PDB</td>
</tr>
<tr>
<td>pdb_name</td>
<td>Name of a PDB to be described. A remote database is specified by including <code>@dblink</code>.</td>
</tr>
</tbody>
</table>

Usage Notes

- If `pdb_name` is omitted, the PDB to which the session is connected will be described.
- If `pdb_name` is omitted, and the session is connected to the Root, an error will be returned.

Related Topics

- **CHECK_PLUGIN_COMPATIBILITY Function**
  This function uses an XML file describing a pluggable database (PDB) to determine whether it may be plugged into a given multitenant container database (CDB).

### 116.3.3 EXPORTRMANBACKUP Procedure

This procedure exports the RMAN backup information that belongs to a pluggable database to its dictionary before unplug so that pre-plugin backups can be used. This procedure needs to be called only for non-CDB to PDB migration. For PDB relocation, this procedure need not be called.

Syntax

```
DBMS_PDB.EXPORTRMANBACKUP (pdb_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 116-4  
**EXPORTRMANBACKUP Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdb_name</td>
<td>Name of a pluggable database whose backup information needs to be exported.</td>
</tr>
<tr>
<td></td>
<td>Omit this parameter if you are connected to a pluggable database or a non-cdb.</td>
</tr>
</tbody>
</table>
Usage Notes

- The PDB must be opened in read/write mode.
- If the database is non-CDB, then pdb_name must be omitted.
- If the pdb_name is omitted, then the pluggable database to which the session is connected is exported.
- If the pdb_name is omitted and the session is connected to the root, an error is returned.

116.3.4 RECOVER Procedure

This procedure generates an XML file describing a pluggable database by using data files belonging to the pluggable database. This XML file can then be used to plug the pluggable database into a multitenant container database (CDB) using the CREATE PLUGGABLE DATABASE statement.

Use this procedure when an XML file describing a pluggable database is corrupted or lost.

Syntax

```
DBMS_PDB.RECOVER (
    pdb_descr_file    IN   VARCHAR2,
    pdb_name          IN   VARCHAR2,
    filenames         IN   VARCHAR2);
```

Parameters

Table 116-5  RECOVER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdb_descr_file</td>
<td>Path of the XML file that contains description of a pluggable database</td>
</tr>
<tr>
<td>pdb_name</td>
<td>Name of a pluggable database</td>
</tr>
<tr>
<td>filenames</td>
<td>Comma-separated list of datafile paths and/or directories containing datafiles for the pluggable database</td>
</tr>
</tbody>
</table>

116.3.5 REMOVE_LINK Procedure

This procedure specifies that a database object is not an application common object. In an application container, application common objects are shared between multiple containers.

Syntax

```
DBMS_PDB.REMOVE_LINK (
    schema_name  IN VARCHAR2,
    object_name  IN VARCHAR2,
    namespace    IN NUMBER,
    edition_name IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 116-6 REMOVE_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the database object.</td>
</tr>
<tr>
<td>namespace</td>
<td>The namespace of the database object.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition for the database object.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database Administrator’s Guide for information about migrating an application to an application container

116.3.6 SET_DATA_LINKED Procedure

This procedure sets a database object to a data-linked application common object. In an application container, data-linked application common objects store data in the application root only, and the data can be accessed by all of the application PDBs in the application container. The data in a data-linked application common object can be modified only in the application root.

You can use this procedure to set data-linked application common objects when you migrate an application that is installed in a PDB to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.

Syntax

DBMS_PDB.SET_DATA_LINKED (  
  schema_name IN VARCHAR2,  
  object_name IN VARCHAR2,  
  namespace IN NUMBER,  
  edition_name IN VARCHAR2 DEFAULT NULL);  

Parameters

Table 116-7 SET_DATA_LINKED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the database object.</td>
</tr>
</tbody>
</table>
116.3.7 SET_EXT_DATA_LINKED Procedure

This procedure sets a database object to an extended data-linked application common object. In an application container, for an extended data-linked object, each application PDB can create its own specific data while sharing the common data in the application root. Therefore, only the data stored in the application root is common for all application PDBs.

You can use this procedure to set extended data-linked application common objects when you migrate an application that is installed in a PDB to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.

Syntax

```
DBMS_PDB.SET_EXT_DATA_LINKED (  
schema_name  IN VARCHAR2,  
object_name  IN VARCHAR2,  
namespace    IN NUMBER,  
edition_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 116-8  SET_EXT_DATA_LINKED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the database object.</td>
</tr>
<tr>
<td>namespace</td>
<td>The namespace of the database object.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition for the database object.</td>
</tr>
</tbody>
</table>
116.3.8 SET_METADATA_LINKED Procedure

This procedure sets a database object to a metadata-linked application common object. In an application container, metadata-linked application common objects store the metadata for specific objects, such as tables, so that the containers that share the application common object have the same structure but different data.

You can use this procedure to set metadata-linked application common objects when you migrate an application that is installed in a PDB to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.

Syntax

```sql
DBMS_PDB.SET_METADATA_LINKED (  
    schema_name  IN VARCHAR2,  
    object_name  IN VARCHAR2,  
    namespace    IN NUMBER,  
    edition_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 116-9  SET_METADATA_LINKED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>schema_name</code></td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td><code>object_name</code></td>
<td>The name of the database object.</td>
</tr>
<tr>
<td><code>namespace</code></td>
<td>The namespace of the database object.</td>
</tr>
<tr>
<td><code>edition_name</code></td>
<td>The <code>NAMESPACE</code> column of the <code>DBA_OBJECTS</code> view shows the namespace of an object.</td>
</tr>
</tbody>
</table>

See Also:

*Oracle Database Administrator’s Guide* for information about migrating an application to an application container
116.3.9 SET_PROFILE_EXPLICIT Procedure

This procedure sets a profile as an application common profile in an application container. This procedure is intended for migrating a profile from a previous release to an application container in the current release.

This procedure must be invoked in an application install, patch, upgrade, or uninstall operation in an application root.

Syntax

```sql
DBMS_PDB.SET_PROFILE_EXPLICIT (
    profile_name IN VARCHAR2);
```

Parameters

Table 116-10 SET_PROFILE_EXPLICIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>The name of the profile.</td>
</tr>
</tbody>
</table>

See Also:

*Oracle Database Administrator’s Guide* for information about migrating an application to an application container.

116.3.10 SET_ROLE_EXPLICIT Procedure

This procedure sets a role as an application common role in an application container. This procedure is intended for migrating a role from a previous release to an application container in the current release.

This procedure must be invoked in an application install, patch, upgrade, or uninstall operation in an application root.

Syntax

```sql
DBMS_PDB.SET_ROLE_EXPLICIT (
    role_name IN VARCHAR2);
```

Parameters

Table 116-11 SET_ROLE_EXPLICIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_name</td>
<td>The name of the role.</td>
</tr>
</tbody>
</table>
116.3.11 SET_USER_EXPLICIT Procedure

This procedure sets a local user as an application common user in an application container.

This procedure must be invoked in an application install, patch, upgrade, or uninstall operation in an application root.

Syntax

```sql
DBMS_PDB.SET_USER_EXPLICIT (user_name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The name of the user.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database Administrator's Guide for information about migrating an application to an application container.
In an application container with a pre-installed application, the `DBMS_PDB_ALTER_SHARING` package provides an interface to set database objects as application common objects or to specify that a database object is not an application common object.

**Note:**

This package provides a subset of the subprograms in the `DBMS_PDB` package. Users who do not have `EXECUTE` privilege on the `DBMS_PDB` package can be granted `EXECUTE` privilege on this package to run these subprograms.

**See Also:**

- *Oracle Database Administrator's Guide* for information about migrating an application to an application container
- *Oracle XML DB Developer's Guide* for information about configuring protocol ports and DNS mappings
- *Oracle Database SQL Language Reference* for information about creating PDBs
- *Oracle Database Security Guide* regarding how to create audit policies in a multitenant environment

This chapter contains the following topics:

- DBMS_PDB_ALTER_SHARING Overview
- DBMS_PDB_ALTER_SHARING Security Model
- Summary of DBMS_PDB_ALTER_SHARING Subprograms

### 117.1 DBMS_PDB_ALTER_SHARING Overview

The subprograms in this package can set a database object to one of the following types of application common objects in an application container: data-linked object, extended data-linked object, or metadata-linked object.

A data-linked application common object stores data in the application root that can be accessed and modified by all of the application PDBs in the application container. For an extended data-linked object, each application PDB can create its own specific data while sharing the common data in the application root. Therefore, with an extended data-linked object, only the data stored in the application root is common for all application PDBs. A metadata-linked application common object stores the metadata for the specific object, such as a table, in the application root, so that the containers in the application container have the
same structure for the object but different data. This package also contains a procedure that specifies that a database object is not an application common object.

You can use the subprograms in this package when you

Typically, the subprograms in this package are used in the following cases:

- An application that is installed in a PDB is migrated to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.
- An application is installed in an application root using an installation program that does not allow setting application common objects.

See Also:
Oracle Database Administrator's Guide for information about migrating an application to an application container

117.2 DBMS_PDB_ALTER_SHARING Security Model

This package provides a subset of the subprograms in the DBMS_PDB package. User who do not have EXECUTE privilege on the DBMS_PDB package can be granted EXECUTE privilege on this package to run these subprograms.

Security on this package can be controlled in either of the following ways:
- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

117.3 Summary of DBMS_PDB_ALTER_SHARING Subprograms

This table lists and briefly describes DBMS_PDB_ALTER_SHARING package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOVE_LINK Procedure</td>
<td>Specifies that a database object is not an application common object</td>
</tr>
<tr>
<td>SET_DATA_LINKED Procedure</td>
<td>Sets a database object to a data-linked application common object</td>
</tr>
<tr>
<td>SET_EXT_DATA_LINKED Procedure</td>
<td>Sets a database object to an extended data-linked application common object</td>
</tr>
</tbody>
</table>
Table 117-1  (Cont.) DBMS_PDB_ALTER_SHARING Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_METADATA_LINKED Procedure</td>
<td>Sets a database object to a metadata-linked application common object</td>
</tr>
<tr>
<td>SET_PROFILE_EXPLICIT Procedure</td>
<td>Sets a profile as an application common profile in an application container.</td>
</tr>
<tr>
<td>SET_ROLE_EXPLICIT Procedure</td>
<td>Sets a role as an application common role in an application container.</td>
</tr>
<tr>
<td>SET_USER_EXPLICIT Procedure</td>
<td>Sets a local user as an application common user in an application container.</td>
</tr>
</tbody>
</table>

117.3.1 REMOVE_LINK Procedure

This procedure specifies that a database object is not an application common object. In an application container, application common objects are shared between multiple containers.

Syntax

```
DBMS_PDB_ALTER_SHARING.REMOVE_LINK (
  schema_name  IN VARCHAR2,
  object_name  IN VARCHAR2,
  namespace    IN NUMBER,
  edition_name IN VARCHAR2  DEFAULT NULL);
```

Parameters

Table 117-2  REMOVE_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the database object.</td>
</tr>
<tr>
<td>namespace</td>
<td>The namespace of the database object.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition for the database object.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database Administrator’s Guide for information about migrating an application to an application container

117.3.2 SET_DATA_LINKED Procedure

This procedure sets a database object to a data-linked application common object. In an application container, data-linked application common objects store data in the application root only, and the data can be accessed by all of the application PDBs in the application.
container. The data in a data-linked application common object can be modified only in the application root.

You can use this procedure to set data-linked application common objects when you migrate an application that is installed in a PDB to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.

**Syntax**

```sql
DBMS_PDB_ALTER_SHARING.SET_DATA_LINKED (  
    schema_name  IN VARCHAR2,  
    object_name  IN VARCHAR2,  
    namespace    IN NUMBER,  
    edition_name IN VARCHAR2  DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the database object.</td>
</tr>
<tr>
<td>namespace</td>
<td>The namespace of the database object.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition for the database object.</td>
</tr>
</tbody>
</table>

**See Also:**

*Oracle Database Administrator's Guide* for information about migrating an application to an application container

### 117.3.3 SET_EXT_DATA_LINKED Procedure

This procedure sets a database object to an extended data-linked application common object. In an application container, for an extended data-linked object, each application PDB can create its own specific data while sharing the common data in the application root. Therefore, only the data stored in the application root is common for all application PDBs.

You can use this procedure to set extended data-linked application common objects when you migrate an application that is installed in a PDB to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.
### Syntax

```sql
DBMS_PDB_ALTER_SHARING.SET_EXT_DATA_LINKED (
    schema_name  IN VARCHAR2,
    object_name  IN VARCHAR2,
    namespace    IN NUMBER,
    edition_name IN VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 117-4 SET_EXT_DATA_LINKED Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the database object.</td>
</tr>
<tr>
<td>namespace</td>
<td>The namespace of the database object.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition for the database object.</td>
</tr>
</tbody>
</table>

The `NAMESPACE` column of the `DBA_OBJECTS` view shows the namespace of an object.

### See Also:

*Oracle Database Administrator’s Guide* for information about migrating an application to an application container

### 117.3.4 SET_METADATA_LINKED Procedure

This procedure sets a database object to a metadata-linked application common object. In an application container, metadata-linked application common objects store the metadata for specific objects, such as tables, so that the containers that share the application common object have the same structure but different data.

You can use this procedure to set metadata-linked application common objects when you migrate an application that is installed in a PDB to an application container. The application can be migrated to the application root or to an application PDB. For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c Release 1 (12.1) CDB to an application container in an Oracle Database 12c Release 2 (12.2) CDB.

**Syntax**

```sql
DBMS_PDB_ALTER_SHARING.SET_METADATA_LINKED (
    schema_name  IN VARCHAR2,
    object_name  IN VARCHAR2,
    namespace    IN NUMBER,
    edition_name IN VARCHAR2  DEFAULT NULL);
```
Parameters

Table 117-5  SET_METADATA_LINKED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema that owns the database object.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the database object.</td>
</tr>
<tr>
<td>namespace</td>
<td>The namespace of the database object.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition for the database object.</td>
</tr>
</tbody>
</table>

The NAMESPACE column of the DBA_OBJECTS view shows the namespace of an object.

See Also:

Oracle Database Administrator’s Guide for information about migrating an application to an application container

117.3.5 SET_PROFILE_EXPLICIT Procedure

This procedure sets a profile as an application common profile in an application container. This procedure is intended for migrating a profile from a previous release to an application container in the current release.

This procedure must be invoked in an application install, patch, upgrade, or uninstall operation in an application root.

Syntax

```
DBMS_PDB_ALTER_SHARING.SET_PROFILE_EXPLICIT (profile_name IN VARCHAR2);
```

Parameters

Table 117-6  SET_PROFILE_EXPLICIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>The name of the profile.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database Administrator’s Guide for information about migrating an application to an application container.
117.3.6 SET_ROLE_EXPLICIT Procedure

This procedure sets a role as an application common role in an application container. This procedure is intended for migrating a role from a previous release to an application container in the current release.

This procedure must be invoked in an application install, patch, upgrade, or uninstall operation in an application root.

Syntax

DBMS_PDB_ALTER_SHARING.SET_ROLE_EXPLICIT (role_name IN VARCHAR2);

Parameters

Table 117-7  SET_ROLE_EXPLICIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_name</td>
<td>The name of the role.</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database Administrator’s Guide for information about migrating an application to an application container.

117.3.7 SET_USER_EXPLICIT Procedure

This procedure sets a local user as an application common user in an application container.

This procedure must be invoked in an application install, patch, upgrade, or uninstall operation in an application root.

Syntax

DBMS_PDB_ALTER_SHARING.SET_USER_EXPLICIT (user_name IN VARCHAR2);

Parameters

Table 117-8  SET_USER_EXPLICIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The name of the user.</td>
</tr>
</tbody>
</table>
See Also:

Oracle Database Administrator's Guide for information about migrating an application to an application container.
The DBMS_PERF package provides an interface for generating database performance reports. All subprograms return an active report and these reports can be generated at the system level, session level or at SQL level.

The DBMS_PERF package requires the DBA role.

This table lists the DBMS_PERF subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT_PERFHUB</td>
<td>Generates a composite active performance report of the entire database system for a specified time period</td>
</tr>
<tr>
<td>REPORT_SESSION</td>
<td>Generates a performance report for a specific database session where a session is identified by inst_id, sid, and serial_num.</td>
</tr>
<tr>
<td>REPORT_SQL Function</td>
<td>Generates an active performance report for a particular SQL statement identified by its sql_id.</td>
</tr>
</tbody>
</table>
118.3.1 REPORT_PERFHUB Function

This function generates a composite active performance report of the entire database system for a specified time period.

Syntax

```
DBMS_PERF.REPORT_PERFHUB (  
  is_realtime          IN NUMBER   DEFAULT NULL,  
  outer_start_time     IN DATE     DEFAULT NULL,  
  outer_end_time       IN DATE     DEFAULT NULL,  
  selected_start_time  IN DATE     DEFAULT NULL,  
  selected_end_time    IN DATE     DEFAULT NULL,  
  inst_id              IN NUMBER   DEFAULT NULL,  
  dbid                 IN NUMBER   DEFAULT NULL,  
  monitor_list_detail  IN NUMBER   DEFAULT NULL,  
  workload_sql_detail  IN NUMBER   DEFAULT NULL,  
  addm_task_detail     IN NUMBER   DEFAULT NULL,  
  report_reference     IN VARCHAR2 DEFAULT NULL,  
  report_level         IN VARCHAR2 DEFAULT NULL,  
  type                 IN VARCHAR2 DEFAULT 'ACTIVE',  
  base_path            IN VARCHAR2 DEFAULT NULL);  
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_realtime</td>
<td>If 1, then real-time. If NULL (default) or 0, then historical mode.</td>
</tr>
<tr>
<td>outer_start_time</td>
<td>Start time of outer period shown in the time selector.</td>
</tr>
<tr>
<td></td>
<td>If NULL (default):</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0 (historical), then 24 hours before outer_end_time.</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1 (realtime mode), then 1 hour before outer_end_time.</td>
</tr>
<tr>
<td>outer_end_time</td>
<td>End time of outer period shown in the time selector.</td>
</tr>
<tr>
<td></td>
<td>If NULL (default), then latest AWR snapshot.</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0 (historical), then the latest AWR snapshot</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1 (realtime mode), this is the current time (and any</td>
</tr>
<tr>
<td></td>
<td>input is ignored)</td>
</tr>
<tr>
<td>selected_start_time</td>
<td>Start time period of selection. If NULL (default)</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0, then 1 hour before selected_end_time</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1, then 5 minutes before selected_end_time</td>
</tr>
<tr>
<td>selected_end_time</td>
<td>End time period of selection. If NULL (default)</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0, then latest AWR snapshot</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1, then current time</td>
</tr>
</tbody>
</table>
### Table 118-2  (Cont.) REPORT_PERFHUB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst_id</td>
<td>Instance ID to for which to retrieve data</td>
</tr>
<tr>
<td></td>
<td>• If -1, then current instance</td>
</tr>
<tr>
<td></td>
<td>• If number is specified, then for that instance</td>
</tr>
<tr>
<td></td>
<td>• If NULL (default), then all instances</td>
</tr>
<tr>
<td>dbid</td>
<td>DBID to query.</td>
</tr>
<tr>
<td></td>
<td>• If NULL, then current DBID.</td>
</tr>
<tr>
<td></td>
<td>• If is_realtime=1, then DBID must be the local DBID.</td>
</tr>
<tr>
<td>monitor_list_detail</td>
<td>Top N in SQL monitor list for which to retrieve SQL monitor details.</td>
</tr>
<tr>
<td></td>
<td>• If NULL (default), then retrieves top 10</td>
</tr>
<tr>
<td></td>
<td>• If 0, then retrieves no monitor list details</td>
</tr>
<tr>
<td>workload_sql_detail</td>
<td>Top N in Workload Top SQL list to retrieve monitor details,</td>
</tr>
<tr>
<td></td>
<td>• If NULL (default), then retrieves top 10</td>
</tr>
<tr>
<td></td>
<td>• If 0, then retrieves no monitor list details</td>
</tr>
<tr>
<td>addm_task_detail</td>
<td>Maximum N latest ADDM tasks to retrieve</td>
</tr>
<tr>
<td></td>
<td>• If NULL (default), retrieves available data but no more than N</td>
</tr>
<tr>
<td></td>
<td>• If 0, then retrieves no ADDM task details</td>
</tr>
<tr>
<td>report_reference</td>
<td>Must be NULL when used from SQL*Plus.</td>
</tr>
<tr>
<td>report_level</td>
<td>'typical' will get all tabs in performance hub</td>
</tr>
<tr>
<td>type</td>
<td>Report type:</td>
</tr>
<tr>
<td></td>
<td>• 'ACTIVE' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'xml' returns XML</td>
</tr>
<tr>
<td>base_path</td>
<td>URL path for HTML resources since flex HTML requires access to external files. This is only valid for type='ACTIVE' and is typically not used. Default value will retrieve the required files from OTN.</td>
</tr>
</tbody>
</table>

### Usage Notes

- Once a time period is selected, the performance information is collected and presented based on performance subject areas.
- The time period can be real-time or historical.
- When real-time data is selected, more granular data is presented because data points are available every minute.
- When historical data is selected, more detailed data (broken down by different metrics) is presented, but the data points are averaged out to the Automatic Workload Repository (AWR) interval (usually an hour).
- Different tabs are available in the Performance Hub, depending on whether is_real-time is 1 for real time mode or 0 for historical mode.
118.3.2 REPORT_SESSION Function

This function produces a performance report for a specific database session where a session is identified by `inst_id`, `sid`, and `serial_num`.

If any of those parameters are missing, then the report is for the current session.

The session-level performance report contains the following tabs:

- **Summary** - This tab contains key identifiers and attributes of the session along with a summary of its activity data. It also contains a list of SQLs, PLSQL blocks and Database Operations (DBOP) executed by that session that were monitored by Real-time SQL Monitoring.

- **Activity** - This tab shows activity broken down by wait classes for this session. The data used for this chart is fetched from Active Session History (ASH).

- **Metrics** - This tab shows charts for certain key metrics for the selected session over time and is only available in historical mode. Some of the metrics shown are CPU usage, PGA usage, IO Throughput and IO Requests.

**Syntax**

```sql
DBMS_PERF.REPORT_SESSION (  
    inst_id               IN NUMBER   DEFAULT NULL,
    sid                   IN NUMBER   DEFAULT NULL,
    serial                IN NUMBER   DEFAULT NULL,
    is_realtime           IN NUMBER   DEFAULT NULL,
    outer_start_time      IN DATE     DEFAULT NULL,
    outer_end_time        IN DATE     DEFAULT NULL,
    selected_start_time   IN DATE     DEFAULT NULL,
    selected_end_time     IN DATE     DEFAULT NULL,
    dbid                  IN NUMBER   DEFAULT NULL,
    monitor_list_detail   IN NUMBER   DEFAULT NULL,
    report_reference      IN VARCHAR2 DEFAULT NULL,
    report_level          IN VARCHAR2 DEFAULT NULL,
    type                   IN VARCHAR2 DEFAULT 'ACTIVE',
    base_path             IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inst_id</code></td>
<td>Instance ID to which to retrieve data. If <code>NULL</code> (default), then instance of current session.</td>
</tr>
<tr>
<td><code>sid</code></td>
<td>Session ID for which to retrieve performance. If <code>NULL</code>, uses current session.</td>
</tr>
<tr>
<td><code>serial</code></td>
<td>Serial# of session. If <code>NULL</code>, then the serial# of the specified <code>sid</code> is used provided the session is connected.</td>
</tr>
<tr>
<td><code>is_realtime</code></td>
<td>If 1, then real-time. If <code>NULL</code> (default) or 0, then historical mode.</td>
</tr>
</tbody>
</table>
Table 118-3 (Cont.) REPORT_SESSION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outer_start_time</td>
<td>Start time of outer period shown in the time selector. If NULL (default):</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0 (historical), then 24 hours before outer_end_time.</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1 (realtime mode), then 1 hour before outer_end_time.</td>
</tr>
<tr>
<td>outer_end_time</td>
<td>End time of outer period shown in the time selector. If NULL (default):</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0 (historical), then latest AWR snapshot.</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1 (realtime mode), this is the current time (and any</td>
</tr>
<tr>
<td></td>
<td>input is ignored)</td>
</tr>
<tr>
<td>selected_start_time</td>
<td>Start time period of selection. If NULL (default)</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0, then 1 hour before selected_end_time</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1, then 5 minutes before selected_end_time</td>
</tr>
<tr>
<td>selected_end_time</td>
<td>End time period of selection. If NULL (default)</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=0, then latest AWR snapshot</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1, then current time</td>
</tr>
<tr>
<td>dbid</td>
<td>DBID to query.</td>
</tr>
<tr>
<td></td>
<td>- If NULL, then current DBID.</td>
</tr>
<tr>
<td></td>
<td>- If is_realtime=1, then DBID must be the local DBID.</td>
</tr>
<tr>
<td>monitor_list_detail</td>
<td>Top N in SQL monitor list for which to retrieve SQL monitor details.</td>
</tr>
<tr>
<td></td>
<td>- If NULL (default), then retrieves top 10</td>
</tr>
<tr>
<td></td>
<td>- If 0, then retrieves no monitor list details</td>
</tr>
<tr>
<td>report_reference</td>
<td>Must be NULL when used from SQL‘Plus.</td>
</tr>
<tr>
<td>report_level</td>
<td>'typical' will get all tabs in the session hub (or session details)</td>
</tr>
<tr>
<td>type</td>
<td>Report type:</td>
</tr>
<tr>
<td></td>
<td>- 'ACTIVE' (default)</td>
</tr>
<tr>
<td></td>
<td>- 'xml' returns XML</td>
</tr>
<tr>
<td>base_path</td>
<td>URL path for HTML resources since flex HTML requires access to external</td>
</tr>
<tr>
<td></td>
<td>files</td>
</tr>
</tbody>
</table>

118.3.3 REPORT_SQL Function

This function generates an active performance report for a particular SQL statement identified by its sql_id.

The SQL-level performance report contains the following tabs:

- Summary - This tab contains an overview of the SQL statement with key attributes like the SQL text, user name, sessions executing it, and related information. It also contains a
Plans tab which shows statistics and activity for each distinct plan for this SQL statement found in memory and in the AWR.

- **Activity** - This tab shows activity broken down by wait classes for this SQL statement. The data used for this chart is fetched from Active Session History (ASH).
- **Execution Statistics** - This tab shows statistics and activity for each distinct plan for this statement along with a graphical and tabular representation of the plan.
- **Monitored SQL** - All executions of this SQL statement that were monitored by Real-time SQL Monitoring are listed in this tab.
- **Plan Control** - This tab shows information about SQL Profiles and SQL Plan Baselines if they exist for this SQL statement.
- **Historical Statistics** - This tab is available only in Historical mode. It contains statistics, such as number of executions, number of I/Os, rows processed, and other information produced over time for different execution plans. This information is retrieved from AWR.

**Syntax**

```sql
DBMS_PERF.REPORT_SQL (  
    sql_id               IN varchar2 default null,  
    is_realtime          IN number   default null,  
    outer_start_time     IN date     default null,  
    outer_end_time       IN date     default null,  
    selected_start_time  IN date     default null,  
    selected_end_time    IN date     default null,  
    inst_id              IN number   default null,  
    dbid                 IN number   default null,  
    monitor_list_detail  IN number   default null,  
    report_reference     IN varchar2 default null,  
    report_level         IN varchar2 default null,  
    type                 IN varchar2 default 'ACTIVE',  
    base_path            IN varchar2 default null),  
RETURN CLOB;
```

**Parameters**

**Table 118-4 REPORT_SQL Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQL_ID for which to retrieve performance. If NULL, gets SQL details for the last executed SQL statement.</td>
</tr>
<tr>
<td>is_realtime</td>
<td>If 1, then real-time. If NULL (default) or 0, then historical mode.</td>
</tr>
<tr>
<td>outer_start_time</td>
<td>Start of outer period shown in the time selector. If NULL (default):</td>
</tr>
<tr>
<td></td>
<td>• If is_realtime=0 (historical), then 24 hours before outer_end_time.</td>
</tr>
<tr>
<td></td>
<td>• If is_realtime=1 (realtime mode), then 1 hour before outer_end_time.</td>
</tr>
</tbody>
</table>
### Table 118-4 (Cont.) REPORT_SQL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outer_end_time</td>
<td>End time of outer period shown in the time selector. If NULL (default), then latest AWR snapshot.</td>
</tr>
<tr>
<td></td>
<td>• If <code>is_realtime=0</code> (historical), then the latest AWR snapshot</td>
</tr>
<tr>
<td></td>
<td>• If <code>is_realtime=1</code> (realtime mode), this is the current time (and any input is ignored)</td>
</tr>
<tr>
<td>selected_start_time</td>
<td>Start time period of selection. If NULL (default)</td>
</tr>
<tr>
<td></td>
<td>• If <code>is_realtime=0</code>, then 1 hour before <code>selected_end_time</code></td>
</tr>
<tr>
<td></td>
<td>• If <code>is_realtime=1</code>, then 5 minutes before <code>selected_end_time</code></td>
</tr>
<tr>
<td>selected_end_time</td>
<td>End time period of selection. If NULL (default)</td>
</tr>
<tr>
<td></td>
<td>• If <code>is_realtime=0</code>, then latest AWR snapshot</td>
</tr>
<tr>
<td></td>
<td>• If <code>is_realtime=1</code>, then current time</td>
</tr>
<tr>
<td>inst_id</td>
<td>Instance ID to for which to retrieve data. If NULL (default), then instance of current session.</td>
</tr>
<tr>
<td>dbid</td>
<td>DBID to query.</td>
</tr>
<tr>
<td></td>
<td>• If NULL, then current DBID.</td>
</tr>
<tr>
<td></td>
<td>• If <code>is_realtime=1</code>, then DBID must be the local DBID.</td>
</tr>
<tr>
<td>monitor_list_detail</td>
<td>Top N in SQL monitor list for which to retrieve SQL monitor details.</td>
</tr>
<tr>
<td></td>
<td>• If NULL (default), then retrieves top 10</td>
</tr>
<tr>
<td></td>
<td>• If 0, then retrieves no monitor list details</td>
</tr>
<tr>
<td>report_reference</td>
<td>Must be NULL when used from SQL*Plus.</td>
</tr>
<tr>
<td>report_level</td>
<td>'typical' will get all tabs in performance hub</td>
</tr>
<tr>
<td>type</td>
<td>Report type:</td>
</tr>
<tr>
<td></td>
<td>• 'ACTIVE' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'xml' returns XML</td>
</tr>
<tr>
<td>base_path</td>
<td>URL path for HTML resources since flex HTML requires access to external files</td>
</tr>
</tbody>
</table>
DBMS_PIPE

The DBMS_PIPE package lets two or more sessions in the same instance communicate. Oracle pipes are similar in concept to the pipes used in UNIX, but Oracle pipes are not implemented using the operating system pipe mechanisms.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Operational Notes
- Exceptions
- Examples
- Summary of DBMS_PIPE Subprograms

119.1 DBMS_PIPE Overview

Pipe functionality has several potential applications: external service interface, independent transactions, alerters (non-transactional), debugging, and concentrator.

- External service interface: You can communicate with user-written services that are external to the RDBMS. This can be done effectively in a shared server process, so that several instances of the service are executing simultaneously. Additionally, the services are available asynchronously. The requestor of the service does not need to block a waiting reply. The requestor can check (with or without time out) at a later time. The service can be written in any of the 3GL languages that Oracle supports.

- Independent transactions: The pipe can communicate to a separate session which can perform an operation in an independent transaction (such as logging an attempted security violation detected by a trigger).

- Alerters (non-transactional): You can post another process without requiring the waiting process to poll. If an "after-row" or "after-statement" trigger were to alert an application, then the application would treat this alert as an indication that the data probably changed. The application would then read the data to get the current value. Because this is an "after" trigger, the application would want to do a "SELECT FOR UPDATE" to make sure it read the correct data.

- Debugging: Triggers and stored procedures can send debugging information to a pipe. Another session can keep reading out of the pipe and display it on the screen or write it to a file.

- Concentrator: This is useful for multiplexing large numbers of users over a fewer number of network connections, or improving performance by concentrating several user-transactions into one DBMS transaction.
119.2 DBMS_PIPE Security Model

Security can be achieved by use of GRANT EXECUTE on the DBMS_PIPE package by creating a pipe using the private parameter in the CREATE_PIPE function and by writing cover packages that only expose particular features or pipenames to particular users or roles.

Depending upon your security requirements, you may choose to use either public pipes or private pipes, which are described in DBMS_PIPE Operational Notes.

119.3 DBMS_PIPE Constants

This is the maximum time to wait attempting to send or receive a message.

maxwait constant integer := 86400000; /* 1000 days */

119.4 DBMS_PIPE Operational Notes

Information sent through Oracle pipes is buffered in the system global area (SGA). All information in pipes is lost when the instance is shut down.

WARNING:

Pipes are independent of transactions. Be careful using pipes when transaction control can be affected.

The operation of DBMS_PIPE is considered with regard to the following topics:

- Public Pipes
- Writing and Reading Pipes
- Private Pipes

Public Pipes

You may create a public pipe either implicitly or explicitly. For implicit public pipes, the pipe is automatically created when it is referenced for the first time, and it disappears when it no longer contains data. Because the pipe descriptor is stored in the SGA, there is some space usage overhead until the empty pipe is aged out of the cache.

You create an explicit public pipe by calling the CREATE_PIPE function with the private flag set to FALSE. You must deallocate explicitly-created pipes by calling the REMOVE_PIPE function.

The domain of a public pipe is the schema in which it was created, either explicitly or implicitly.

Reading and Writing Pipes

Each public pipe works asynchronously. Any number of schema users can write to a public pipe, as long as they have EXECUTE permission on the DBMS_PIPE package, and
they know the name of the public pipe. However, once buffered information is read by one user, it is emptied from the buffer, and is not available for other readers of the same pipe.

The sending session builds a message using one or more calls to the `PACK_MESSAGE` procedure. This procedure adds the message to the session's local message buffer. The information in this buffer is sent by calling the `SEND_MESSAGE` function, designating the pipe name to be used to send the message. When `SEND_MESSAGE` is called, all messages that have been stacked in the local buffer are sent.

A process that wants to receive a message calls the `RECEIVE_MESSAGE` function, designating the pipe name from which to receive the message. The process then calls the `UNPACK_MESSAGE` procedure to access each of the items in the message.

**Private Pipes**

You explicitly create a private pipe by calling the `CREATE_PIPE` function. Once created, the private pipe persists in shared memory until you explicitly deallocate it by calling the `REMOVE_PIPE` function. A private pipe is also deallocated when the database instance is shut down.

You cannot create a private pipe if an implicit pipe exists in memory and has the same name as the private pipe you are trying to create. In this case, `CREATE_PIPE` returns an error.

Access to a private pipe is restricted to:

- Sessions running under the same userid as the creator of the pipe
- Stored subprograms executing in the same userid privilege domain as the pipe creator
- Users connected as `SYSDBA`

An attempt by any other user to send or receive messages on the pipe, or to remove the pipe, results in an immediate error. Any attempt by another user to create a pipe with the same name also causes an error.

As with public pipes, you must first build your message using calls to `PACK_MESSAGE` before calling `SEND_MESSAGE`. Similarly, you must call `RECEIVE_MESSAGE` to retrieve the message before accessing the items in the message by calling `UNPACK_MESSAGE`.

### 119.5 DBMS_PIPE Exceptions

`DBMS_PIPE` package subprograms can return the errors listed in the following table.

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-23321:</td>
<td>Pipename may not be null. This can be returned by the <code>CREATE_PIPE</code> function, or any subprogram that takes a pipe name as a parameter.</td>
</tr>
<tr>
<td>ORA-23322:</td>
<td>Insufficient privilege to access pipe. This can be returned by any subprogram that references a private pipe in its parameter list.</td>
</tr>
</tbody>
</table>
119.6 DBMS_PIPE Examples

These examples show use of DBMS_PIPE in debugging PL/SQL, debugging Pro*C, executing system commands, and an external service interface.

Example 1: Debugging - PL/SQL

This example shows the procedure that a PL/SQL program can call to place debugging information in a pipe.

```sql
CREATE OR REPLACE PROCEDURE debug (msg VARCHAR2) AS
    status NUMBER;
BEGIN
    DBMS_PIPE.PACK_MESSAGE(LENGTH(msg));
    DBMS_PIPE.PACK_MESSAGE(msg);
    status := DBMS_PIPE.SEND_MESSAGE('plsql_debug');
    IF status != 0 THEN
        raise_application_error(-20099, 'Debug error');
    END IF;
END debug;
```

Example 2: Debugging - Pro*C

The following Pro*C code receives messages from the PLSQL_DEBUG pipe in the previous example, and displays the messages. If the Pro*C session is run in a separate window, then it can be used to display any messages that are sent to the debug procedure from a PL/SQL program executing in a separate session.

```c
#include <stdio.h>
#include <string.h>
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR username[20];
    int     status;
    int     msg_length;
    char    retval[2000];
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;
void sql_error();
main()
{
    -- Prepare username:
    strcpy(username.arr, "HR/<password>");
    username.len = strlen(username.arr);

    EXEC SQL WHENEVER SQLERROR DO sql_error();
    EXEC SQL CONNECT :username;
    printf("connected\n");

    -- Start an endless loop to look for and print messages on the pipe:
    FOR (;;) {
        EXEC SQL EXECUTE
            DECLARE
```
len INTEGER;
typ INTEGER;
sta INTEGER;
chr VARCHAR2(2000);
BEGIN
    chr := '';
    sta := dbms_pipe.receive_message('plsql_debug');
    IF sta = 0 THEN
        DBMS_PIPE.UNPACK_MESSAGE(len);
        DBMS_PIPE.UNPACK_MESSAGE(chr);
    END IF;
    :status := sta;
    :retval := chr;
    IF len IS NOT NULL THEN
        :msg_length := len;
    ELSE
        :msg_length := 2000;
    END IF;
END;
END-EXEC;

IF (status == 0)
    printf("%.*s\n", msg_length, retval);
ELSE
    printf("abnormal status, value is %d\n", status);
}

void sql_error()
{
    char msg[1024];
    int rlen, len;
    len = sizeof(msg);
    sqlglm(msg, &len, &rlen);
    printf("ORACLE ERROR\n");
    printf("%.*s\n", rlen, msg);
    exit(1);
}

Example 3: Execute System Commands

This example shows PL/SQL and Pro*C code let a PL/SQL stored procedure (or anonymous block) call PL/SQL procedures to send commands over a pipe to a Pro*C program that is listening for them.

The Pro*C program sleeps and waits for a message to arrive on the named pipe. When a message arrives, the Pro*C program processes it, carrying out the required action, such as executing a UNIX command through the system() call or executing a SQL command using embedded SQL.

DAEMON.SQL is the source code for the PL/SQL package. This package contains procedures that use the DBMS_PIPE package to send and receive message to and from the Pro*C daemon. Note that full handshaking is used. The daemon always sends a message back to the package (except in the case of the STOP command). This is valuable, because it allows the PL/SQL procedures to be sure that the Pro*C daemon is running.

You can call the DAEMON packaged procedures from an anonymous PL/SQL block using SQL*Plus or Enterprise Manager. For example:

SQLPLUS> variable rv number
SQLPLUS> execute :rv := DAEMON.EXECUTE_SYSTEM('ls -la');
On a UNIX system, this causes the Pro*C daemon to execute the command `system("ls -la").`

Remember that the daemon needs to be running first. You might want to run it in the background, or in another window beside the SQL*Plus or Enterprise Manager session from which you call it.

The DAEMON.SQL also uses the DBMS_OUTPUT package to display the results. For this example to work, you must have execute privileges on this package.

**DAEMON.SQL Example.** This is the code for the PL/SQL DAEMON package:

```sql
CREATE OR REPLACE PACKAGE daemon AS
    FUNCTION execute_sql(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
        RETURN NUMBER;

    FUNCTION execute_system(command VARCHAR2,
                               timeout NUMBER DEFAULT 10)
        RETURN NUMBER;

    PROCEDURE stop(timeout NUMBER DEFAULT 10);
END daemon;
/

CREATE OR REPLACE PACKAGE BODY daemon AS

    FUNCTION execute_system(command VARCHAR2,
                               timeout NUMBER DEFAULT 10)
        RETURN NUMBER IS

        status       NUMBER;
        result       VARCHAR2(20);
        command_code NUMBER;
        pipe_name    VARCHAR2(30);
        BEGIN
            pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

            DBMS_PIPE.PACK_MESSAGE('SYSTEM');
            DBMS_PIPE.PACK_MESSAGE(pipe_name);
            DBMS_PIPE.PACK_MESSAGE(command);
            status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
            IF status <> 0 THEN
                RAISE_APPLICATION_ERROR(-20010,
                            'Execute_system: Error while sending. Status = ' ||
                            status);
            END IF;

            status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);
            IF status <> 0 THEN
                RAISE_APPLICATION_ERROR(-20011,
                            'Execute_system: Error while receiving. Status = ' ||
                            status);
            END IF;

            DBMS_PIPE.UNPACK_MESSAGE(result);
            IF result <> 'done' THEN
                RAISE_APPLICATION_ERROR(-20012,
                            'Execute_system: Done not received. ');
            END IF;

        END execute_system;

END daemon;
```

Chapter 119
DBMS_PIPE Examples

119-6
FUNCTION execute_sql(command VARCHAR2,  
    timeout NUMBER DEFAULT 10) RETURN NUMBER IS  

    status       NUMBER;  
    result       VARCHAR2(20);  
    command_code NUMBER;  
    pipe_name    VARCHAR2(30);  

BEGIN  
    pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;  
    DBMS_PIPE.PACK_MESSAGE('SQL');  
    DBMS_PIPE.PACK_MESSAGE(pipe_name);  
    DBMS_PIPE.PACK_MESSAGE(command);  
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);  
    IF status <> 0 THEN  
        RAISE_APPLICATION_ERROR(-20020,  
            'execute_sql: Error while sending. Status = ' || status);  
    END IF;  

    status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);  
    IF status <> 0 THEN  
        RAISE_APPLICATION_ERROR(-20021,  
            'execute_sql: Error while receiving. Status = ' || status);  
    END IF;  

    DBMS_PIPE.UNPACK_MESSAGE(result);  
    IF result <> 'done' THEN  
        RAISE_APPLICATION_ERROR(-20022,  
            'execute_sql: done not received.');  
    END IF;  

    DBMS_PIPE.UNPACK_MESSAGE(command_code);  
    DBMS_OUTPUT.PUT_LINE  
        ('SQL command executed. sqlcode = ' || command_code);  
    RETURN command_code;  
END execute_sql;  

PROCEDURE stop(timeout NUMBER DEFAULT 10) IS  
    status NUMBER;  
BEGIN  
    DBMS_PIPE.PACK_MESSAGE('STOP');  
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);  
    IF status <> 0 THEN  
        RAISE_APPLICATION_ERROR(-20030,  
            'stop: error while sending. status = ' || status);  
    END IF;  
    END stop;  
END daemon;  

daemon.pc Example. This is the code for the Pro*C daemon. You must precompile this using  
the Pro*C Precompiler, Version 1.5.x or later. You must also specify the USERID and SQLCHECK  
options, as the example contains embedded PL/SQL code.
To use a VARCHAR output host variable in a PL/SQL block, you must initialize the length component before entering the block.

```
proc iname=daemon userid=HR/<password> sqlcheck=semantics

Then C-compile and link in the normal way.

#include <stdio.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
    char *uid = "HR/<password>";
    int status;
    VARCHAR command[20];
    VARCHAR value[2000];
    VARCHAR return_name[30];
EXEC SQL END DECLARE SECTION;

void connect_error()
{
    char msg_buffer[512];
    int msg_length;
    int buffer_size = 512;

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlglm(msg_buffer, &buffer_size, &msg_length);
    printf("Daemon error while connecting:
    printf("\%s\n", msg_length, msg_buffer);
    printf("Daemon quitting.\n");
    exit(1);
}

void sql_error()
{
    char msg_buffer[512];
    int msg_length;
    int buffer_size = 512;

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlglm(msg_buffer, &buffer_size, &msg_length);
    printf("Daemon error while executing:\n");
    printf("\%s\n", msg_length, msg_buffer);
    printf("Daemon continuing.\n");
}

main()
{
    command.len = 20; /*initialize length components*/
    value.len = 2000;
    return_name.len = 30;
    EXEC SQL WHENEVER SQLERROR DO connect_error();
    EXEC SQL CONNECT :uid;
    printf("Daemon connected.\n");
```
EXEC SQL WHENEVER SQLERROR DO sql_error();
printf("Daemon waiting...\n");
while (1) {
  EXEC SQL EXECUTE
  BEGIN
    :status := DBMS_PIPE.RECEIVE_MESSAGE('daemon');
    IF :status = 0 THEN
      DBMS_PIPE.UNPACK_MESSAGE(:command);
    END IF;
  END;
  END-EXEC;
  IF (status == 0)
  {
    command.arr[command.len] = '\0';
    IF (!strcmp((char *) command.arr, "STOP"))
    {
      printf("Daemon exiting.\n");
      break;
    }
  }
  ELSE IF (!strcmp((char *) command.arr, "SYSTEM"))
  {
    EXEC SQL EXECUTE
    BEGIN
      DBMS_PIPE.UNPACK_MESSAGE(:return_name);
      DBMS_PIPE.UNPACK_MESSAGE(:value);
    END;
    END-EXEC;
    value.arr[value.len] = '\0';
    printf("Will execute system command '%s'\n", value.arr);
    status = system(value.arr);
  }
  EXEC SQL EXECUTE
  BEGIN
    DBMS_PIPE.PACK_MESSAGE('done');
    DBMS_PIPE.PACK_MESSAGE(:status);
    :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
  END;
  END-EXEC;
  IF (status)
  {
    printf("Daemon error while responding to system command.\n");
    printf(" status: %d\n", status);
  }
  ELSE IF (!strcmp((char *) command.arr, "SQL"))
  {
    EXEC SQL EXECUTE
    BEGIN
      DBMS_PIPE.UNPACK_MESSAGE(:return_name);
      DBMS_PIPE.UNPACK_MESSAGE(:value);
    END;
    END-EXEC;
    value.arr[value.len] = '\0';
    printf("Will execute sql command '%s'\n", value.arr);
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL EXECUTE IMMEDIATE :value;
    status = sqlca.sqlcode;
EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL EXECUTE
BEGIN
    DBMS_PIPE.PACK_MESSAGE('done');
    DBMS_PIPE.PACK_MESSAGE(:status);
    :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
END;
END-EXEC;

IF (status)
{
    printf("Daemon error while responding to sql command.");
    printf(" status: %d\n", status);
}
ELSE
{
    printf("Daemon error: invalid command '%s' received.\n", command.arr);
}
ELSE
{
    printf("Daemon error while waiting for signal.");
    printf(" status = %d\n", status);
}
EXEC SQL COMMIT WORK RELEASE;
exit(0);

Example 4: External Service Interface

Put the user-written 3GL code into an OCI or Precompiler program. The program connects to the database and executes PL/SQL code to read its request from the pipe, computes the result, and then executes PL/SQL code to send the result on a pipe back to the requestor.

Below is an example of a stock service request. The recommended sequence for the arguments to pass on the pipe for all service requests is:

```
protocol_version      VARCHAR2        - '1', 10 bytes or less
returnpipe            VARCHAR2        - 30 bytes or less
service               VARCHAR2        - 30 bytes or less
arg1                  VARCHAR2/NUMBER/DATE
... 
argn                  VARCHAR2/NUMBER/DATE
```

The recommended format for returning the result is:

```
success               VARCHAR2        - 'SUCCESS' if OK,
                      otherwise error message
arg1                  VARCHAR2/NUMBER/DATE
... 
argn                  VARCHAR2/NUMBER/DATE
```

The "stock price request server" would do, using OCI or PRO* (in pseudo-code):

```
<loop forever>
    BEGIN dbms_stock_server.get_request(:stocksymbol); END;
```
figure out price based on stocksymbol (probably from some radio signal), set error if can't find such a stock>
BEGIN dbms_stock_server.return_price(:error, :price); END;

A client would do:
BEGIN :price := stock_request('YOURCOMPANY'); end;

The stored procedure, \texttt{dbms\_stock\_server}, which is called by the preceding "stock price request server" is:

\begin{verbatim}
CREATE OR REPLACE PACKAGE dbms_stock_server IS
  PROCEDURE get_request(symbol OUT VARCHAR2);
  PROCEDURE return_price(errormsg IN VARCHAR2, price IN VARCHAR2);
END;

CREATE OR REPLACE PACKAGE BODY dbms_stock_server IS
  returnpipe VARCHAR2(30);
  PROCEDURE returnerror(reason VARCHAR2) IS
    s INTEGER;
    BEGIN
      dbms_pipe.pack_message(reason);
      s := dbms_pipe.send_message(returnpipe);
      IF s <> 0 THEN
        raise_application_error(-20000, 'Error:' || to_char(s) || ' sending on pipe');
      END IF;
    END;

  PROCEDURE get_request(symbol OUT VARCHAR2) IS
    protocol_version VARCHAR2(10);
    s INTEGER;
    service VARCHAR2(30);
    BEGIN
      s := dbms_pipe.receive_message('stock_service');
      IF s <> 0 THEN
        raise_application_error(-20000, 'Error:' || to_char(s) || 'reading pipe');
      END IF;
      dbms_pipe.unpack_message(protocol_version);
      IF protocol_version <> '1' THEN
        raise_application_error(-20000, 'Bad protocol: ' || protocol_version);
      END IF;
      dbms_pipe.unpack_message(returnpipe);
      dbms_pipe.unpack_message(service);
      IF service != 'getprice' THEN
        returnerror('Service ' || service || ' not supported');
      END IF;
      dbms_pipe.unpack_message(symbol);
    END;

  PROCEDURE return_price(errormsg in VARCHAR2, price in VARCHAR2) IS
    s INTEGER;
    BEGIN
      IF errormsg is NULL THEN
        dbms_pipe.pack_message('SUCCESS');
        dbms_pipe.pack_message(price);
      ELSE
        dbms_pipe.pack_message(errormsg);
      END IF;
    END;
\end{verbatim}
The procedure called by the client is:

```sql
CREATE OR REPLACE FUNCTION stock_request (symbol VARCHAR2)
    RETURN VARCHAR2 IS
    s        INTEGER;
    price    VARCHAR2(20);
    errormsg VARCHAR2(512);
BEGIN
    dbms_pipe.pack_message('1');  -- protocol version
    dbms_pipe.pack_message(dbms_pipe.unique_session_name); -- return pipe
    dbms_pipe.pack_message('getprice');
    dbms_pipe.pack_message(symbol);
    s := dbms_pipe.send_message('stock_service');
    IF s <> 0 THEN
        raise_application_error(-20000, 'Error:'||to_char(s)||
            ' sending on pipe');
    END IF;
    s := dbms_pipe.receive_message(dbms_pipe.unique_session_name);
    IF s <> 0 THEN
        raise_application_error(-20000, 'Error:'||to_char(s)||
            ' receiving on pipe');
    END IF;
    dbms_pipe.unpack_message(errormsg);
    IF errormsg <> 'SUCCESS' THEN
        raise_application_error(-20000, errormsg);
    END IF;
    dbms_pipe.unpack_message(price);
    RETURN price;
END;
```

You would typically only GRANT EXECUTE on DBMS_STOCK_SERVICE to the stock service application server, and would only GRANT EXECUTE on stock_request to those users allowed to use the service.
Table 119-2  DBMS_PIPE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_PIPE Function</td>
<td>Creates a pipe (necessary for private pipes)</td>
</tr>
<tr>
<td>NEXT_ITEM_TYPE Function</td>
<td>Returns datatype of next item in buffer</td>
</tr>
<tr>
<td>PACK_MESSAGE Procedures</td>
<td>Builds message in local buffer</td>
</tr>
<tr>
<td>PURGE Procedure</td>
<td>Purges contents of named pipe</td>
</tr>
<tr>
<td>RECEIVE_MESSAGE Function</td>
<td>Copies message from named pipe into local buffer</td>
</tr>
<tr>
<td>REMOVE_PIPE Function</td>
<td>Removes the named pipe</td>
</tr>
<tr>
<td>RESET_BUFFER Procedure</td>
<td>Purges contents of local buffer</td>
</tr>
<tr>
<td>SEND_MESSAGE Function</td>
<td>Sends message on named pipe: This implicitly creates a public pipe if the named pipe does not exist</td>
</tr>
<tr>
<td>UNIQUE_SESSION_NAME Function</td>
<td>Returns unique session name</td>
</tr>
<tr>
<td>UNPACK_MESSAGE Procedures</td>
<td>Accesses next item in buffer</td>
</tr>
</tbody>
</table>

119.7.1 CREATE_PIPE Function

This function explicitly creates a public or private pipe. If the `private` flag is `TRUE`, then the pipe creator is assigned as the owner of the private pipe.

Explicitly-created pipes can only be removed by calling `REMOVE_PIPE`, or by shutting down the instance.

Syntax

```sql
DBMS_PIPE.CREATE_PIPE (  
    pipename IN VARCHAR2,  
    maxpipesize IN INTEGER DEFAULT 8192,  
    private IN BOOLEAN DEFAULT TRUE)  
RETURN INTEGER;
```

Pragmas

```cpp
pragma restrict_references(create_pipe,WNDS,RNDS);
```

Parameters

Table 119-3  CREATE_PIPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Name of the pipe you are creating. You must use this name when you call SEND_MESSAGE and RECEIVE_MESSAGE. This name must be unique across the instance. Caution: Do not use pipe names beginning with ORAS. These are reserved for use by procedures provided by Oracle. Pipename should not be longer than 128 bytes, and is case insensitive. At this time, the name cannot contain Globalization Support characters.</td>
</tr>
</tbody>
</table>
Table 119-3    (Cont.) CREATE_PIPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxpipesize</td>
<td>The maximum size allowed for the pipe, in bytes. The total size of all of the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default maxpipesize is 8192 bytes. The maxpipesize for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of SEND_MESSAGE with larger values cause the maxpipesize to be increased. Callers with a smaller value use the existing, larger value.</td>
</tr>
<tr>
<td>private</td>
<td>Uses the default, TRUE, to create a private pipe. Public pipes can be implicitly created when you call SEND_MESSAGE.</td>
</tr>
</tbody>
</table>

Return Values

Table 119-4    CREATE_PIPE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBA/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.</td>
</tr>
<tr>
<td>ORA-23322</td>
<td>Failure due to naming conflict. If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.</td>
</tr>
</tbody>
</table>

Exceptions

Table 119-5    CREATE_PIPE Function Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null pipe name</td>
<td>Permission error: Pipe with the same name already exists, and you are not allowed to use it.</td>
</tr>
</tbody>
</table>

119.7.2 NEXT_ITEM_TYPE Function

This function determines the datatype of the next item in the local message buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call NEXT_ITEM_TYPE.

Syntax

```
DBMS_PIPE.NEXT_ITEM_TYPE
RETURN INTEGER;
```
Pragmas

pragma restrict_references(next_item_type,WNDS,RNDS);

Return Values

Table 119-6 NEXT_ITEM_TYPE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No more items</td>
</tr>
<tr>
<td>6</td>
<td>NUMBER</td>
</tr>
<tr>
<td>9</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>11</td>
<td>ROWID</td>
</tr>
<tr>
<td>12</td>
<td>DATE</td>
</tr>
<tr>
<td>23</td>
<td>RAW</td>
</tr>
</tbody>
</table>

119.7.3 PACK_MESSAGE Procedures

This procedure builds your message in the local message buffer.
To send a message, first make one or more calls to PACK_MESSAGE. Then, call SEND_MESSAGE to send the message in the local buffer on the named pipe.
The procedure is overloaded to accept items of type VARCHAR2, NCHAR, NUMBER, DATE, RAW and ROWID items. In addition to the data bytes, each item in the buffer requires one byte to indicate its type, and two bytes to store its length. One additional byte is needed to terminate the message. The overhead for all types other than VARCHAR is 4 bytes.

Syntax

DBMS_PIPE.PACK_MESSAGE (item IN VARCHAR2);
DBMS_PIPE.PACK_MESSAGE (item IN NCHAR);
DBMS_PIPE.PACK_MESSAGE (item IN NUMBER);
DBMS_PIPE.PACK_MESSAGE (item IN DATE);
DBMS_PIPE.PACK_MESSAGE_RAW (item IN RAW);
DBMS_PIPE.PACK_MESSAGE_ROWID (item IN ROWID);

Pragmas

pragma restrict_references(pack_message,WNDS,RNDS);
pragma restrict_references(pack_message_raw,WNDS,RNDS);
pragma restrict_references(pack_message_rowid,WNDS,RNDS);
Parameters

Table 119-7 PACK_MESSAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Item to pack into the local message buffer.</td>
</tr>
</tbody>
</table>

Usage Notes

In Oracle database version 8.x, the char-set-id (2 bytes) and the char-set-form (1 byte) are stored with each data item. Therefore, the overhead when using Oracle database version 8.x is 7 bytes.

When you call `SEND_MESSAGE` to send this message, you must indicate the name of the pipe on which you want to send the message. If this pipe already exists, then you must have sufficient privileges to access this pipe. If the pipe does not already exist, then it is created automatically.

Exceptions

`ORA-06558` is raised if the message buffer overflows (currently 4096 bytes). Each item in the buffer takes one byte for the type, two bytes for the length, plus the actual data. There is also one byte needed to terminate the message.

119.7.4 PURGE Procedure

This procedure empties the contents of the named pipe.

An empty implicitly-created pipe is aged out of the shared global area according to the least-recently-used algorithm. Thus, calling `PURGE` lets you free the memory associated with an implicitly-created pipe.

Syntax

```
DBMS_PIPE.PURGE (
    pipename  IN  VARCHAR2);
```

Pragmas

```
pragma restrict_references(purge,WNDS,RNDS);
```

Parameters

Table 119-8 PURGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Name of pipe from which to remove all messages. The local buffer may be overwritten with messages as they are discarded. Pipename should not be longer than 128 bytes, and is case-insensitive.</td>
</tr>
</tbody>
</table>
Usage Notes

Because PURGE calls RECEIVE_MESSAGE, the local buffer might be overwritten with messages as they are purged from the pipe. Also, you can receive an ORA-23322 (insufficient privileges) error if you attempt to purge a pipe with which you have insufficient access rights.

Exceptions

Permission error if pipe belongs to another user.

119.7.5 RECEIVE_MESSAGE Function

This function copies the message into the local message buffer.

Syntax

DBMS_PIPE.RECEIVE_MESSAGE (  
    pipename IN VARCHAR2,  
    timeout IN INTEGER DEFAULT maxwait)  
RETURN INTEGER;

Pragmas

pragma restrict_references(receive_message,WNDS,RNDS);

Parameters

Table 119-9 RECEIVE_MESSAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Name of the pipe on which you want to receive a message. Names beginning with ORA$ are reserved for use by Oracle.</td>
</tr>
<tr>
<td>timeout</td>
<td>Time to wait for a message, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days). A timeout of 0 lets you read without blocking.</td>
</tr>
</tbody>
</table>

Return Values

Table 119-10 RECEIVE_MESSAGE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>Timed out. If the pipe was implicitly-created and is empty, then it is removed.</td>
</tr>
<tr>
<td>2</td>
<td>Record in the pipe is too large for the buffer. (This should not happen.)</td>
</tr>
<tr>
<td>3</td>
<td>An interrupt occurred.</td>
</tr>
<tr>
<td>ORA-23322</td>
<td>User has insufficient privileges to read from the pipe.</td>
</tr>
</tbody>
</table>
Usage Notes

To receive a message from a pipe, first call `RECEIVE_MESSAGE`. When you receive a message, it is removed from the pipe; hence, a message can only be received once. For implicitly-created pipes, the pipe is removed after the last record is removed from the pipe.

If the pipe that you specify when you call `RECEIVE_MESSAGE` does not already exist, then Oracle implicitly creates the pipe and waits to receive the message. If the message does not arrive within a designated timeout interval, then the call returns and the pipe is removed.

After receiving the message, you must make one or more calls to `UNPACK_MESSAGE` to access the individual items in the message. The `UNPACK_MESSAGE` procedure is overloaded to unpack items of type `DATE`, `NUMBER`, `VARCHAR2`, and there are two additional procedures to unpack `RAW` and `ROWID` items. If you do not know the type of data that you are attempting to unpack, then call `NEXT_ITEM_TYPE` to determine the type of the next item in the buffer.

Exceptions

Table 119-11  `RECEIVE_MESSAGE` Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null pipe name</td>
<td>Permission error. Insufficient privilege to remove the record from the pipe. The pipe is owned by someone else.</td>
</tr>
</tbody>
</table>

119.7.6 RESET_BUFFER Procedure

This procedure resets the `PACK_MESSAGE` and `UNPACK_MESSAGE` positioning indicators to 0.

Because all pipes share a single buffer, you may find it useful to reset the buffer before using a new pipe. This ensures that the first time you attempt to send a message to your pipe, you do not inadvertently send an expired message remaining in the buffer.

Syntax

```
DBMS_PIPE.RESET_BUFFER;
```  

Pragmas

```
pragma restrict_references(reset_buffer,WNDS,RNDS);
```  

119.7.7 REMOVE_PIPE Function

This function removes explicitly-created pipes.

Pipes created implicitly by `SEND_MESSAGE` are automatically removed when empty. However, pipes created explicitly by `CREATE_PIPE` are removed only by calling `REMOVE_PIPE`, or by shutting down the instance. All unconsumed records in the pipe are removed before the pipe is deleted.

This is similar to calling `PURGE` on an implicitly-created pipe.
Syntax

DBMS_PIPE.REMOVE_PIPE (  
    pipename  IN  VARCHAR2)  
RETURN INTEGER;

Pragmas

pragma restrict_references(remove_pipe,WNDS,RNDS);

Parameters

Table 119-12  REMOVE_PIPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Name of pipe that you want to remove.</td>
</tr>
</tbody>
</table>

Return Values

Table 119-13  REMOVE_PIPE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>If the pipe does not exist, or if the pipe already exists and the user attempting to remove it is authorized to do so, then Oracle returns 0, indicating success, and any data remaining in the pipe is removed.</td>
</tr>
<tr>
<td>ORA-23322</td>
<td>Insufficient privileges.</td>
</tr>
<tr>
<td></td>
<td>If the pipe exists, but the user is not authorized to access the pipe, then Oracle signals error ORA-23322, indicating insufficient privileges.</td>
</tr>
</tbody>
</table>

Exceptions

Table 119-14  REMOVE_PIPE Function Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null pipe name</td>
<td>Permission error: Insufficient privilege to remove pipe. The pipe was created and is owned by someone else.</td>
</tr>
</tbody>
</table>

119.7.8 SEND_MESSAGE Function

This function sends a message on the named pipe.

The message is contained in the local message buffer, which was filled with calls to PACK_MESSAGE. You can create a pipe explicitly using CREATE_PIPE, otherwise, it is created implicitly.

Syntax

DBMS_PIPE.SEND_MESSAGE (  
    pipename     IN VARCHAR2,  
    timeout      IN INTEGER DEFAULT MAXWAIT,  
    ...)
```sql
maxpipesize IN INTEGER DEFAULT 8192)
RETURN INTEGER;
```

**Pragmas**

```sql
pragma restrict_references(send_message,WNDS,RNDS);
```

**Parameters**

**Table 119-15  SEND_MESSAGE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipename</td>
<td>Name of the pipe on which you want to place the message. If you are using an explicit pipe, then this is the name that you specified when you called CREATE_PIPE. Caution: Do not use pipe names beginning with ‘$ORA$’. These names are reserved for use by procedures provided by Oracle. Pipename should not be longer than 128 bytes, and is case-insensitive. At this time, the name cannot contain Globalization Support characters.</td>
</tr>
<tr>
<td>timeout</td>
<td>Time to wait while attempting to place a message on a pipe, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days).</td>
</tr>
<tr>
<td>maxpipesize</td>
<td>Maximum size allowed for the pipe, in bytes. The total size of all the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default is 8192 bytes. The maxpipesize for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of SEND_MESSAGE with larger values cause the maxpipesize to be increased. Callers with a smaller value simply use the existing, larger value. Specifying maxpipesize as part of the SEND_MESSAGE procedure eliminates the need for a separate call to open the pipe. If you created the pipe explicitly, then you can use the optional maxpipesize parameter to override the creation pipe size specifications.</td>
</tr>
</tbody>
</table>

**Return Values**

**Table 119-16  SEND_MESSAGE Function Return Values**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBS/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.</td>
</tr>
</tbody>
</table>
Table 119-16 (Cont.) SEND_MESSAGE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1      | Timed out.  
This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used. If the pipe was implicitly-created and is empty, then it is removed. |
| 3      | An interrupt occurred.  
If the pipe was implicitly created and is empty, then it is removed. |
| ORA-23322 | Insufficient privileges.  
If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict. |

Exceptions

Table 119-17 SEND_MESSAGE Function Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null pipe name</td>
<td>Permission error. Insufficient privilege to write to the pipe. The pipe is private and owned by someone else.</td>
</tr>
</tbody>
</table>

119.7.9 UNIQUE_SESSION_NAME Function

This function receives a name that is unique among all of the sessions that are currently connected to a database.

Multiple calls to this function from the same session always return the same value. You might find it useful to use this function to supply the PIPENAME parameter for your SEND_MESSAGE and RECEIVE_MESSAGE calls.

Syntax

DBMS_PIPE.UNIQUE_SESSION_NAME
RETURN VARCHAR2;

Pragmas

pragma restrict_references(unique_session_name,WNDS,RNDS,WNPS);

Return Values

This function returns a unique name. The returned name can be up to 30 bytes.

119.7.10 UNPACK_MESSAGE Procedures

This procedure retrieves items from the buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call UNPACK_MESSAGE.
The UNPACK_MESSAGE procedure is overloaded to return items of type VARCHAR2, NCHAR, NUMBER, or DATE. There are two additional procedures to unpack RAW and ROWID items.

Syntax

```
DBMS_PIPE.UNPACK_MESSAGE (item  OUT VARCHAR2);
DBMS_PIPE.UNPACK_MESSAGE (item  OUT NCHAR);
DBMS_PIPE.UNPACK_MESSAGE (item  OUT NUMBER);
DBMS_PIPE.UNPACK_MESSAGE (item  OUT DATE);
DBMS_PIPE.UNPACK_MESSAGE_RAW (item  OUT RAW);
DBMS_PIPE.UNPACK_MESSAGE_ROWID (item  OUT ROWID);
```

Pragmas

```
pragma restrict_references(unpack_message,WNDS,RNDS);
pragma restrict_references(unpack_message_raw,WNDS,RNDS);
pragma restrict_references(unpack_message_rowid,WNDS,RNDS);
```

Parameters

Table 119-18  UNPACK_MESSAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Argument to receive the next unpacked item from the local message buffer.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-06556 or 06559 are generated if the buffer contains no more items, or if the item is not of the same type as that requested.
The `DBMS_PLSQL_CODE_COVERAGE` package provides an interface for the collection of code coverage data of PL/SQL applications at the basic block level.

This chapter contains the following topics:

- `DBMS_PLSQL_CODE_COVERAGE` Overview
- `DBMS_PLSQL_CODE_COVERAGE` Security Model
- Summary of `DBMS_PLSQL_CODE_COVERAGE` Subprograms

### 120.1 DBMS_PLSQL_CODE_COVERAGE Overview

The `DBMS_PLSQL_CODE_COVERAGE` package provides an interface for collecting code coverage information at the basic block level of PL/SQL applications. A basic block refers to a single entry single exit block of PL/SQL code. PL/SQL developers want to know how well their test infrastructure exercised their code. The coverage tables are created using the `CREATE_COVERAGE_TABLES` procedure.

A typical code coverage run in a session involves calls to:

- START_COVERAGE
- Run PL/SQL code
- STOP_COVERAGE

**See Also:**

- Oracle Database Development Guide for more information about using PL/SQL basic block coverage to maintain quality
- Oracle Database PL/SQL Language Reference for the `COVERAGE` PRAGMA syntax and semantics
- Oracle Database PL/SQL Language Reference for more information about the `PLSQL_OPTIMIZE_LEVEL` compilation parameter

### 120.2 DBMS_PLSQL_CODE_COVERAGE Security Model

The user must have `EXECUTE` privilege on the `DBMS_PLSQL_CODE_COVERAGE` package.

The user must have `CREATE` privilege on the unit to collect coverage information about this unit.

Coverage information is not collected for wrapped units.
120.3 Summary of DBMS_PLSQL_CODE_COVERAGE Subprograms

This table lists the DBMS_PLSQL_CODE_COVERAGE subprograms and briefly describes them.

Table 120-1  DBMS_PLSQL_CODE_COVERAGE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_COVERAGE_TABLES Procedure</td>
<td>Creates coverage tables</td>
</tr>
<tr>
<td>START_COVERAGE Function</td>
<td>Starts the coverage data collection in the user’s session and returns the RUN_ID</td>
</tr>
<tr>
<td>STOP_COVERAGE Procedure</td>
<td>Ends the current coverage run</td>
</tr>
</tbody>
</table>

120.3.1 CREATE_COVERAGE_TABLES Procedure

This procedure creates the tables used for coverage data collection.

Syntax

DBMS_PLSQL_CODE_COVERAGE.CREATE_COVERAGE_TABLES (    
    FORCE_IT     IN BOOLEAN DEFAULT FALSE);    

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE_IT</td>
<td>The default is to raise an error if the coverage tables already exist. If set to TRUE, the tables are dropped silently if the tables already exist, and new tables are created.</td>
</tr>
</tbody>
</table>

Exceptions

Table 120-2  CREATE_COVERAGE_TABLES Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COVERAGE_ERROR</td>
<td>The FORCE_IT parameter is FALSE and the tables already exist.</td>
</tr>
</tbody>
</table>

120.3.2 START_COVERAGE Function

This function starts the coverage data collection in the user’s session and returns a unique identifier RUN_ID for the run.
Syntax

DBMS_PLSQL_CODE_COVERAGE.START_COVERAGE (  
    run_comment IN VARCHAR2)  
RETURN NUMBER;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_comment</td>
<td>Allows the user to name a run and identify the test.</td>
</tr>
</tbody>
</table>

120.3.3 STOP_COVERAGE Procedure

This procedure ends the current coverage run.

Syntax

DBMS_PLSQL_CODE_COVERAGE.STOP_COVERAGE;

Exceptions

Table 120-3 STOP_COVERAGE Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COVERAGE_ERROR</td>
<td>An error is raised if the coverage tables do not exist.</td>
</tr>
</tbody>
</table>
Data mining can discover useful information buried in vast amounts of data. However, it is often the case that both the programming interfaces and the data mining expertise required to obtain these results are too complex for use by the wide audiences that can obtain benefits from using Oracle Data Mining.

The `DBMS_PREDICTIVE_ANALYTICS` package addresses both of these complexities by automating the entire data mining process from data preprocessing through model building to scoring new data. This package provides an important tool that makes data mining possible for a broad audience of users, in particular, business analysts.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Summary of `DBMS_PREDICTIVE_ANALYTICS` Subprograms**

### 121.1 DBMS_PREDICTIVE_ANALYTICS Overview

`DBMS_PREDICTIVE_ANALYTICS` automates parts of the data mining process.

Data mining, according to a commonly used process model, requires the following steps:

1. Understand the business problem.
2. Understand the data.
3. Prepare the data for mining.
4. Create models using the prepared data.
5. Evaluate the models.
6. Deploy and use the model to score new data.

`DBMS_PREDICTIVE_ANALYTICS` automates parts of step 3 — 5 of this process.

Predictive analytics procedures analyze and prepare the input data, create and test mining models using the input data, and then use the input data for scoring. The results of scoring are returned to the user. The models and supporting objects are not preserved after the operation completes.

### 121.2 DBMS_PREDICTIVE_ANALYTICS Security Model

The `DBMS_PREDICTIVE_ANALYTICS` package is owned by user **SYS** and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers’ rights (run with the privileges of the current user).

The `DBMS_PREDICTIVE_ANALYTICS` package exposes APIs which are leveraged by the Oracle Data Mining option. Users who wish to invoke procedures in this package require the **CREATE**
MINING_MODEL system privilege (as well as the CREATE TABLE and CREATE VIEW system privilege).

121.3 Summary of DBMS_PREDICTIVE_ANALYTICS Subprograms

This table lists and briefly describes the DBMS_PREDICTIVE_ANALYTICS package subprograms.

Table 121-1  DBMS_PREDICTIVE_ANALYTICS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN Procedure</td>
<td>Ranks attributes in order of influence in explaining a target column.</td>
</tr>
<tr>
<td>PREDICT Procedure</td>
<td>Predicts the value of a target column based on values in the input data.</td>
</tr>
<tr>
<td>PROFILE Procedure</td>
<td>Generates rules that identify the records that have the same target value.</td>
</tr>
</tbody>
</table>

121.3.1 EXPLAIN Procedure

The EXPLAIN procedure identifies the attributes that are important in explaining the variation in values of a target column.

The input data must contain some records where the target value is known (not NULL). These records are used by the procedure to train a model that calculates the attribute importance.

Note:

EXPLAIN supports DATE and TIMESTAMP datatypes in addition to the numeric, character, and nested datatypes supported by Oracle Data Mining models.

Data requirements for Oracle Data Mining are described in Oracle Data Mining User's Guide.

The EXPLAIN procedure creates a result table that lists the attributes in order of their explanatory power. The result table is described in the Usage Notes.

Syntax

DBMS_PREDICTIVE_ANALYTICS.EXPLAIN (  
  data_table_name IN VARCHAR2,  
  explain_column_name IN VARCHAR2,  
  result_table_name IN VARCHAR2,  
  data_schema_name IN VARCHAR2 DEFAULT NULL);
Parameters

Table 121-2  EXPLAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_table_name</td>
<td>Name of input table or view</td>
</tr>
<tr>
<td>explain_column_name</td>
<td>Name of the column to be explained</td>
</tr>
<tr>
<td>result_table_name</td>
<td>Name of the table where results are saved</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.</td>
</tr>
</tbody>
</table>

Usage Notes

The EXPLAIN procedure creates a result table with the columns described in Table 121-3.

Table 121-3  EXPLAIN Procedure Result Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE_NAME</td>
<td>VARCHAR2(30)</td>
<td>Name of a column in the input data; all columns except the explained column are listed in the result table.</td>
</tr>
<tr>
<td>EXPLANATORY_VALUE</td>
<td>NUMBER</td>
<td>Value indicating how useful the column is for determining the value of the explained column. Higher values indicate greater explanatory power. Value can range from 0 to 1. An individual column's explanatory value is independent of other columns in the input table. The values are based on how strong each individual column correlates with the explained column. The value is affected by the number of records in the input table, and the relations of the values of the column to the values of the explain column. An explanatory power value of 0 implies there is no useful correlation between the column's values and the explain column's values. An explanatory power of 1 implies perfect correlation; such columns should be eliminated from consideration for PREDICT. In practice, an explanatory power equal to 1 is rarely returned.</td>
</tr>
<tr>
<td>RANK</td>
<td>NUMBER</td>
<td>Ranking of explanatory power. Rows with equal values for explanatory_value have the same rank. Rank values are not skipped in the event of ties.</td>
</tr>
</tbody>
</table>

Example

The following example performs an EXPLAIN operation on the SUPPLEMENTARY_DEMOGRAPHICS table of Sales History.

```sql
--Perform EXPLAIN operation
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
        data_table_name  => 'supplementary_demographics',
        explain_column_name  => 'home_theater_package',
        result_table_name    => 'demographics_explain_result');
END;
/
--Display results
SELECT * FROM demographics_explain_result;
```
The results show that **Y_BOX_GAMES**, **YRS_RESIDENCE**, and **HOUSEHOLD_SIZE** are the best predictors of **HOME_THEATER_PACKAGE**.

### 121.3.2 PREDICT Procedure

The **PREDICT** procedure predicts the values of a target column.

The input data must contain some records where the target value is known (not NULL). These records are used by the procedure to train and test a model that makes the predictions.

**Note:**

**PREDICT** supports **DATE** and **TIMESTAMP** datatypes in addition to the numeric, character, and nested datatypes supported by Oracle Data Mining models.

Data requirements for Oracle Data Mining are described in *Oracle Data Mining User’s Guide*.

The **PREDICT** procedure creates a result table that contains a predicted target value for every record. The result table is described in the Usage Notes.

**Syntax**

```sql
DBMS_PREDICTIVE_ANALYTICS.PREDICT (
    accuracy OUT NUMBER,
    data_table_name IN VARCHAR2,
    case_id_column_name IN VARCHAR2,
    target_column_name IN VARCHAR2,
    result_table_name IN VARCHAR2,
    data_schema_name IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 121-4  PREDICT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>Output parameter that returns the predictive confidence, a measure of the accuracy of the predicted values. The predictive confidence for a categorical target is the most common target value; the predictive confidence for a numerical target is the mean.</td>
</tr>
<tr>
<td>data_table_name</td>
<td>Name of the input table or view.</td>
</tr>
<tr>
<td>case_id_column_name</td>
<td>Name of the column that uniquely identifies each case (record) in the input data.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Name of the column to predict.</td>
</tr>
<tr>
<td>result_table_name</td>
<td>Name of the table where results will be saved.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.</td>
</tr>
</tbody>
</table>

Usage Notes

The PREDICT procedure creates a result table with the columns described in Table 121-5.

Table 121-5  PREDICT Procedure Result Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case ID column name</td>
<td>VARCHAR2 or NUMBER</td>
<td>The name of the case ID column in the input data.</td>
</tr>
<tr>
<td>PREDICTION</td>
<td>VARCHAR2 or NUMBER</td>
<td>The predicted value of the target column for the given case.</td>
</tr>
<tr>
<td>PROBABILITY</td>
<td>NUMBER</td>
<td>For classification (categorical target), the probability of the prediction. For regression problems (numerical target), this column contains NULL.</td>
</tr>
</tbody>
</table>

Note:

Make sure that the name of the case ID column is not 'PREDICTION' or 'PROBABILITY'.

Predictions are returned for all cases whether or not they contained target values in the input.

Predicted values for known cases may be interesting in some situations. For example, you could perform deviation analysis to compare predicted values and actual values.

Example

The following example performs a PREDICT operation and displays the first 10 predictions. The results show an accuracy of 79% in predicting whether each customer has an affinity card.
--Perform PREDICT operation
DECLARE
  v_accuracy NUMBER(10,9);
BEGIN
  DBMS_PREDICTIVE_ANALYTICS.PREDICT(
    accuracy             => v_accuracy,
    data_table_name      => 'supplementary_demographics',
    case_id_column_name  => 'cust_id',
    target_column_name   => 'affinity_card',
    result_table_name    => 'pa_demographics_predict_result');
  DBMS_OUTPUT.PUT_LINE('Accuracy = ' || v_accuracy);
END;
/
Accuracy = .788696903

--Display results
SELECT * FROM pa_demographics_predict_result WHERE rownum < 10;

<table>
<thead>
<tr>
<th>CUST_ID</th>
<th>PREDICTION</th>
<th>PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>101501</td>
<td>1</td>
<td>.834069848</td>
</tr>
<tr>
<td>101502</td>
<td>0</td>
<td>.991269965</td>
</tr>
<tr>
<td>101503</td>
<td>0</td>
<td>.99978311</td>
</tr>
<tr>
<td>101504</td>
<td>1</td>
<td>.971643388</td>
</tr>
<tr>
<td>101505</td>
<td>1</td>
<td>.541754127</td>
</tr>
<tr>
<td>101506</td>
<td>0</td>
<td>.803719133</td>
</tr>
<tr>
<td>101507</td>
<td>0</td>
<td>.999999303</td>
</tr>
<tr>
<td>101508</td>
<td>0</td>
<td>.999999987</td>
</tr>
<tr>
<td>101509</td>
<td>0</td>
<td>.999953074</td>
</tr>
</tbody>
</table>

121.3.3 PROFILE Procedure

The PROFILE procedure generates rules that describe the cases (records) from the input data.

For example, if a target column CHURN has values 'Yes' and 'No', PROFILE generates a set of rules describing the expected outcomes. Each profile includes a rule, record count, and a score distribution.

The input data must contain some cases where the target value is known (not NULL). These cases are used by the procedure to build a model that calculates the rules.

Note:

PROFILE does not support nested types or dates.

Data requirements for Oracle Data Mining are described in Oracle Data Mining User's Guide.

The PROFILE procedure creates a result table that specifies rules (profiles) and their corresponding target values. The result table is described in the Usage Notes.
Syntax

DBMS_PREDICTIVE_ANALYTICS.PROFILE (
    data_table_name           IN VARCHAR2,
    target_column_name        IN VARCHAR2,
    result_table_name         IN VARCHAR2,
    data_schema_name          IN VARCHAR2 DEFAULT NULL);

Parameters

Table 121-6   PROFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_table_name</td>
<td>Name of the table containing the data to be analyzed.</td>
</tr>
<tr>
<td>target_column_name</td>
<td>Name of the target column.</td>
</tr>
<tr>
<td>result_table_name</td>
<td>Name of the table where the results will be saved.</td>
</tr>
<tr>
<td>data_schema_name</td>
<td>Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.</td>
</tr>
</tbody>
</table>

Usage Notes

The PROFILE procedure creates a result table with the columns described in Table 121-7.

Table 121-7   PROFILE Procedure Result Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE_ID</td>
<td>NUMBER</td>
<td>A unique identifier for this profile (rule).</td>
</tr>
<tr>
<td>RECORD_COUNT</td>
<td>NUMBER</td>
<td>The number of records described by the profile.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>SYS.XMLTYPE</td>
<td>The profile rule. See &quot;XML Schema for Profile Rules&quot;.</td>
</tr>
</tbody>
</table>

XML Schema for Profile Rules

The DESCRIPTION column of the result table contains XML that conforms to the following XSD:

```xml
<xs:element name="SimpleRule">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="PREDICATE"/>
      <xs:element ref="ScoreDistribution" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="optional"/>
    <xs:attribute name="score" type="xs:string" use="required"/>
    <xs:attribute name="recordCount" type="NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>
```

Example

This example generates a rule describing customers who are likely to use an affinity card (target value is 1) and a set of rules describing customers who are not likely to use an affinity card (target value is 0). The rules are based on only two predictors: education and occupation.
This example generates eight rules in the result table \textit{profile\_result}. Seven of the rules suggest a target value of 0; one rule suggests a target value of 1. The \texttt{score} attribute on a rule identifies the target value.

This \textsc{select} statement returns all the rules in the result table.

\begin{verbatim}
SELECT a.profile_id, a.record_count, a.description.getstringval()
FROM profile_result a;
\end{verbatim}

This \textsc{select} statement returns the rules for a target value of 0.

\begin{verbatim}
SELECT *
FROM profile_result t
WHERE extractvalue(t.description, '/SimpleRule/@score') = 0;
\end{verbatim}

The eight rules generated by this example are displayed as follows.

\begin{verbatim}
<SimpleRule id="1" score="0" recordCount="443">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
    </Array>
  </SimpleSetPredicate>
  <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
    <Array type="string">"< Bach." "Assoc-V" "HS-grad"
  </Array>
  </SimpleSetPredicate>
</CompoundPredicate>
<ScoreDistribution value="0" recordCount="297" />
<ScoreDistribution value="1" recordCount="146" />
</SimpleRule>

<SimpleRule id="2" score="0" recordCount="18">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
    </Array>
  </SimpleSetPredicate>
  <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
    <Array type="string">"10th" "11th" "12th" "1st-4th" "5th-6th" "7th-8th" "9th" "Presch."
  </Array>
  </SimpleSetPredicate>
</CompoundPredicate>
<ScoreDistribution value="0" recordCount="18" />
\end{verbatim}
Summary of DBMS_PREDICTIVE_ANALYTICS Subprograms

Chapter 121

121-10

...
The DBMS_PREPROCESSOR package provides an interface to print or retrieve the source
text of a PL/SQL unit in its post-processed form.

This package contains the following topics:

- Overview
- Operating Notes
- Data Structures
- Summary of DBMS_PREPROCESSOR Subprograms

### 122.1 DBMS_PREPROCESSOR Overview

It is necessary to first understand the three styles of subprograms, in order to understand
how DBMS_PREPROCESSOR works.

The following are the three styles of subprograms:

1. Subprograms that take a schema name, a unit type name, and the unit name.
2. Subprograms that take a `VARCHAR2` string which contains the source text of an arbitrary
   PL/SQL compilation unit.
3. Subprograms that take a `VARCHAR2` index-by table which contains the segmented source
   text of an arbitrary PL/SQL compilation unit.

Subprograms of the first style are used to print or retrieve the post-processed source text of a
stored PL/SQL unit. The user must have the privileges necessary to view the original source
text of this unit. The user must also specify the schema in which the unit is defined, the type
of the unit, and the name of the unit. If the schema is null, then the current user schema is
used. If the status of the stored unit is `VALID` and the user has the required privilege, then the
post-processed source text is guaranteed to be the same as that of the unit the last time it
was compiled.

Subprograms of the second or third style are used to generate post-processed source text in
the current user schema. The source text is passed in as a single `VARCHAR2` string in the
second style, or as a `VARCHAR2` index-by table in the third style. The source text can represent
an arbitrary PL/SQL compilation unit. A typical usage is to pass the source text of an
anonymous block and generate its post-processed source text in the current user schema.
The third style can be useful when the source text exceeds the `VARCHAR2` length limit.

### 122.2 DBMS_PREPROCESSOR Operating Notes

These notes explain how DBMS_PREPROCESSOR works with the three subprogram styles.

- For subprograms of the first style, the status of the stored PL/SQL unit does not need to
  be `VALID`. Likewise, the source text passed in as a `VARCHAR2` string or a `VARCHAR2` index-
  by table may contain compile time errors. If errors are found when generating the post-
processed source, the error message text will also appear at the end of the post-
processed source text. In some cases, the preprocessing can be aborted because
of errors. When this happens, the post-processed source text will appear to be
incomplete and the associated error message can help to indicate that an error
has occurred during preprocessing.

• For subprograms of the second or third style, the source text can represent any
arbitrary PL/SQL compilation unit. However, the source text of a valid PL/SQL
compilation unit cannot include commonly used prefixes such as CREATE OR
REPLACE. In general, the input source should be syntactically prepared in a way as
if it were obtained from the ALL_SOURCE view. The following list gives some
examples of valid initial syntax for some PL/SQL compilation units.

  anonymous block   (BEGIN | DECLARE) ...
  package           PACKAGE <name> ...
  package body      PACKAGE BODY <name> ...
  procedure         PROCEDURE <name> ...
  function          FUNCTION <name> ...
  type              TYPE <name> ...
  type body         TYPE BODY <name> ...
  trigger           (BEGIN | DECLARE) ...

If the source text represents a named PL/SQL unit that is valid, that unit will not be
created after its post-processed source text is generated.

• If the text of a wrapped PL/SQL unit is obtained from the ALL_SOURCE view, the
keyword WRAPPED always immediately follows the name of the unit, as in this
example:

  PROCEDURE "some proc" WRAPPED
  a000000
  b2
  ...

If such source text is presented to one of the
GET_POST_PROCESSED_SOURCE Functions or to one of the
PRINT_POST_PROCESSED_SOURCE Procedures, the exception
DBMS_PREPROCESSOR.WRAPPED_INPUT is raised.

122.3 DBMS_PREPROCESSOR Data Structures

The DBMS_PREPROCESSOR package defines a TABLE type.

Table Types

SOURCE_LINES_T Table Type

122.3.1 DBMS_PREPROCESSOR SOURCE_LINES_T Table Type

This table type stores lines of post-processed source text. It is used to hold PL/SQL
source text both before and after it is processed. It is especially useful in cases in
which the amount of text exceeds 32K.

Syntax

TYPE source_lines_t IS
  TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
122.4 Summary of DBMS_PREPROCESSOR Subprograms

This table lists the DBMS_PREPROCESSOR subprograms and briefly describes them.

Table 122-1 DBMS_PREPROCESSOR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_POST_PROCESSED_SOURCE Functions</td>
<td>Returns the post-processed source text</td>
</tr>
<tr>
<td>PRINT_POST_PROCESSED_SOURCE Procedures</td>
<td>Prints post-processed source text</td>
</tr>
</tbody>
</table>

122.4.1 GET_POST_PROCESSED_SOURCE Functions

This overloaded function returns the post-processed source text. The different functionality of each form of syntax is presented along with the definition.

Syntax

Returns post-processed source text of a stored PL/SQL unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (  
  object_type    IN VARCHAR2,  
  schema_name    IN VARCHAR2,  
  object_name    IN VARCHAR2)  
RETURN source_lines_t;
```

Returns post-processed source text of a compilation unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (  
  source        IN VARCHAR2)  
RETURN source_lines_t;
```

Returns post-processed source text of an INDEX-BY table containing the source text of the compilation unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (  
  source        IN source_lines_t)  
RETURN source_lines_t;
```

Parameters

Table 122-2 GET_POST_PROCESSED_SOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>Must be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER. Case sensitive.</td>
</tr>
<tr>
<td>schema_name</td>
<td>The schema name. Case insensitive unless a quoted identifier is used. If NULL, use current schema.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object. The object_type is always case insensitive. Case insensitive unless a quoted identifier is used.</td>
</tr>
<tr>
<td>source</td>
<td>The source text of the compilation unit</td>
</tr>
</tbody>
</table>
Table 122-2  (Cont.) GET_POST_PROCESSED_SOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_lines_t</td>
<td>INDEX-BY table containing the source text of the compilation unit. The source text is a concatenation of all the non-NULL INDEX-BY table elements in ascending index order.</td>
</tr>
</tbody>
</table>

Return Values

The function returns an INDEX-BY table containing the lines of the post-processed source text starting from index 1.

Usage Notes

- Newline characters are not removed.
- Each line in the post-processed source text is mapped to a row in the INDEX-BY table.
- In the post-processed source, unselected text will have blank lines.

Exceptions

Table 122-3  GET_POST_PROCESSED_SOURCE Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-24234</td>
<td>Insufficient privileges or object does not exist</td>
</tr>
<tr>
<td>ORA-24235</td>
<td>Bad value for object type. Should be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE BODY or TRIGGER.</td>
</tr>
<tr>
<td>ORA-24236</td>
<td>The source text is empty</td>
</tr>
<tr>
<td>ORA-00931</td>
<td>Missing identifier. The object_name should not be NULL.</td>
</tr>
<tr>
<td>ORA-06502</td>
<td>Numeric or value error:</td>
</tr>
<tr>
<td></td>
<td>• Character string buffer too small</td>
</tr>
<tr>
<td></td>
<td>• A line is too long (&gt; 32767 bytes)</td>
</tr>
</tbody>
</table>

122.4.2 PRINT_POST_PROCESSED_SOURCE Procedures

This overloaded procedure calls DBMS_OUTPUT.PUT_LINE to let you view post-processed source text. The different functionality of each form of syntax is presented along with the definition.

Syntax

Prints post-processed source text of a stored PL/SQL unit:

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (  
  object_type    IN VARCHAR2,  
  schema_name    IN VARCHAR2,  
  object_name    IN VARCHAR2);  
```

Prints post-processed source text of a compilation unit:
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (source IN VARCHAR2);

Prints post-processed source text of an INDEX-BY table containing the source text of the compilation unit:

DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (source IN source_lines_t);

Parameters

Table 122-4 PRINT_POST_PROCESSED_SOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>Must be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER. Case sensitive.</td>
</tr>
<tr>
<td>schema_name</td>
<td>The schema name. Case insensitive unless a quoted identifier is used. If NULL, use current schema.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object. The object_type is always case insensitive. Case insensitive unless a quoted identifier is used.</td>
</tr>
<tr>
<td>source</td>
<td>The source text of the compilation unit</td>
</tr>
<tr>
<td>source_lines_t</td>
<td>INDEX-BY table containing the source text of the compilation unit. The source text is a concatenation of all the non-NULL INDEX-BY table elements in ascending index order.</td>
</tr>
</tbody>
</table>

Exceptions

Table 122-5 PRINT_POST_PROCESSED_SOURCE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-24234</td>
<td>Insufficient privileges or object does not exist</td>
</tr>
<tr>
<td>ORA-24235</td>
<td>Bad value for object type. Should be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER.</td>
</tr>
<tr>
<td>ORA-24236</td>
<td>The source text is empty</td>
</tr>
<tr>
<td>ORA-00931</td>
<td>Missing identifier. The object_name should not be NULL.</td>
</tr>
<tr>
<td>ORA-06502</td>
<td>Numeric or value error:</td>
</tr>
<tr>
<td></td>
<td>• Character string buffer too small</td>
</tr>
<tr>
<td></td>
<td>• A line is too long (&gt; 32767 bytes)</td>
</tr>
</tbody>
</table>

Usage Notes

The index-by table may contain holes. NULL elements are ignored when doing the concatenation.
DBMS_PRIVILEGE_CAPTURE

The DBMS_PRIVILEGE_CAPTURE package provides an interface to database privilege analysis.

See Also:
Oracle Database Vault Administrator's Guide regarding on how to analyze the use of privilege grants

This chapter contains the following topics:

• Overview
• Security Model
• Constants
• Examples
• Summary of DBMS_PRIVILEGE_CAPTURE Subprograms

123.1 DBMS_PRIVILEGE_CAPTURE Overview

Database privilege analysis enables you to create a policy that records the usage of system and object privileges that have been granted to users. You then can determine the privileges that your users are using and not using. From there, you can revoke any unused privileges, thereby reducing the number of excess privilege grants for users.

By analyzing the privileges that users must have to perform specific tasks, privilege analysis policies help you to achieve a least privilege model for your users.

123.2 DBMS_PRIVILEGE_CAPTURE Security Model

The privilege analysis administrator role, CAPTURE_ADMIN, is granted EXECUTE permission on the DBMS_PRIVILEGE_CAPTURE package by default.

The CAPTURE_ADMIN role is granted to the DBA role during database installation.

123.3 DBMS_PRIVILEGE_CAPTURE Constants

The DBMS_PRIVILEGE_CAPTURE package defines several enumerated constants for specifying parameter values.
Table 123-1  DBMS_PRIVILEGE_CAPTURE Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_DATABASE</td>
<td>1</td>
<td>NUMBER</td>
<td>Analyzes all privilege use, except privileges used by the SYS user.</td>
</tr>
<tr>
<td>G_ROLE</td>
<td>2</td>
<td>NUMBER</td>
<td>Analyzes privilege use for the specified roles.</td>
</tr>
<tr>
<td>G_CONTEXT</td>
<td>3</td>
<td>NUMBER</td>
<td>Analyzes privilege use when the condition parameter evaluates to true.</td>
</tr>
<tr>
<td>G_ROLE_AND_CONTEXT</td>
<td>4</td>
<td>NUMBER</td>
<td>Analyzes privilege use for the specified roles when the condition parameter evaluates to true.</td>
</tr>
</tbody>
</table>

123.4 DBMS_PRIVILEGE_CAPTURE Examples

These examples illustrate using the `DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE` procedure to create various types of privilege analysis, like database analysis, role analysis, and context-specific analysis. The examples also illustrate combining different conditions in context-specific analysis.

```sql
-- Create a database privilege analysis policy
BEGIN
DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
    name => 'all_priv_analysis_pol',
    description => 'database-wide policy to analyze all privileges',
    type => DBMS_PRIVILEGE_CAPTURE.G_DATABASE);
END;

-- Create a privilege analysis policy to analyze privileges from the role PUBLIC
BEGIN
DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
    name => 'pub_analysis_pol',
    description => 'Policy to record privilege use by PUBLIC',
    type => DBMS_PRIVILEGE_CAPTURE.G_ROLE,
    roles => role_name_list('PUBLIC'));
END;

-- Create a policy to analyze privileges from the application module, "Account Payable"
BEGIN
DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
    name => 'acc_pay_analysis_pol',
    type => DBMS_PRIVILEGE_CAPTURE.G_CONTEXT,
    condition => 'SYS_CONTEXT(''USERENV'', ''MODULE'') = ''Account Payable'');
END;

-- Create a policy that records privileges for session user APPS when running the application module "Account Payable"
BEGIN
DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
    name => 'acc_pay_analysis_pol',
    type => DBMS_PRIVILEGE_CAPTURE.G_CONTEXT,
    condition => 'SYS_CONTEXT(''USERENV'', ''MODULE'') = ''Account Payable'' AND SYS_CONTEXT(''USERENV'', ''SESSION_USER'') = ''APPS'');
END;
```
123.5 Summary of DBMS_PRIVILEGE_CAPTURE Subprograms

This table lists and briefly describes the DBMS_PRIVILEGE_CAPTURE package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPTURE_DEPENDENCY_PRIVS</td>
<td>Captures the privileges that are used by definer’s rights and invoker’s rights PL/SQL program units for compilation.</td>
</tr>
<tr>
<td>CREATE_CAPTURE Procedure</td>
<td>Creates a policy that specifies the conditions for analyzing privilege use.</td>
</tr>
<tr>
<td>DELETE_RUN Procedure</td>
<td>Deletes a privilege analysis capture run</td>
</tr>
<tr>
<td>DISABLE_CAPTURE Procedure</td>
<td>Stops the recording of privilege use for a specified privilege analysis policy</td>
</tr>
<tr>
<td>DROP_CAPTURE Procedure</td>
<td>Removes a privilege analysis policy together with the data recorded</td>
</tr>
<tr>
<td>ENABLE_CAPTURE Procedure</td>
<td>Starts the recording of privilege analysis for a specified privilege analysis policy</td>
</tr>
<tr>
<td>GENERATE_RESULT Procedure</td>
<td>Populates the privilege analysis data dictionary views with data</td>
</tr>
</tbody>
</table>

123.5.1 CAPTURE_DEPENDENCY_PRIVS Procedure

This procedure captures the privileges that are used by definer’s rights and invoker’s rights PL/SQL program units for compilation.

Syntax

```
DBMS_PRIVILEGE_CAPTURE.CAPTURE_DEPENDENCY_PRIVS ();
```

Parameters

This procedure has no parameters.

Usage Notes

Every rerun of the DBMS_PRIVILEGE_CAPTURE.CAPTURE_DEPENDENCY_PRIVS procedure deletes any existing records from the privilege analysis data dictionary views. It then recaptures records based on the existing PL/SQL program units.
123.5.2 CREATE_CAPTURE Procedure

This procedure creates a privilege analysis policy that specifies the conditions for analyzing privilege use. It also optionally specifies the roles for which privilege use is to be analyzed, and the conditions under which privilege use is to be analyzed.

Syntax

```sql
DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE (
  name            IN  VARCHAR2,
  description     IN  VARCHAR2 DEFAULT NULL,
  type            IN  NUMBER DEFAULT G_DATABASE,
  roles           IN  ROLE_NAME_LIST DEFAULT ROLE_NAME_LIST(),
  condition       IN  VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 123-3 CREATE_CAPTURE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the privilege analysis policy. A string of size up to 30 characters.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the policy (up to 1024 characters)</td>
</tr>
<tr>
<td>type</td>
<td>Type of the privilege analysis policy. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- G_DATABASE: Captures all privilege use in the database, except privileges used by the SYS user.</td>
</tr>
<tr>
<td></td>
<td>- G_ROLE: Captures the use of a privilege if the privilege is part of a specified role or list of roles.</td>
</tr>
<tr>
<td></td>
<td>- G_CONTEXT: Captures the use of a privilege if the context specified by the condition parameter evaluates to true.</td>
</tr>
<tr>
<td></td>
<td>- G_ROLE_AND_CONTEXT: Captures the use of a privilege if the privilege is part of the specified list of roles and when the condition specified by the condition parameter is true.</td>
</tr>
<tr>
<td>roles</td>
<td>The roles whose privileges are to be analyzed. Required if the type is G_ROLE or G_ROLE_AND_CONTEXT.</td>
</tr>
<tr>
<td>condition</td>
<td>PL/SQL boolean expression containing up to 4000 characters. Required if type is G_CONTEXT or G_ROLE_AND_CONTEXT. Note that the boolean expression can only contain SYS_CONTEXT, but not other functions.</td>
</tr>
</tbody>
</table>

Usage Notes

- When using role-based analysis for the CREATE_CAPTURE procedure, privilege use is analyzed even if the privilege is indirectly granted to the specified role.

  For example, say role R2 contains role R1, and R1 contains privilege P1. If the privilege policy includes only role R2, any use of the P1 privilege is still analyzed, as privilege P1 is an indirect part of role R2.

- When using the condition parameter, use the following syntax for the PL/SQL expression:

  ```sql
  condition ::= predicate | (predicate1) AND (predicate2)
            | (predicate1) OR (predicate2)
  ```
Where,

\[
predicate ::= \text{sys\_context}(namespace, attribute) \text{ relop constant\_value} | \text{sys\_context}(namespace, attribute) \text{ between constant\_value and constant\_value} | \text{sys\_context}(namespace, attribute) \text{ in\{ constant\_value, (constant\_value)\}^*} \]

Where,

\[
relop ::= = | < | <= | > | >= | <>\]

- A privilege analysis policy cannot analyze the use of **SYS** user privileges.

### 123.5.3 DELETE\_RUN Procedure

This procedure deletes a privilege analysis capture run.

**Syntax**

```
DBMS_PRIVILEGE_CAPTURE.DELETE\_RUN (  
    name IN VARCHAR2,  
    run\_name IN VARCHAR2);
```

**Parameters**

**Table 123-4 ** **DELETE\_RUN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the privilege analysis policy with which the capture run is associated</td>
</tr>
<tr>
<td>run_name</td>
<td>Name of the capture run</td>
</tr>
</tbody>
</table>

**Usage Notes**

- You can find the names of existing privilege capture policies by querying the **DBA\_PRIV\_CAPTURES** data dictionary view.
- Another way to delete a capture run is to drop the policy with which the capture run is associated. Dropping the policy automatically drops its associated capture runs.
- When you drop a capture run it is no longer accessible through the privilege capture data dictionary views.

### 123.5.4 DISABLE\_CAPTURE Procedure

This procedure stops the recording of privilege use for a specified privilege analysis policy. When a policy is disabled, privilege use meeting the policy condition is no longer recorded.

**Syntax**

```
DBMS_PRIVILEGE_CAPTURE.DISABLE\_CAPTURE (  
    name IN VARCHAR2);
```
Parameters

Table 123-5  **DISABLE_CAPTURE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the privilege analysis policy to be disabled</td>
</tr>
</tbody>
</table>

**Usage Notes**

When a privilege analysis policy is first created, it is disabled by default.

### 123.5.5 DROP_CAPTURE Procedure

This procedure removes a privilege analysis policy together with the data recorded. When a policy is removed, all previously recorded privilege use data associated with the policy is deleted.

**Syntax**

```sql
DBMS_PRIVILEGE_CAPTURE.DROP_CAPTURE (name IN VARCHAR2);
```

**Parameters**

Table 123-6  **DROP_CAPTURE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the privilege analysis policy to be removed</td>
</tr>
</tbody>
</table>

**Usage Notes**

- You must disable a privilege analysis policy before removing it. An enabled policy cannot be removed.
- If there are capture runs associated with this policy, then they are automatically dropped when you drop the policy.

### 123.5.6 ENABLE_CAPTURE Procedure

This procedure starts the recording of privilege analysis for a specified privilege analysis policy and optionally provides a capture run for this policy. After a policy is enabled, all privilege use under the policy condition is recorded.

**Syntax**

```sql
DBMS_PRIVILEGE_CAPTURE.ENABLE_CAPTURE (name IN VARCHAR2, run_name IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 123-7  ENABLE_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the privilege analysis policy to be enabled</td>
</tr>
<tr>
<td>run_name</td>
<td>Name of the capture run to associate with this policy, less than 128 characters. Enclose exotic characters in double quotation marks.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply:

- When a privilege analysis policy is first created, it is disabled by default. You must run `ENABLE_CAPTURE` to enable the privilege analysis policy.
- You can enable only one privilege analysis policy at a time. However, a database-wide privilege analysis of the `G_DATABASE` type can be enabled together with another non `G_DATABASE` privilege analysis.
- You cannot enable the same run multiple times. For example, `run_01` cannot be used again if you want to re-enable the capture for `run_01`. Instead, create a new run.

123.5.7 GENERATE_RESULT Procedure

This procedure populates the privilege analysis data dictionary views with data.

See Also:

Oracle Database Vault Administrator’s Guide for more information about privilege analysis views

Syntax

```
DBMS_PRIVILEGE_CAPTURE.GENERATE_RESULT (  
  name        IN VARCHAR2,  
  run_name    IN VARCHAR2 DEFAULT NULL,  
  DEPENDENCY  IN BOOLEAN DEFAULT NULL);
```

Parameters

Table 123-8  GENERATE_RESULT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the privilege analysis policy for which views are populated</td>
</tr>
<tr>
<td>run_name</td>
<td>Name of the capture run that is associated with the privilege analysis policy. If you omit this parameter, then the records of all created runs will be analyzed. When you specify the <code>run_name</code> parameter, only the records of that run are analyzed and all other runs are unaffected.</td>
</tr>
</tbody>
</table>
Table 123-8  (Cont.) GENERATE_RESULT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependency</td>
<td>Enter Y (yes) or N (no) to indicate if PL/SQL compilation privileges, set by the <code>DBMS_PRIVILEGE_CAPTURE.CAPTURE_DEPENDENCY_PRIVS</code> procedure, should be included in the report.</td>
</tr>
</tbody>
</table>

**Usage Notes**

You must disable a privilege analysis policy before populating the privilege analysis views for the policy. You cannot invoke this subprogram on an enabled privilege analysis policy.
The **DBMS_PROCESS** package provides an interface to manage the prespawned servers.

This chapter contains the following topics:

- **DBMS_PROCESS Overview**
- **DBMS_PROCESS Security Model**
- **Summary of DBMS_PROCESS Subprograms**

### 124.1 DBMS_PROCESS Overview

By default, Oracle Database can prespawn foreground processes to improve the performance of client connections.

A prespawned process refers to a process that has been spawned but does not have a session yet. When a user connects to the database or a service process is needed, the process performs further initialization as needed.

To manage foreground processes, use the `DBMS_PROCESS` package. The procedures in this package configure the number of foreground processes for a connection pool, start a connection pool, and stop a connection pool.

**See Also:**

- *Oracle Database Administrator’s Guide* for a detailed description of managing pre-created processes
- *Oracle Database Reference* for information about the `USE_DEDICATED_BROKER` parameter, which determines how dedicated servers are spawned. `TRUE` enables the listener to send the connection to a dedicated connection broker that spawns the dedicated server. `FALSE` sets the listener to spawn a dedicated server directly.

### 124.2 DBMS_PROCESS Security Model

You must be granted the `SYSDBA` administrative privilege to use the `DBMS_PROCESS` package.
124.3 Summary of DBMS_PROCESS Subprograms

This table lists the DBMS_PROCESS subprograms and briefly describes them.

Table 124-1   DBMS_PROCESS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIGURE_POOL Procedure</td>
<td>Configures the minimum number of prespawned processes, the number of prespawned processes in a batch, and the initial number of pre-spawned processes for a foreground connection pool.</td>
</tr>
<tr>
<td>START_POOL Procedure</td>
<td>Starts a prespawned foreground process pool.</td>
</tr>
<tr>
<td>STOP_POOL Procedure</td>
<td>Stops a prespawned foreground process pool.</td>
</tr>
</tbody>
</table>

124.3.1 CONFIGURE_POOL Procedure

This procedure configures the minimum number of prespawned processes, the number of prespawned processes in a batch, and the initial number of pre-spawned processes for a foreground connection pool.

Syntax

```
DBMS_PROCESS.CONFIGURE_POOL(
    pool_name       IN  VARCHAR2 默认 "SYS_DEFAULT_FOREGROUND_POOL",
    min_count       IN  NUMBER 默认 10,
    batch_count     IN  NUMBER 默认 20,
    init_count      IN  NUMBER 默认 0);
```

Parameters

Table 124-2   CONFIGURE_POOL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Name of the foreground pool</td>
</tr>
<tr>
<td>min_count</td>
<td>Count after which a batch of prespawned processes will be spawned in the pool (default 10; maximum number of processes 64,000)</td>
</tr>
<tr>
<td>batch_count</td>
<td>Batch of prespawned servers to be spawned in the pool (default 20). The total value of the min_count and batch_count values cannot exceed the PROCESSES parameter value.</td>
</tr>
<tr>
<td>init_count</td>
<td>Set of prespawned servers to be spawned in startup, or before a storm of connections, spawned in batches (default 0). Do not exceed the PROCESSES parameter value.</td>
</tr>
</tbody>
</table>
Exceptions

Table 124-3  CONFIGURE_POOL Procedure Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-456</td>
<td>Prespawn not enabled</td>
</tr>
<tr>
<td>ORA-457</td>
<td>Prespawn pool not found</td>
</tr>
<tr>
<td>ORA-458</td>
<td>Prespawn pool already started</td>
</tr>
<tr>
<td>ORA-459</td>
<td>Prespawn pool not started</td>
</tr>
<tr>
<td>ORA-460</td>
<td>Process pool invalid configuration value(s)</td>
</tr>
</tbody>
</table>

Usage Notes

- To find information about foreground processes, query the `V$PROCESS_POOL` dynamic view.

Example

BEGIN
    DBMS_PROCESS.CONFIGURE_POOL (  
        pool_name => 'hrdb_pool',  
        min_count => 40,  
        batch_count => 20,  
        init_count  => 10);  
END;  /

124.3.2 START_POOL Procedure

This procedure starts a prespawned foreground process pool.

Syntax

```
DBMS_PROCESS.START_POOL(  
    pool_name      IN VARCHAR2 DEFAULT "SYS_DEFAULT_FOREGROUND_POOL");
```

Parameters

Table 124-4  START_POOL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Name of prespawned foreground process pool. To find existing pools, query the <code>V$PROCESS_POOL</code> dynamic view.</td>
</tr>
</tbody>
</table>
Exceptions

Table 124-5  START_POOL Procedure Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-457</td>
<td>Prespawn pool not found</td>
</tr>
<tr>
<td>ORA-458</td>
<td>Prespawn pool already started</td>
</tr>
<tr>
<td>ORA-459</td>
<td>Prespawn pool not started</td>
</tr>
</tbody>
</table>

Example

BEGIN
  DBMS_PROCESS.START_POOL ( 
    pool_name   => 'hrdb_pool',
  ) END;

124.3.3 STOP_POOL Procedure

This procedure stops a prespawned foreground process pool.

Syntax

DBMS_PROCESS.STOP_POOL( 
  pool_name IN VARCHAR2 DEFAULT "SYS_DEFAULT_FOREGROUND_POOL"); 

Parameters

Table 124-6  STOP_POOL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_name</td>
<td>Name of the prespawned foreground process pool. To find existing pools, query the V$PROCESS_POOL dynamic view.</td>
</tr>
</tbody>
</table>

Exceptions

Table 124-7  STOP_POOL Procedure Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-457</td>
<td>Prespawn pool not found</td>
</tr>
<tr>
<td>ORA-458</td>
<td>Prespawn pool already started</td>
</tr>
<tr>
<td>ORA-459</td>
<td>Prespawn pool not started</td>
</tr>
</tbody>
</table>

Example

BEGIN
  DBMS_PROCESS.STOP_POOL ( 
    pool_name   => 'hrdb_pool',
  ) END;
125

DBMS_PROFILER

The package provides an interface to profile existing PL/SQL applications and identify performance bottlenecks. You can then collect and persistently store the PL/SQL profiler data.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Exceptions
- Summary of DBMS_PROFILER Subprograms

125.1 DBMS_PROFILER Overview

This package enables the collection of profiler (performance) data for performance improvement or for determining code coverage for PL/SQL applications. Application developers can use code coverage data to focus their incremental testing efforts.

With this interface, you can generate profiling information for all named library units that are executed in a session. The profiler gathers information at the PL/SQL virtual machine level. This information includes the total number of times each line has been executed, the total amount of time that has been spent executing that line, and the minimum and maximum times that have been spent on a particular execution of that line.

**Note:**

It is possible to infer the code coverage figures for PL/SQL units for which data has been collected.

The profiling information is stored in database tables. This enables querying on the data: you can build customizable reports (summary reports, hottest lines, code coverage data, and so on. And you can analyze the data.

The PROFTAB.SQL script creates tables with the columns, datatypes, and definitions as shown in the following tables.

### Table 125-1  Columns in Table PSQL_PROFILER_RUNS

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>runid</td>
<td>NUMBER PRIMARY KEY</td>
<td>Unique run identifier from psql_profile_runnumber</td>
</tr>
<tr>
<td>related_run</td>
<td>NUMBER</td>
<td>Runid of related run (for client/server correlation)</td>
</tr>
</tbody>
</table>
### Table 125-1 (Cont.) Columns in Table PLSQL_PROFILER_RUNS

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_owner</td>
<td>VARCHAR2(128)</td>
<td>User who started run</td>
</tr>
<tr>
<td>run_date</td>
<td>DATE</td>
<td>Start time of run</td>
</tr>
<tr>
<td>run_comment</td>
<td>VARCHAR2(2047)</td>
<td>User provided comment for this run</td>
</tr>
<tr>
<td>run_total_time</td>
<td>NUMBER</td>
<td>Elapsed time for this run in nanoseconds</td>
</tr>
<tr>
<td>run_system_info</td>
<td>VARCHAR2(2047)</td>
<td>Currently unused</td>
</tr>
<tr>
<td>run_comment1</td>
<td>VARCHAR2(2047)</td>
<td>Additional comment</td>
</tr>
<tr>
<td>spare1</td>
<td>VARCHAR2(256)</td>
<td>Unused</td>
</tr>
</tbody>
</table>

### Table 125-2 Columns in Table PLSQL_PROFILER_UNITS

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>runid</td>
<td>NUMBER</td>
<td>Primary key, references plsql_profiler_runs,</td>
</tr>
<tr>
<td>unit_number</td>
<td>NUMBER</td>
<td>Primary key, internally generated library unit #</td>
</tr>
<tr>
<td>unit_type</td>
<td>VARCHAR2(128)</td>
<td>Library unit type</td>
</tr>
<tr>
<td>unit_owner</td>
<td>VARCHAR2(128)</td>
<td>Library unit owner name</td>
</tr>
<tr>
<td>unit_name</td>
<td>VARCHAR2(128)</td>
<td>Library unit name timestamp on library unit</td>
</tr>
<tr>
<td>unit_timestamp</td>
<td>DATE</td>
<td>In the future will be used to detect changes to unit between runs</td>
</tr>
<tr>
<td>total_time</td>
<td>NUMBER</td>
<td>Total time spent in this unit in nanoseconds. The profiler does not set this field, but it is provided for the convenience of analysis tools.</td>
</tr>
<tr>
<td>spare1</td>
<td>NUMBER</td>
<td>Unused</td>
</tr>
<tr>
<td>spare2</td>
<td>NUMBER</td>
<td>Unused</td>
</tr>
</tbody>
</table>

### Table 125-3 Columns in Table PLSQL_PROFILER_DATA

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>runid</td>
<td>NUMBER</td>
<td>Primary key, unique (generated) run identifier</td>
</tr>
<tr>
<td>unit_number</td>
<td>NUMBER</td>
<td>Primary key, internally generated library unit number</td>
</tr>
<tr>
<td>line#</td>
<td>NUMBER</td>
<td>Primary key, not null, line number in unit</td>
</tr>
<tr>
<td>total_occur</td>
<td>NUMBER</td>
<td>Number of times line was executed</td>
</tr>
<tr>
<td>total_time</td>
<td>NUMBER</td>
<td>Total time spent executing line in nanoseconds</td>
</tr>
<tr>
<td>min_time</td>
<td>NUMBER</td>
<td>Minimum execution time for this line in nanoseconds</td>
</tr>
<tr>
<td>max_time</td>
<td>NUMBER</td>
<td>Maximum execution time for this line in nanoseconds</td>
</tr>
<tr>
<td>spare1</td>
<td>NUMBER</td>
<td>Unused</td>
</tr>
<tr>
<td>spare2</td>
<td>NUMBER</td>
<td>Unused</td>
</tr>
<tr>
<td>spare3</td>
<td>NUMBER</td>
<td>Unused</td>
</tr>
<tr>
<td>spare4</td>
<td>NUMBER</td>
<td>Unused</td>
</tr>
</tbody>
</table>
With Oracle database version 8.x, a sample textual report writer (profrep.sql) is provided with the PL/SQL demo scripts.

Note that prior to Oracle Database 10g, the DBMS_PROFILER package was not automatically loaded when the database was created, and the Oracle-supplied PROFLOAD.SQL script was used to create it. In 10g and beyond, the DBMS_PROFILER package is loaded automatically when the database is created, and PROFLOAD.SQL is no longer needed.

### 125.2 DBMS_PROFILER Security Model

The profiler only gathers data for units for which a user has CREATE privilege; you cannot use the package to profile units for which EXECUTE ONLY access has been granted. In general, if a user can debug a unit, the same user can profile it. However, a unit can be profiled whether or not it has been compiled DEBUG. Oracle advises that modules that are being profiled should be compiled DEBUG, since this provides additional information about the unit in the database.

**Note:**

DBMS_PROFILER treats any program unit that is compiled in NATIVE mode as if you do not have CREATE privilege, that is, you will not get any output.

### 125.3 DBMS_PROFILER Operational Notes

These notes describe a typical run, how to interpret output, and two methods of exception generation.

**Typical Run**

Improving application performance is an iterative process. Each iteration involves the following steps:

1. Running the application with one or more benchmark tests with profiler data collection enabled.
2. Analyzing the profiler data and identifying performance problems.
3. Fixing the problems.

The PL/SQL profiler supports this process using the concept of a "run". A run involves running the application through benchmark tests with profiler data collection enabled. You can control the beginning and the ending of a run by calling the START_PROFILER and STOP_PROFILER functions.

The user must first create database tables in the profiler user’s schema to collect the data. The PROFTAB.SQL script creates the tables and other data structures required for persistently storing the profiler data.

Note that running PROFTAB.SQL drops the current tables. The PROFTAB.SQL script is in the RDBMS/ADMIN directory. Some PL/SQL operations, such as the first execution of a PL/SQL unit, may involve I/O to catalog tables to load the byte code for the PL/SQL unit being executed. Also, it may take some time executing package initialization code the first time a package procedure or function is called.
To avoid timing this overhead, "warm up" the database before collecting profile data. To do this, run the application once without gathering profiler data.

You can allow profiling across all users of a system, for example, to profile all users of a package, independent of who is using it. In such cases, the SYSADMIN should use a modified PROFTAB.SQL script which:

- Creates the profiler tables and sequence
- Grants SELECT/INSERT/UPDATE on those tables and sequence to all users
- Defines public synonyms for the tables and sequence

**Note:**

Do not alter the actual fields of the tables.

A typical run then involves:

- Starting profiler data collection in the run.
- Executing PL/SQL code for which profiler and code coverage data is required.
- Stopping profiler data collection, which writes the collected data for the run into database tables

**Note:**

The collected profiler data is not automatically stored when the user disconnects. You must issue an explicit call to the FLUSH_DATA or the STOP_PROFILER function to store the data at the end of the session. Stopping data collection stores the collected data.

As the application executes, profiler data is collected in memory data structures that last for the duration of the run. You can call the FLUSH_DATA function at intermediate points during the run to get incremental data and to free memory for allocated profiler data structures. Flushing the collected data involves storing collected data in the database tables created earlier.

**See Also:**

"FLUSH_DATA Function and Procedure".

**Interpreting Output**

The table plsql_profiler_data contains one row for each line of the source unit for which code was generated. The line# value specifies which source line. If the row exists, and the total_occur value in that row is > 0, some code associated with that line was executed. If the row exists, and total_occur value is 0, no code associated with that line was executed. If the row doesn't exist in the table, no code was generated for that line, and therefore it should not be mentioned in reports.
If the source of a single statement is on a single line, any code generated for that statement will be attributed to that line number. (In some cases, such as a simple declaration, or because of optimization, no code will be needed). To get coverage information, units should be compiled with `PLSQL_OPTIMIZE_LEVEL=1`.

If a statement spans multiple lines, any code generated for that statement will be attributed to some line in the range, but it is not guaranteed that every line in the range will have code attributed to it. In such a case there will be gaps in the set of line# values. In particular, multi-line SQL-related statements may appear to be on a single line (usually the first). This is because PL/SQL passes the processed text of the cursor to the SQL engine; therefore, as far as PL/SQL is concerned, the entire SQL statement is a single indivisible operation.

When multiple statements are on the same line, the profiler will combine the occurrences for each statement. This may be confusing if a line has embedded control flow. For example, if 'then ...' and 'else ...' are on the same line, it will not be possible to determine whether the 'then' or the 'else' was taken.

In general, profiler and coverage reports are most easily interpreted if each statement is on its own line.

**Two Methods of Exception Generation**

Each routine in this package has two versions that allow you to determine how errors are reported.

- A function that returns success/failure as a status value and will never raise an exception
- A procedure that returns normally if it succeeds and raises an exception if it fails

In each case, the parameters of the function and procedure are identical. Only the method by which errors are reported differs. If there is an error, there is a correspondence between the error codes that the functions return, and the exceptions that the procedures raise.

To avoid redundancy, the following section only provides details about the functional form.

### 125.4 DBMS_PROFILER Exceptions

DBMS_PROFILER throws the exceptions described in this topic.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version_mismatch</td>
<td>Corresponds to <code>error_version</code>.</td>
</tr>
<tr>
<td>profiler_error</td>
<td>Corresponds to either &quot;error_param&quot; or &quot;error_io&quot;.</td>
</tr>
</tbody>
</table>

A 0 return value from any function denotes successful completion; a nonzero return value denotes an error condition. The possible errors are as follows:

- 'A subprogram was called with an incorrect parameter.'
  ```sql
  error_param constant binary_integer := 1;
  ```

- 'Data flush operation failed. Check whether the profiler tables have been created, are accessible, and that there is adequate space.'
There is a mismatch between package and database implementation. Oracle returns this error if an incorrect version of the DBMS_PROFILER package is installed, and if the version of the profiler package cannot work with this database version. The only recovery is to install the correct version of the package.

125.5 Summary of DBMS_PROFILER Subprograms

This table lists the DBMS_PROFILER subprograms and briefly describes them.

Table 125-5   DBMS_PROFILER Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUSH_DATA Function and Procedure</td>
<td>Flushes profiler data collected in the user's session</td>
</tr>
<tr>
<td>GET_VERSION Procedure</td>
<td>Gets the version of this API</td>
</tr>
<tr>
<td>INTERNAL_VERSION_CHECK Function</td>
<td>Verifies that this version of the DBMS_PROFILER package can work with the implementation in the database</td>
</tr>
<tr>
<td>PAUSE_PROFILER Function and Procedure</td>
<td>Pauses profiler data collection</td>
</tr>
<tr>
<td>RESUME_PROFILER Function and Procedure</td>
<td>Resumes profiler data collection</td>
</tr>
<tr>
<td>START_PROFILER Functions and Procedures</td>
<td>Starts profiler data collection in the user's session</td>
</tr>
<tr>
<td>STOP_PROFILER Function and Procedure</td>
<td>Stops profiler data collection in the user's session</td>
</tr>
</tbody>
</table>

125.5.1 FLUSH_DATA Function and Procedure

This function flushes profiler data collected in the user's session. The data is flushed to database tables, which are expected to preexist.

>Note:

Use the PROFTAB.SQL script to create the tables and other data structures required for persistently storing the profiler data.

Syntax

```sql
DBMS_PROFILER.FLUSH_DATA
RETURN BINARY_INTEGER;

DBMS_PROFILER.FLUSH_DATA;
```
125.5.2 GET_VERSION Procedure

This procedure gets the version of this API.

Syntax

```sql
DBMS_PROFILER.GET_VERSION (
    major  OUT BINARY_INTEGER,
    minor  OUT BINARY_INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>Major version of DBMS_PROFILER.</td>
</tr>
<tr>
<td>minor</td>
<td>Minor version of DBMS_PROFILER.</td>
</tr>
</tbody>
</table>

125.5.3 INTERNAL_VERSION_CHECK Function

This function verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.

Syntax

```sql
DBMS_PROFILER.INTERNAL_VERSION_CHECK
    RETURN BINARY_INTEGER;
```

125.5.4 PAUSE_PROFILER Function and Procedure

This function pauses profiler data collection.

Syntax

```sql
DBMS_PROFILER.PAUSE_PROFILER
    RETURN BINARY_INTEGER;
```

```java
DBMS_PROFILER.PAUSE_PROFILER;
```

125.5.5 RESUME_PROFILER Function and Procedure

This function resumes profiler data collection.

Syntax

```sql
DBMS_PROFILER.RESUME_PROFILER
    RETURN BINARY_INTEGER;
```

```java
DBMS_PROFILER.RESUME_PROFILER;
```
125.5.6 START_PROFILER Functions and Procedures

This function starts profiler data collection in the user's session.

There are two overloaded forms of the START_PROFILER function; one returns the run number of the started run, as well as the result of the call. The other does not return the run number. The first form is intended for use with GUI-based tools controlling the profiler.

Syntax

```sql
DBMS(PROFILER.START_PROFILER(
    run_comment   IN VARCHAR2 := sysdate,
    run_comment1  IN VARCHAR2 :=''
    run_number    OUT BINARY_INTEGER)
RETURN BINARY_INTEGER;

DBMS_PROFILER.START_PROFILER(
    run_comment IN VARCHAR2 := sysdate,
    run_comment1 IN VARCHAR2 :='')
RETURN BINARY_INTEGER;

DBMS_PROFILER.START_PROFILER(
    run_comment IN VARCHAR2 := sysdate,
    run_comment1 IN VARCHAR2 :=''
    run_number OUT BINARY_INTEGER);

DBMS_PROFILER.START_PROFILER(
    run_comment IN VARCHAR2 := sysdate,
    run_comment1 IN VARCHAR2 :='');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_comment</td>
<td>Each profiler run can be associated with a comment. For example, the comment could provide the name and version of the benchmark test that was used to collect data.</td>
</tr>
<tr>
<td>run_number</td>
<td>Stores the number of the run so you can store and later recall the run's data.</td>
</tr>
<tr>
<td>run_comment1</td>
<td>Allows you to make interesting comments about the run.</td>
</tr>
</tbody>
</table>

125.5.7 STOP_PROFILER Function and Procedure

This function stops profiler data collection in the user's session.

This function has the side effect of flushing data collected so far in the session, and it signals the end of a run.

Syntax

```sql
DBMS_PROFILER.STOP_PROFILER
RETURN BINARY_INTEGER;
```
DBMS_PROFILER.STOP_PROFILER;
The `DBMS_PROPAGATION_ADM` package, one of a set of Oracle Streams packages, provides administrative interfaces for configuring a propagation from a source queue to a destination queue.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_PROPAGATION_ADM Subprograms

### 126.1 DBMS_PROPAGATION_ADM Overview

This package provides interfaces to start, stop, and configure a propagation.

See Also:

*Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and propagations

### 126.2 DBMS_PROPAGATION_ADM Security Model

Security on this package can be controlled by granting `EXECUTE` on this package to selected users or roles, or by granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

When the `DBMS_PROPAGATION_ADM` package is used to manage an Oracle Streams configuration, it requires that the user is granted the privileges of an Oracle Streams administrator.

See Also:

*Oracle Streams Concepts and Administration* for information about configuring an Oracle Streams administrator

### 126.3 Summary of DBMS_PROPAGATION_ADM Subprograms

This table lists the `DBMS_PROPAGATION_ADM` subprograms and briefly describes them.
### Table 126-1  DBMS_PROPAGATION_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_PROPAGATION Procedure</td>
<td>Adds, alters, or removes a rule set for a propagation</td>
</tr>
<tr>
<td>CREATE_PROPAGATION Procedure</td>
<td>Creates a propagation and specifies the source queue, destination queue, and&lt;br&gt;rule set for the propagation</td>
</tr>
<tr>
<td>DROP_PROPAGATION Procedure</td>
<td>Drops a propagation</td>
</tr>
<tr>
<td>START_PROPAGATION Procedure</td>
<td>Starts a propagation</td>
</tr>
<tr>
<td>STOP_PROPAGATION Procedure</td>
<td>Stops a propagation</td>
</tr>
</tbody>
</table>

Note: All subprograms commit unless specified otherwise.

#### 126.3.1 ALTER_PROPAGATION Procedure

This procedure adds, alters, or removes a rule set for a propagation.

See Also:

Oracle Streams Concepts and Administration and DBMS_RULE_ADM for more information about rules and rule sets

Syntax

```sql
DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
    propagation_name          IN  VARCHAR2,
    rule_set_name             IN  VARCHAR2  DEFAULT NULL,
    remove_rule_set           IN  BOOLEAN   DEFAULT FALSE,
    negative_rule_set_name    IN  VARCHAR2  DEFAULT NULL,
    remove_negative_rule_set  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

#### Table 126-2  ALTER_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagation you are altering. You must specify an existing&lt;br&gt;propagation name. Do not specify an owner.</td>
</tr>
</tbody>
</table>
Table 126-2  (Cont.) ALTER_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_set_name</td>
<td>The name of the positive rule set for the propagation. The positive rule set contains the rules that instruct the propagation to propagate messages.</td>
</tr>
<tr>
<td></td>
<td>If you want to use a positive rule set for the propagation, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the hr schema named prop_rules, enter hr.prop_rules. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</td>
</tr>
<tr>
<td></td>
<td>If you specify NULL and the remove_rule_set parameter is set to FALSE, then the procedure retains any existing positive rule set. If you specify NULL and the remove_rule_set parameter is set to TRUE, then the procedure removes any existing positive rule set.</td>
</tr>
<tr>
<td>remove_rule_set</td>
<td>If TRUE, then the procedure removes the positive rule set for the specified propagation. If you remove a positive rule set for a propagation, and the propagation does not have a negative rule set, then the propagation propagates all messages.</td>
</tr>
<tr>
<td></td>
<td>If you remove a positive rule set for a propagation, and a negative rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the negative rule set.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure retains the positive rule set for the specified propagation. If the rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
</tbody>
</table>
Table 126-2  (Cont.) ALTER_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>negative_rule_set_name</td>
<td>The name of the negative rule set for the propagation. The negative rule set contains the rules that instruct the propagation to discard messages. If you want to use a negative rule set for the propagation, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a negative rule set in the hr schema named neg_rules, enter hr.neg_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL and the remove_negative_rule_set parameter is set to FALSE, then the procedure retains any existing negative rule set. If you specify NULL and the remove_negative_rule_set parameter is set to TRUE, then the procedure removes any existing negative rule set. If you specify both a positive and a negative rule set for a propagation, then the negative rule set is always evaluated first.</td>
</tr>
<tr>
<td>remove_negative_rule_set</td>
<td>If TRUE, then the procedure removes the negative rule set for the specified propagation. If you remove a negative rule set for a propagation, and the propagation does not have a positive rule set, then the propagation propagates all messages. If you remove a negative rule set for a propagation, and a positive rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the positive rule set. If FALSE, then the procedure retains the negative rule set for the specified propagation. If the negative_rule_set_name parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
</tbody>
</table>

126.3.2 CREATE_PROPAGATION Procedure

This procedure creates a propagation and specifies the source queue, destination queue, and any rule set for the propagation. A propagation propagates messages in a local source queue to a destination queue. The destination queue might or might not be in the same database as the source queue.

Syntax

```sql
DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name           IN  VARCHAR2,
    source_queue               IN  VARCHAR2,
    destination_queue          IN  VARCHAR2,
    destination_dblink         IN  VARCHAR2  DEFAULT NULL,
    rule_set_name              IN  VARCHAR2  DEFAULT NULL,
    negative_rule_set_name     IN  VARCHAR2  DEFAULT NULL,
);```

Chapter 126
Summary of DBMS_PROPAGATION_ADM Subprograms
Parameters

Table 126-3  CREATE_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagation you are creating. A NULL setting is not allowed. Do not specify an owner.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The propagation_name setting cannot be altered after the propagation is created.</td>
</tr>
<tr>
<td>source_queue</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue</td>
<td>The name of the destination queue, specified as [schema_name.]queue_name.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a destination queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_dblink</td>
<td>The name of the database link that will be used by the propagation. The database link is from the database that contains the source queue to the database that contains the destination queue.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the source queue and destination queue must be in the same database.</td>
</tr>
<tr>
<td>rule_set_name</td>
<td>The name of the positive rule set for the propagation. The positive rule set contains the rules that instruct the propagation to propagate messages.</td>
</tr>
<tr>
<td></td>
<td>If you want to use a positive rule set for the propagation, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a positive rule set in the hr schema named prop_rules, enter hr.prop_rules. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</td>
</tr>
<tr>
<td></td>
<td>If you specify NULL, and no negative rule set exists for the propagation, then the propagation propagates all messages in its queue.</td>
</tr>
<tr>
<td></td>
<td>If you specify NULL, and a negative rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the negative rule set.</td>
</tr>
</tbody>
</table>
Table 126-3   (Cont.) CREATE_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>negative_rule_set_name</td>
<td>The name of the negative rule set for the propagation. The negative rule set contains the rules that instruct the propagation to discard messages. If you want to use a negative rule set for the propagation, then you must specify an existing rule set in the form [schema_name.]rule_set_name. For example, to specify a negative rule set in the hr schema named neg_rules, enter hr.neg_rules. If the schema is not specified, then the current user is the default. An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package. If you specify NULL, and no positive rule set exists for the propagation, then the propagation propagates all messages in its queue. If you specify NULL, and a positive rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the positive rule set. If you specify both a positive and a negative rule set for a propagation, then the negative rule set is always evaluated first.</td>
</tr>
<tr>
<td>queue_to_queue</td>
<td>If TRUE or NULL, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database. If FALSE, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment. <strong>See Also:</strong> Oracle Streams Concepts and Administration for more information about queue-to-queue propagations</td>
</tr>
<tr>
<td>original_propagation_name</td>
<td>Specify the original propagation name if the propagation being created is part of a split and merge operation initiated by the SPLIT_STREAMS procedure in the DBMS_STREAMS_ADM package. The split operation clones the original propagation under a new name. The name of the original propagation is important when the cloned propagation is copied back to the original stream using the MERGE_STREAMS procedure in the DBMS_STREAMS_ADM package. Specify NULL if the propagation being created is not part of a split and merge operation. <strong>See Also:</strong> SPLIT_STREAMS Procedure and MERGE_STREAMS Procedure</td>
</tr>
</tbody>
</table>
Table 126-3  (Cont.) CREATE_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_merge_theshold</td>
<td>Specify a positive number if both of the following conditions are met:</td>
</tr>
<tr>
<td></td>
<td>• The propagation being created is part of a split and merge operation initiated by the SPLIT_STREAMS procedure in the DBMS_STREAMS_ADM package.</td>
</tr>
<tr>
<td></td>
<td>• The stream will be merged back to the original stream automatically.</td>
</tr>
<tr>
<td></td>
<td>Specify NULL if either of the following conditions are met:</td>
</tr>
<tr>
<td></td>
<td>• The propagation being created is not part of a split and merge operation.</td>
</tr>
<tr>
<td></td>
<td>• The propagation being created is part of a split and merge operation, but the stream being split off will not be merged back to the original stream automatically.</td>
</tr>
</tbody>
</table>

See Also: SPLIT_STREAMS Procedure and MERGE_STREAMS Procedure

Usage Notes

This procedure starts propagation and might create a propagation job. If this procedure creates a propagation job, then it establishes a default schedule for the propagation job. Each propagation job is an Oracle Scheduler job. You can adjust the schedule of a propagation job using Oracle Scheduler.

The user who owns the source queue is the user who propagates messages. This user must have the necessary privileges to propagate messages.

See Also:
- DBMS_RULE_ADM
- Oracle Streams Concepts and Administration for more information about propagations, the privileges required to propagate messages, propagation jobs, and propagation schedules

126.3.3 DROP_PROPAGATION Procedure

This procedure drops a propagation and deletes all messages for the destination queue in the source queue. This procedure also removes the schedule for propagation from the source queue to the destination queue.

Syntax

```sql
DBMS_PROPAGATION_ADM.DROP_PROPAGATION(
    propagation_name IN VARCHAR2,
    drop_unused_rule_sets IN BOOLEAN DEFAULT FALSE);
```
Parameters

Table 126-4  DROP_PROPAGATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagation you are dropping. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
<tr>
<td>drop_unused_rule_sets</td>
<td>If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified propagation if these rule sets are not used by any other Oracle Streams client, which includes capture processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set. If FALSE, then the procedure does not drop the rule sets used by the specified propagation, and the rule sets retain their rules.</td>
</tr>
</tbody>
</table>

Usage Notes

When you use this procedure to drop a propagation, information about rules created for the propagation using the DBMS_STREAMS_ADM package is removed from the data dictionary views for Oracle Streams rules. Information about such a rule is removed even if the rule is not in either rule set for the propagation.

See Also:

Oracle Streams Concepts and Administration for more information about Oracle Streams data dictionary views

The following are the data dictionary views for Oracle Streams rules:

- ALL_STREAMS_GLOBAL_RULES
- DBA_STREAMS_GLOBAL_RULES
- ALL_STREAMS_MESSAGE_RULES
- DBA_STREAMS_MESSAGE_RULES
- ALL_STREAMS_SCHEMA_RULES
- DBA_STREAMS_SCHEMA_RULES
- ALL_STREAMS_TABLE_RULES
- DBA_STREAMS_TABLE_RULES

Note:

When you drop a propagation, the propagation job used by the propagation is dropped automatically, if no other propagations are using the propagation job.
126.3.4 START_PROPAGATION Procedure

This procedure starts a propagation.

Syntax

DBMS_PROPAGATION_ADM.START_PROPAGATION(
    propagation_name  IN  VARCHAR2);

Parameter

Table 126-5  START_PROPAGATION Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagation you are starting. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
</tbody>
</table>

Usage Notes

The propagation status is persistently recorded. Hence, if the status is ENABLED, then the propagation is started upon database instance startup.

126.3.5 STOP_PROPAGATION Procedure

This procedure stops a propagation.

Syntax

DBMS_PROPAGATION_ADM.STOP_PROPAGATION(
    propagation_name  IN  VARCHAR2,
    force             IN  BOOLEAN  DEFAULT FALSE);

Parameter

Table 126-6  STOP_PROPAGATION Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagation you are stopping. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, then the procedure stops the propagation and clears the statistics for the propagation. If FALSE, then the procedure stops the propagation without clearing the statistics for the propagation.</td>
</tr>
</tbody>
</table>

Usage Notes

The propagation status is persistently recorded. Hence, if the status is DISABLED or ABORTED, then the propagation is not started upon database instance startup.
The DBMS_QOPATCH package provides an interface to view the installed database patches.

This chapter contains the following topics:

• Overview
• Security Model
• Operational Notes
• Error Messages
• Summary of DBMS_QOPATCH Subprograms

127.1 DBMS_QOPATCH Overview

The DBMS_QOPATCH package provides a PLSQL/SQL interface to view the installed database patches. The interface provides all the patch information available as part of the OPATCH LSINVENTORY -XML command. The package accesses the OUI patch inventory in real time to provide patch and meta-information.

127.2 DBMS_QOPATCH Security Model

The DBMS_QOPATCH package is created as part of SYS schema and SYS is the only user who can execute these subprograms.

127.3 DBMS_QOPATCH Operational Notes

The following operational notes apply to DBMS_QOPATCH.

• The package will work only if the database is OPEN.
• In an Oracle Real Application Clusters (RAC) environment, if the subprogram requires to fetch data from other RAC nodes it spawns a job in the other node(s) to get the data. In this case JOB_QUEUE_PROCESSES needs to be >0 for the package to fetch the data from other RAC nodes.
• If there is a delay in the job execution, the package returns ORA–20008 error.

127.4 DBMS_QOPATCH Exceptions

This table lists the exceptions raised by the DBMS_QOPATCH package.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA–20001</td>
<td>Latest XML inventory is not loaded into table.</td>
</tr>
</tbody>
</table>
Table 127-1  (Cont.) DBMS_QOPATCH Error Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20002</td>
<td>Directory creation failed.</td>
</tr>
<tr>
<td>ORA-20003</td>
<td>Configuration of a job on a node failed.</td>
</tr>
<tr>
<td>ORA-20004</td>
<td>Job configuration failed as node is inactive.</td>
</tr>
<tr>
<td>ORA-20005</td>
<td>Job is not configured with given node, instance name.</td>
</tr>
<tr>
<td>ORA-20006</td>
<td>Number of RAC active instances and opatch jobs configured are not same.</td>
</tr>
<tr>
<td>ORA-20007</td>
<td>Job configuration failed as node or instance is not active.</td>
</tr>
<tr>
<td>ORA-20008</td>
<td>Timed out - job execution time is more than 120Secs.</td>
</tr>
<tr>
<td>ORA-20009</td>
<td>Job execution failed.</td>
</tr>
<tr>
<td>ORA-20010</td>
<td>Node is inactive and job cannot be executed.</td>
</tr>
<tr>
<td>ORA-20011</td>
<td>Job name is NULL and inventory cannot be loaded.</td>
</tr>
<tr>
<td>ORA-20012</td>
<td>JOB_QUEUE_PROCESSES is set to zero and the inventory cannot be loaded.</td>
</tr>
<tr>
<td>ORA-20013</td>
<td>DBMS_QOPATCH ran mostly in non-install area.</td>
</tr>
<tr>
<td>ORA-20014</td>
<td>Database is not opened.</td>
</tr>
<tr>
<td>ORA-20015</td>
<td>Database opened as read-only.</td>
</tr>
</tbody>
</table>

127.5 Summary of DBMS_QOPATCH Subprograms

This table lists and briefly describes the DBMS_QOPATCH package subprograms.

Table 127-2  DBMS_QOPATCH Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK_PATCH_INSTALLED Function</td>
<td>Checks if a patch is installed.</td>
</tr>
<tr>
<td>GET_OPATCH_BUGS Function</td>
<td>Provides a bugs list for a patch in XML format if the patch number is given.</td>
</tr>
<tr>
<td></td>
<td>If patch is not given then it lists all the bugs installed in all the patches in XML format.</td>
</tr>
<tr>
<td>GET_OPATCH_COUNT Function</td>
<td>Provides the total number of installed patches in XML format.</td>
</tr>
<tr>
<td>GET_OPATCH_DATA Function</td>
<td>Provides top level patch information for the patch (such as Patch ID, patch creation time) in the XML element</td>
</tr>
<tr>
<td>GET_OPATCH_FILES Function</td>
<td>Provides the list of files modified in the given patch number in XML format</td>
</tr>
<tr>
<td>GET_OPATCH_INSTALL_INFO Function</td>
<td>Returns the XML element containing the ORACLE_HOME details such as patch and inventory location</td>
</tr>
<tr>
<td>GET_OPATCH_LIST Function</td>
<td>Provides list of patches installed as an XML element from the XML inventory</td>
</tr>
<tr>
<td>GET_OPATCH_LSiNVENTORY</td>
<td>Returns whole opatch inventory as XML instance document.</td>
</tr>
<tr>
<td>GET_OPATCH_OLAYS Function</td>
<td>Provides overlay patches for a given patch as XML element</td>
</tr>
</tbody>
</table>
### Table 127-2 (Cont.) DBMS_QOPATCH Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_OPATCH_PREQS Function</td>
<td>Provides prerequisite patches for a given patch as XML element</td>
</tr>
<tr>
<td>GET_OPATCH_XSLT</td>
<td>Returns the style-sheet for the opatch XML inventory presentation</td>
</tr>
<tr>
<td>GET_PATCH_DETAILS Function</td>
<td>Displays the detailed patch information from <code>opatch lsinventory</code>.</td>
</tr>
<tr>
<td>GET_SQLPATCH_STATUS Procedure</td>
<td>Displays the SQL patch status by querying from SQL patch registry to produce complete patch level information</td>
</tr>
<tr>
<td>IS_PATCH_INSTALLED Function</td>
<td>Provides information (such as patchID, application date, and SQL patch information) on the installed patch as XML node by querying the XML inventory</td>
</tr>
<tr>
<td>OPATCH_COMPARE_CURRENT Function</td>
<td>Compares the current database with a list of bugs.</td>
</tr>
<tr>
<td>OPATCH_COMPARE_NODES Function</td>
<td>Compares the given RAC node(s) with the instance to the present connected node with current node. The return value indicates whether the data was refreshed or not.</td>
</tr>
<tr>
<td>SET_CURRENT_OPINST Procedure</td>
<td>Sets the node name and instance to get the inventory details specific to it in an Oracle Real Application Clusters (RAC) environment</td>
</tr>
</tbody>
</table>

#### 127.5.1 CHECK_PATCH_INSTALLED Function

Checks if the patch is installed.

**Syntax**

```sql
DBMS_QOPATCH.CHECK_PATCH_INSTALLED (    bugs     IN QOPATCH_LIST);    RETURN VARCHAR2;
```

**Parameters**

**Table 127-3 CHECK_PATCH_INSTALLED Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bugs</td>
<td>List of patches to be checked.</td>
</tr>
</tbody>
</table>

#### 127.5.2 GET_OPATCH_BUGS Function

This function provides a bugs list in a patch if the patch number is given. If a patch number is not given, it lists all the bugs in the specified XML format.

**Syntax**

```sql
DBMS_QOPATCH.GET_OPATCH_BUGS (    patchnum IN VARCHAR2 DEFAULT NULL);    RETURN XMLTYPE;
```
Parameters

Table 127-4  GET_OPATCH_BUGS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>

127.5.3 GET_OPATCH_COUNT Function

This function provides the total number of installed patches in XML format.

Syntax

```sql
DBMS_QOPATCH.GET_OPATCH_COUNT (
   patchnum IN VARCHAR2);
RETURN XMLTYPE;
```

Parameters

Table 127-5  GET_OPATCH_COUNT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>

127.5.4 GET_OPATCH_DATA Function

This function provides top level patch information for the patch (such as Patch ID, patch creation time) in the XML element.

Syntax

```sql
DBMS_QOPATCH.GET_OPATCH_DATA (
   patchnum IN VARCHAR2);
RETURN XMLTYPE;
```

Parameters

Table 127-6  GET_OPATCH_DATA Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>
127.5.5 GET_OPATCH_FILES Function

This function provides the list of files modified in the given patch number in XML format.

Syntax

```sql
DBMS_QOPATCH.GET_OPATCH_FILES (patchnum IN VARCHAR2);
RETURN XMLTYPE;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>

127.5.6 GET_OPATCH_INSTALL_INFO Function

This function returns the XML element containing the ORACLE_HOME details such as patch and inventory location.

Syntax

```sql
DBMS_QOPATCH.GET_OPATCH_INSTALL_INFO
RETURNS XMLTYPE;
```

127.5.7 GET_OPATCH_LIST Function

This function provides list of patches installed as an XML element from the XML inventory.

Syntax

```sql
DBMS_QOPATCH.GET_OPATCH_LIST
RETURN XMLTYPE;
```

127.5.8 GET_OPATCH_LSIINVENTORY

This function returns whole opatch inventory as XML instance document.

Syntax

```sql
DBMS_QOPATCH.GET_OPATCH_LSIINVENTORY
RETURN XMLTYPE;
```
127.5.9 GET_OPATCH_OLAYS Function

This function provides overlay patches for a given patch as XML element.

Syntax

```
DBMS_QOPATCH.GET_OPATCH_OLAYS (
    patchnum IN VARCHAR2);
RETURN XMLTYPE;
```

Parameters

Table 127-8  GET_OPATCH_OLAYS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>

127.5.10 GET_OPATCH_PREQS Function

This function provides prerequisite patches for a given patch as XML element.

Syntax

```
DBMS_QOPATCH.GET_OPATCH_PREQS (
    patchnum IN VARCHAR2);
RETURN XMLTYPE;
```

Parameters

Table 127-9  GET_OPATCH_PREQS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>

127.5.11 GET_OPATCH_XSLT

This function returns the style-sheet for the opatch XML inventory presentation. You can use the return type of this subprogram to perform XMLTRANSFORM and the transformed result has the same appearance as opatch text output.

Syntax

```
DBMS_QOPATCH.GET_OPATCH_XSLT
RETURN XMLTYPE;
```

127.5.12 GET_PATCH_DETAILS Function

Displays the detailed patch information from opatch lsinventory.
Syntax

```sql
DBMS_QOPATCH.GET_PATCH_DETAILS (  
    patch IN VARCHAR2);  
RETURN XMLTYPE;
```

Parameters

**Table 127-10**  GET_PATCH_DETAILS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patch</td>
<td>The patch number.</td>
</tr>
</tbody>
</table>

127.5.13 GET_SQLPATCH_STATUS Procedure

This procedure displays the SQL patch status by querying from SQL patch registry to produce complete patch level information. If the patch number is given, it displays the information specific to the given SQL patch, otherwise information for all SQL patches.

Syntax

```sql
DBMS_QOPATCH.GET_SQLPATCH_STATUS (  
    patchnum IN VARCHAR2 DEFAULT NULL);  
```

Parameters

**Table 127-11**  GET_SQLPATCH_STATUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>

127.5.14 IS_PATCH_INSTALLED Function

This function provides information (such as patchID, application date, and SQL patch information) on the installed patch as XML node by querying the XML inventory.

Syntax

```sql
DBMS_QOPATCH.IS_PATCH_INSTALLED (  
    patchnum IN VARCHAR2);  
RETURN XMLTYPE;
```

Parameters

**Table 127-12**  IS_PATCH_INSTALLED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchnum</td>
<td>Patch number</td>
</tr>
</tbody>
</table>
127.5.15 OPATCH_COMPARE_CURRENT Function

The function compares the current database with a list of bugs.

**Syntax**

```sql
DBMS_QOPATCH.OPATCH_COMPARE_CURRENT (bugs IN QOPATCH_LIST);
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bugs</td>
<td>List of bugs to compare with the current database.</td>
</tr>
</tbody>
</table>

127.5.16 OPATCH_COMPARE_NODES Function

The function compares the given RAC node(s) with the instance to the present connected node with current node. The return value indicates whether the data was refreshed or not.

**Syntax**

```sql
DBMS_QOPATCH.OPATCH_COMPARE_NODES (node IN VARCHAR2 DEFAULT NULL,
inst  IN VARCHAR2 DEFAULT NULL);
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>Node name</td>
</tr>
<tr>
<td>inst</td>
<td>Instance name</td>
</tr>
</tbody>
</table>

127.5.17 SET_CURRENT_OPINST Procedure

This procedure sets the node name and instance to get the inventory details specific to it in an Oracle Real Application Clusters (RAC) environment.

**Syntax**

```sql
DBMS_QOPATCH.SET_CURRENT_OPINST (node_name IN VARCHAR2 DEFAULT NULL,
inst_name IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 127-15  SET_CURRENT_OPINST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_name</td>
<td>Name of node</td>
</tr>
<tr>
<td>inst_name</td>
<td>Name of instance</td>
</tr>
</tbody>
</table>
The DBMS_RANDOM package provides a built-in random number generator. DBMS_RANDOM is not intended for cryptography.

This chapter contains the following topics:

• Deprecated Subprograms
• Security Model
• Operational Notes
• Summary of DBMS_RANDOM Subprograms

128.1 DBMS_RANDOM Deprecated Subprograms

These subprograms are deprecated with Oracle Database 11g. Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

• INITIALIZE Procedure
• RANDOM Function
• TERMINATE Procedure

128.2 DBMS_RANDOM Security Model

This package should be installed as SYS. By default, the package is initialized with the current user name, current time down to the second, and the current session. Oracle recommends that users who need to execute this package should be given EXECUTE privilege explicitly and should not rely on PUBLIC EXECUTE privilege.

128.3 DBMS_RANDOM Operational Notes

These operational notes apply to DBMS_RANDOM.

• DBMS_RANDOM.RANDOM produces integers in [-2^31, 2^31).
• DBMS_RANDOM.VALUE produces numbers in [0,1) with 38 digits of precision.

DBMS_RANDOM can be explicitly initialized, but does not need to be initialized before calling the random number generator. It will automatically initialize with the date, user ID, and process ID if no explicit initialization is performed.

If this package is seeded twice with the same seed, then accessed in the same way, it will produce the same results in both cases.

In some cases, such as when testing, you may want the sequence of random numbers to be the same on every run. In that case, you seed the generator with a constant value by calling
one of the overloads of `DBMS_RANDOM.SEED`. To produce different output for every run, simply to omit the call to “Seed” and the system will choose a suitable seed for you.

### 128.4 Summary of DBMS_RANDOM Subprograms

This table lists the `DBMS_RANDOM` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALIZE Procedure</td>
<td>Initializes the package with a seed value</td>
</tr>
<tr>
<td>NORMAL Function</td>
<td>Returns random numbers in a normal distribution</td>
</tr>
<tr>
<td>RANDOM Function</td>
<td>Generates a random number</td>
</tr>
<tr>
<td>SEED Procedures</td>
<td>Resets the seed</td>
</tr>
<tr>
<td>STRING Function</td>
<td>Gets a random string</td>
</tr>
<tr>
<td>TERMINATE Procedure</td>
<td>Terminates package</td>
</tr>
<tr>
<td>VALUE Functions</td>
<td>Gets a random number, greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal (38-digit precision), while the overloaded function gets a random Oracle number x, where x is greater than or equal to low and less than high</td>
</tr>
</tbody>
</table>

### 128.4.1 INITIALIZE Procedure

This deprecated procedure initializes the generator.

**Note:**

This procedure is deprecated with Release 11gR1 and, although currently supported, it should not be used.

**Syntax**

```sql
DBMS_RANDOM.INITIALIZE (  
  val  IN  BINARY_INTEGER);
```

**Pragmas**

```sql
PRAGMA restrict_references (initialize, WNDS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Seed number used to generate a random number</td>
</tr>
</tbody>
</table>
Usage Notes

This procedure is obsolete as it simply calls the SEED Procedures.

128.4.2 NORMAL Function

This function returns random numbers in a standard normal distribution.

Syntax

DBMS_RANDOM.NORMAL
  RETURN NUMBER;

Pragmas

PRAGMA restrict_references (normal, WNDS);

Return Values

Table 128-3  NORMAL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Returns a random number</td>
</tr>
</tbody>
</table>

128.4.3 RANDOM Function

This deprecated procedure generates a random number.

Note:

This function is deprecated with Release 11gR1 and, although currently supported, it should not be used.

Syntax

DBMS_RANDOM.RANDOM
  RETURN binary_integer;

Pragmas

PRAGMA restrict_references (random, WNDS);

Return Values

Table 128-4  RANDOM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary_integer</td>
<td>Returns a random integer greater or equal to -power(2,31) and less than power(2,31)</td>
</tr>
</tbody>
</table>
128.4.4 SEED Procedures

This procedure resets the seed.

Syntax

DBMS_RANDOM.SEED (val IN BINARY_INTEGER);

DBMS_RANDOM.SEED (val IN VARCHAR2);

Pragmas

PRAGMA restrict_references (seed, WNDS);

Parameters

Table 128-5   SEED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Seed number or string used to generate a random number</td>
</tr>
</tbody>
</table>

Usage Notes

The seed can be a string up to length 2000.

128.4.5 STRING Function

This function gets a random string.

Syntax

DBMS_RANDOM.STRING opt IN CHAR, len IN NUMBER) RETURN VARCHAR2;

Pragmas

PRAGMA restrict_references (string, WNDS);
Parameters

Table 128-6  STRING Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt</td>
<td>Specifies what the returning string looks like:</td>
</tr>
<tr>
<td></td>
<td>- 'u', 'U' - returning string in uppercase alpha characters</td>
</tr>
<tr>
<td></td>
<td>- 'l', 'L' - returning string in lowercase alpha characters</td>
</tr>
<tr>
<td></td>
<td>- 'a', 'A' - returning string in mixed case alpha characters</td>
</tr>
<tr>
<td></td>
<td>- 'x', 'X' - returning string in uppercase alpha-numeric characters</td>
</tr>
<tr>
<td></td>
<td>- 'p', 'P' - returning string in any printable characters.</td>
</tr>
<tr>
<td></td>
<td>Otherwise the returning string is in uppercase alpha characters.</td>
</tr>
<tr>
<td>len</td>
<td>Length of the returning string</td>
</tr>
</tbody>
</table>

Return Values

Table 128-7  STRING Function Return Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>Returns a VARCHAR2</td>
</tr>
</tbody>
</table>

128.4.6 TERMINATE Procedure

When you are finished with the package, call the TERMINATE procedure.

Note:

This procedure is deprecated with Release 11gR1 and, although currently supported, it should not be used.

Syntax

DBMS_RANDOM.TERMINATE;

128.4.7 VALUE Functions

The basic function gets a random number, greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal (38-digit precision). Alternatively, you can get a random Oracle number x, where x is greater than or equal to low and less than high.

Syntax

DBMS_RANDOM.VALUE
RETURN NUMBER;

DBMS_RANDOM.VALUE(
    low IN NUMBER,
    high IN NUMBER)
RETURN NUMBER;
Parameters

Table 128-8  VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>Lowest number in a range from which to generate a random number. The number generated may be equal to low.</td>
</tr>
<tr>
<td>high</td>
<td>Highest number below which to generate a random number. The number generated will be less than high.</td>
</tr>
</tbody>
</table>

Return Values

Table 128-9  VALUE Function Return Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>Returns an Oracle Number</td>
</tr>
</tbody>
</table>
The DBMS_REDACT package provides an interface to Oracle Data Redaction, which enables you to mask (redact) data that is returned from queries issued by low-privileged users or an application.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Operating Procedures
- Summary of DBMS_REDACT Subprograms

See Also:

- Oracle Database Advanced Security Guide regarding using Data Redaction to protect sensitive data

### 129.1 DBMS_REDACT Overview

Data redaction provides a way to define masking policies for an application. Oracle Data Redaction provides functionality to mask (redact) data that is returned from user `SELECT` queries in an application. The masking takes place in real time. The Data Redaction policy applies to the querying user, depending on this user's `SYS_CONTEXT` or `XS_SYS_CONTEXT` values. This redaction process does not require that the queried data be static or unchanging, or for the entire data set to be redacted at one time in an off-line manner. Oracle Database redacts only the data for the rows specified by the user's query, not the data for the entire column. The redaction takes place immediately before the data is returned to the querying user or application.

### 129.2 DBMS_REDACT Security Model

If the querying user has the `EXEMPT REDACTION POLICY` system privilege, redaction will not be performed. If the user does not have the `EXEMPT REDACTION POLICY` system privilege, the policy expression will be evaluated in the current user's environment. If the policy expression evaluates to `TRUE`, then redaction will be performed, otherwise no redaction will be performed.

You need the `EXECUTE` privilege on the DBMS_REDACT package in order to execute its subprograms. Procedures in the interface are executed with privileges of the current user.
129.3 DBMS_REDACT Constants

The DBMS_REDACT package defines several constants for specifying parameter values.

### Table 129-1 Values for function_type Parameter of DBMS_REDACT.ADD_POLICY

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>0</td>
<td>BINARY_INTEGER</td>
<td>No redaction</td>
</tr>
<tr>
<td>FULL</td>
<td>1</td>
<td>BINARY_INTEGER</td>
<td>Redact to fixed values</td>
</tr>
<tr>
<td>NULLIFY</td>
<td>6</td>
<td>BINARY_INTEGER</td>
<td>Returns a null value as a redacted value</td>
</tr>
<tr>
<td>PARTIAL</td>
<td>2</td>
<td>BINARY_INTEGER</td>
<td>Partial redaction, redact a portion of the column data</td>
</tr>
<tr>
<td>RANDOM</td>
<td>4</td>
<td>BINARY_INTEGER</td>
<td>Random redaction, each query results in a different random value</td>
</tr>
<tr>
<td>REGEXP</td>
<td>5</td>
<td>BINARY_INTEGER</td>
<td>Regular expression based redaction</td>
</tr>
<tr>
<td>REGEXP_WIDTH</td>
<td>7</td>
<td>BINARY_INTEGER</td>
<td>Regular expression based redaction that preserves the width of a column that uses a regular expression; designed for applications use the OCI_ATTR_CHAR_SIZE attribute of the Oracle OLE DB Provider interface</td>
</tr>
</tbody>
</table>

### Table 129-2 Values for action Parameter of DBMS_REDACT.ALTER_POLICY

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_COLUMN</td>
<td>1</td>
<td>BINARY_INTEGER</td>
<td>Add a column to the redaction policy</td>
</tr>
<tr>
<td>DROP_COLUMN</td>
<td>2</td>
<td>BINARY_INTEGER</td>
<td>Drop a column from the redaction policy</td>
</tr>
<tr>
<td>MODIFY_EXPRESSION</td>
<td>3</td>
<td>BINARY_INTEGER</td>
<td>Modify the expression of a redaction policy (the expression evaluates to a BOOLEAN value: if TRUE then redaction is applied, otherwise not)</td>
</tr>
<tr>
<td>MODIFY_COLUMN</td>
<td>4</td>
<td>BINARY_INTEGER</td>
<td>Modify a column in the redaction policy to change the redaction function_type or the function_parameters</td>
</tr>
<tr>
<td>SET_POLICY_DESCRIPTION</td>
<td>5</td>
<td>BINARY_INTEGER</td>
<td>Set a description for the redaction policy</td>
</tr>
<tr>
<td>SET_COLUMN_DESCRIPTION</td>
<td>6</td>
<td>BINARY_INTEGER</td>
<td>Set a description for the redaction performed on the column</td>
</tr>
</tbody>
</table>
129.4 DBMS_REDACT Operating Procedures

The following table presents the relationship between the type of redaction function and its parameters, based on the datatype of the column being redacted. Examples of the various format strings are provided, showing how to perform some commonplace redaction for a string datatype (in this case, a Social Security Number (SSN)), a `DATE` datatype, and various examples of redaction for the number datatype.

Table 129-3  Data Redaction Function Types

<table>
<thead>
<tr>
<th>function_type</th>
<th>function_parameters</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_REDACT.NONE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DBMS_REDACT.FULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DBMS_REDACT.PARTIAL</td>
<td>(for character types)</td>
<td>A comma-separated list, containing the following five fields (with no spaces after the commas delimiting the fields):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REDACT_PARTIAL_INPUT_FORMAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REDACT_PARTIAL_OUTPUT_FORMAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REDACT_PARTIAL_MASKCHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REDACT_PARTIAL_MASKFROM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REDACT_PARTIAL_MASKTO</td>
</tr>
<tr>
<td>See Table 129-4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBMS_REDACT.PARTIAL</td>
<td>(for character types), continued</td>
<td>The REDACT_PARTIAL_INPUT_FORMAT field value VVVVVVVVVVVV for matching SSN strings like 123-45-6789.</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_INPUT_FORMAT</td>
<td>V for value to be possibly redacted, F for formatting character to be ignored</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DBMS_REDACT.PARTIAL</td>
<td>(for character types), continued</td>
<td>The REDACT_PARTIAL_OUTPUT_FORMAT field value VVVVVVVVVVVV can be used to redact SSN strings into XXX-XX-6789 (X comes from REDACT_PARTIAL_MASKCHAR field).</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_OUTPUT_FORMAT</td>
<td>V for output of redaction. Any other character will be treated as a formatting character and output literally.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DBMS_REDACT.PARTIAL</td>
<td>(for character types), continued</td>
<td>The value X for redacting SSN strings into XXX-XX-6789.</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKCHAR</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DBMS_REDACT.PARTIAL</td>
<td>(for character types), continued</td>
<td>The value 1 for redacting SSN strings starting at the first V of REDACT_PARTIAL_INPUT_FORMAT of VVVVVVVVVVV into strings like XXX-XX-6789.</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKFROM</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 129-3 (Cont.) Data Redaction Function Types

<table>
<thead>
<tr>
<th>function_type</th>
<th>function_parameters</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_REDACT.PARTIAL (for</td>
<td>REDACT_PARTIAL_MASKTO - specifies which V</td>
<td>The value 5 for redacting SSN strings up to and including the fifth V</td>
</tr>
<tr>
<td>character types), continued</td>
<td>within the REDACT_PARTIAL_INPUT_FORMAT at</td>
<td>within REDACT_PARTIAL_INPUT_FORMAT of VVVVVFVVVV into strings like XXX-</td>
</tr>
<tr>
<td></td>
<td>which to end the redaction</td>
<td>XX-6789. Note how the format character '-' (corresponding to the first F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>within REDACT_PARTIAL_INPUT_FORMAT) is ignored as far as redaction is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>concerned, so the value here is 5 as opposed to 6.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The REDACT_PARTIAL_MASKFROM and REDACT_PARTIAL_MASKTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>field values are specified as counts of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the number of V characters in the REDACT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARTIAL_INPUT_FORMAT at field, up to and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>including the intended position, starting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>from the leftmost V. This way, REDACT_PARTIAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARTIAL_MASKFROM and REDACT_PARTIAL_MASKTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>are independent of the specific formatting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of the data. For example, in the common use</td>
<td></td>
</tr>
<tr>
<td></td>
<td>case of masking an SSN to show only the last</td>
<td></td>
</tr>
<tr>
<td></td>
<td>four digits, data like 123456789 (with</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_INPUT_FORMAT of VVVVVVVVVV)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and data like 123-45-6789 (with REDACT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARTIAL_INPUT_FORMAT of VVVVFVFVVV), would</td>
<td></td>
</tr>
<tr>
<td></td>
<td>both use REDACT_PARTIAL_MASKFROM of 1 and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKTO of 5.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBMS_REDACT.PARTIAL (for</td>
<td>A comma-separated list, containing the</td>
<td>'9,1,5' for redacting the first 5 digits of an SSN number 123456789</td>
</tr>
<tr>
<td>number types)</td>
<td>following three fields (with no spaces after</td>
<td>into 999996789; or '0,1,2' for reducting a number 1.23 to 0.03.</td>
</tr>
<tr>
<td></td>
<td>the commas delimiting the fields):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• REDACT_PARTIAL_MASKCHAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• REDACT_PARTIAL_MASKFROM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• REDACT_PARTIAL_MASKTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Table 129-4.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 129-3 (Cont.) Data Redaction Function Types

<table>
<thead>
<tr>
<th>function_type</th>
<th>function_parameters</th>
<th>Examples</th>
</tr>
</thead>
</table>
| `DBMS_REDACT.PARTIAL` (for number types), continued | `REDACT_PARTIAL_MASKCHAR` - the character used to redact the input, in the range between 0 and 9  
`REDACT_PARTIAL_MASKFROM` - the position, starting from 1, from which to start the redaction. The position does not include the decimal point if it is present.  
`REDACT_PARTIAL_MASKTO` - the position at which to end the redaction | - |
| `DBMS_REDACT.PARTIAL` (for datetime datatypes) | A list, containing the following five fields (concatenated so that there is no space between the fields):  
- `REDACT_PARTIAL_DATE_MONTH`  
- `REDACT_PARTIAL_DATE_DAY`  
- `REDACT_PARTIAL_DATE_YEAR`  
- `REDACT_PARTIAL_DATE_HOUR`  
- `REDACT_PARTIAL_DATE_MINUTE`  
- `REDACT_PARTIAL_DATE_SECOND` | 'm12DyHMs', which changes 01-May-01 01:01:01 to 01-Dec-01 01:01:01.

See Table 129-4.
Table 129-3  (Cont.) Data Redaction Function Types

<table>
<thead>
<tr>
<th>function_type</th>
<th>function_parameters</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_REDACT.PARTIAL (for datetime datatypes), continued</td>
<td>REDACT_PARTIAL_DATE_MONTH: 'M' (no masking of month) or 'm#' (mask month to a specific month, if possible), where # (the month specified by its number) is between 1 and 12</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_DAY: 'D' (no masking of date) or 'd#' (mask day to #, if possible), # between 1 and 31</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_YEAR: 'Y' (no masking of year) or 'y#' (mask year to #, if possible), # between 1 and 9999</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_HOUR: 'H' (no masking of hour) or 'h#' (mask hour to #, if possible), # between 0 and 23</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_MINUTE: 'M' (no masking of minute) or 'm#' (mask minute to #, if possible), # between 0 and 59</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_SECOND: 'S' (no masking of second) or 's#' (mask second to #, if possible), # between 0 and 59</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DBMS_REDACT.PARTIAL_WIDTH</td>
<td>Same as</td>
</tr>
<tr>
<td></td>
<td>DBMS_REDACT.RANDOM</td>
<td>DBMS_REDACT.PARTIAL</td>
</tr>
</tbody>
</table>

Table 129-4  Format Descriptors with Component Field Names and Delimiters

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Format Descriptor for Partial redaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>REDACT_PARTIAL_INPUT_FORMAT</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_OUTPUT_FORMAT</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKCHAR</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKFROM</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKTO</td>
</tr>
<tr>
<td>Number</td>
<td>REDACT_PARTIAL_MASKCHAR</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKFROM</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_MASKTO</td>
</tr>
<tr>
<td>Datetime</td>
<td>REDACT_PARTIAL_DATE_MONTH</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_DAY</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_YEAR</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_HOUR</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_MINUTE</td>
</tr>
<tr>
<td></td>
<td>REDACT_PARTIAL_DATE_SECOND</td>
</tr>
</tbody>
</table>
129.5 Summary of DBMS_REDACT Subprograms

This table lists and briefly describes the DBMS_REDACT package subprograms.

Table 129-5  DBMS_REDACT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_POLICY Procedure</td>
<td>Defines a Data Redaction policy for a table or view</td>
</tr>
<tr>
<td>ALTER_POLICY Procedure</td>
<td>Alters a Data Redaction policy for a table or view</td>
</tr>
<tr>
<td>APPLY_POLICY_EXPR_TO_COL Procedure</td>
<td>Applies a Data Redaction policy expression to a redacted column</td>
</tr>
<tr>
<td>CREATE_POLICY_EXPRESSION Procedure</td>
<td>Creates a Data Redaction policy expression</td>
</tr>
<tr>
<td>DISABLE_POLICY Procedure</td>
<td>Disables a Data Redaction policy</td>
</tr>
<tr>
<td>DROP_POLICY Procedure</td>
<td>Drops a Data Redaction policy</td>
</tr>
<tr>
<td>DROP_POLICY_EXPRESSION Procedure</td>
<td>Drops a Data Redaction policy expression</td>
</tr>
<tr>
<td>ENABLE_POLICY Procedure</td>
<td>Enables a Data Redaction policy</td>
</tr>
<tr>
<td>UPDATE_FULL_REDACTION_VALUES Procedure</td>
<td>Modifies the default displayed values for a Data Redaction policy for full redaction</td>
</tr>
<tr>
<td>UPDATE_POLICY_EXPRESSION Procedure</td>
<td>Updates a Data Redaction policy expression</td>
</tr>
</tbody>
</table>

129.5.1 ADD_POLICY Procedure

This procedure defines a Data Redaction policy for a table or view.

Syntax

```sql
DBMS_REDACT.ADD_POLICY ( object_schema IN VARCHAR2 := NULL,
object_name IN VARCHAR2,
policy_name IN VARCHAR2,
column_name IN VARCHAR2 := NULL,
function_type IN BINARY_INTEGER := DBMS_REDACT.FULL,
function_parameters IN VARCHAR2 := NULL,
expression IN VARCHAR2,
enable IN BOOLEAN := TRUE,
regexp_pattern IN VARCHAR2 := NULL,
regexp_replace_string IN VARCHAR2 := NULL,
regexp_position IN BINARY_INTEGER := 1,
regexp_occurrence IN BINARY_INTEGER := 0,
regexp_match_parameter IN VARCHAR2 := NULL,
policy_description IN VARCHAR2 := NULL,
column_description IN VARCHAR2 := NULL);
```
## Parameters

### Table 129-6  ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema owning the table, current user if NULL</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view to which to add a Data Redaction policy</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy</td>
</tr>
<tr>
<td>column_name</td>
<td>[Optional] Name of one column to which the redaction policy applies. If you must redact more than one column, use the ALTER_POLICY Procedure to add the additional columns.</td>
</tr>
<tr>
<td>function_type</td>
<td>Type of redaction function to use. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.NONE</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.FULL (default)</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.NULLIFY</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.PARTIAL</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.RANDOM</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.REGEXP</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.REGEXP_WIDTH</td>
</tr>
</tbody>
</table>

If the function_type is DBMS_REDACT.REGEXP or DBMS_REDACT.REGEXP_WIDTH, then you must omit the function_parameters parameter, and use the regexp_* parameters to define the Data Redaction policy.

See Table 129-1 for an overview of the meanings of these values, and for some examples of their use.
Table 129-6  (Cont.) ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| function_parameters| Parameters to the redaction function. The possible values depend on the value of the function_type provided. If the function_type is DBMS_REDACT.REEXP or DBMS_REDACT.REEXP_WIDTH, then you must omit the function_parameters parameter, and use the regexp_* parameters to define the Data Redaction policy.  
  - DBMS_REDACT.NONE: Can be omitted entirely and defaults to NULL  
  - DBMS_REDACT.FULL: Can be omitted entirely and defaults to NULL  
  - Masking parameters for partial character masking. For character datatypes, a comma-separated list containing these fields:  
    • Input format: 'V' for value to be possibly masked, 'F' for formatting character to be ignored  
    • Output format: 'V' for output of masking, any other characters will be treated as formatting characters.  
    • Mask character: a character that will be used to replace the actual values. Examples are '*' and 'x'.  
    • Starting digit position: specifies the starting (character) position to begin replacing actual values with the masking character. The beginning of the string is position 1. Positions do not include formatting characters.  
    • Ending digit position: specifies the ending (character) position to end masking. An example is 'VVFFVVFVVVV, VVV-VV-VVVV, X, 1, 5' for masking the first 5 digits of SSN string 123-45-6789, and adding dashes back to format it like an SSN, resulting in XXX-XX-6789.  
  For number datatypes, a comma-separated list containing these fields:  
    • Mask character: this is a character between '0' to '9' that will be used to replace the actual values.  
    • Starting digit position: specifies the starting (digit) position to begin replacing actual values with the masking character. The beginning of the string is position 1. Positions do not include the decimal point.  
    • Ending digit position: this specifies the ending digit position to end masking. An example is '9, 1, 5' for masking the first 5 digits of a Social Security number number 123456789, resulting in 999996789.  
  For datetime datatypes, the format is a packed string (no spaces or commas) containing the following sequence of fields. Please note that each field can consist of one or more characters, and the field length depends on whether masking is required. The one-character fields are used to specify that no redaction of that component of the datetime value is to take place. The longer fields indicate a specific time or date to use as the redacted value of that component of the datetime value. |
Table 129-6  (Cont.) ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month: 'M' (no masking of month) or 'm#' (mask month to a specific month, if possible), where # (the month specified by its number) is between 1 and 12</td>
<td></td>
</tr>
<tr>
<td>Day: 'D' (no masking of date) or 'd#' (mask day to #, if possible), # between 1 and 31</td>
<td></td>
</tr>
<tr>
<td>Year: 'Y' (no masking of year) or 'y#' (mask year to #, if possible), # between 1 and 9999</td>
<td></td>
</tr>
<tr>
<td>Hour: 'H' (no masking of hour) or 'h#' (mask hour to #, if possible), # between 0 and 23</td>
<td></td>
</tr>
<tr>
<td>Minute: 'M' (no masking of minute) or 'm#' (mask minute to #, if possible), # between 0 and 59</td>
<td></td>
</tr>
<tr>
<td>Second: 'S' (no masking of second) or 's#' (mask second to #, if possible), # between 0 and 59</td>
<td></td>
</tr>
</tbody>
</table>

An example is 'm12d1y2001h1m1s1', which changes 02-May-13 12:30:23 to 01-Dec-01 01:01:01.

For partial character and number-masking shortcuts, see Oracle Database Advanced Security Guide.

expression

Default boolean expression for the table or view. If this expression is used, then redaction takes place only if this policy expression evaluates to TRUE.

See

The following functions are supported:

- SYS_CONTEXT
- XS_SYS_CONTEXT
- SUBSTR functions (includes SUBSTRB, SUBSTRC, SUBSTR2, SUBSTR4)
- LENGTH functions (includes LENGTHB, LENGTHC, LENGTH2, LENGTH4)
- V (APEX_UTIL.GET_SESSION_STATE)
- NV (APEX_UTIL.GET_NUMERIC_SESSION_STATE)
- OLS_LABEL_DOMINATES
- DOMINATES
- OLS_DOMINATES
- OLS_DOM
- DOM
- OLS_STRICTLY_DOMINATES
- STRICTLY_DOMINATES
- S_DOM
- SA_UTL.DOMINATES
- SA_UTL.CHECK_READ
- SA_UTL.NUMERIC_LABEL
- CHAR_TO_LABEL
- SA_SESSION_LABEL

See Oracle Database Advanced Security Guide for more information about these supported functions.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Boolean value that determines whether the Data Redaction policy is enabled on creation.</td>
</tr>
<tr>
<td></td>
<td>The default value is TRUE, which means that the policy is automatically enabled upon creation.</td>
</tr>
<tr>
<td></td>
<td>If the enable parameter is set to FALSE, the policy takes effect only when it is subsequently enabled by calling the DBMS_REDACT.ENABLE_POLICY procedure.</td>
</tr>
<tr>
<td>regexp_pattern</td>
<td>Regular expression pattern up to 512 bytes.</td>
</tr>
<tr>
<td></td>
<td>Use only if the function_type parameter is DBMS_REDACT.REGEXP. Also, do not specify the function_parameters parameter when function_type is DBMS_REDACT.REGEXP.</td>
</tr>
<tr>
<td></td>
<td>See Oracle Database SQL Language Reference for more information and examples on using regular expression patterns.</td>
</tr>
<tr>
<td>regexp_replace_string</td>
<td>Replacement string (up to 4000 characters in length) with up to 500 back-references to subexpressions in the form \n, where n is a number between 1 and 9.</td>
</tr>
<tr>
<td></td>
<td>Use only if the function_type parameter is DBMS_REDACT.REGEXP.</td>
</tr>
<tr>
<td>regexp_position</td>
<td>Integer counting from 1, specifies the position where the search must begin.</td>
</tr>
<tr>
<td></td>
<td>Use only if the function_type parameter is DBMS_REDACT.REGEXP.</td>
</tr>
<tr>
<td>regexp_occurrence</td>
<td>• Use 0 to replace all occurrences of the match</td>
</tr>
<tr>
<td></td>
<td>• Use positive integer n to replace the n-th occurrence of the match.</td>
</tr>
<tr>
<td></td>
<td>Use only if the function_type parameter is DBMS_REDACT.REGEXP.</td>
</tr>
<tr>
<td>regexp_match_parameter</td>
<td>Changes the default matching behavior, possible values are a combination of 'i', 'c', 'n', 'm', 'x'</td>
</tr>
<tr>
<td></td>
<td>Use only if the function_type parameter is DBMS_REDACT.REGEXP.</td>
</tr>
<tr>
<td></td>
<td>See Oracle Database SQL Language Reference for more information and examples on using regular expression match parameters.</td>
</tr>
<tr>
<td>policy_description</td>
<td>Description of redaction policy</td>
</tr>
<tr>
<td>column_description</td>
<td>Description of the column being redacted</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-28060 - A Data Redaction policy already exists on this column.
- ORA-28061 - This object cannot have a Data Redaction policy defined on it.
- ORA-28062 - The policy expression is too long.
- ORA-28063 - The policy expression is empty.
• ORA-28064 - The type of redaction function is not valid.
• ORA-28066 - Invalid column column
• ORA-28069 - A Data Redaction policy already exists on this object.
• ORA-28073 - The column column_name has an unsupported datatype.
• ORA-28074 - The field field_name of the masking parameters is not valid

The field can be any of the following:

- REDACT_PARTIAL_INPUT_FORMAT
- REDACT_PARTIAL_OUTPUT_FORMAT
- REDACT_PARTIAL_MASKCHAR
- REDACT_PARTIAL_MASKFROM
- REDACT_PARTIAL_MASKTO
- REDACT_PARTIAL_DATE_MONTH
- REDACT_PARTIAL_DATE_DAY
- REDACT_PARTIAL_DATE_YEAR
- REDACT_PARTIAL_DATE_HOUR
- REDACT_PARTIAL_DATE_MINUTE
- REDACT_PARTIAL_DATE_SECOND

See Table 129-3 and Table 129-4 for examples of the field contents and field ordering.

• ORA-28075 - The policy expression has unsupported functions
• ORA-28076 - An attribute was not specified for SYS_SESSION_ROLES
• ORA-28077 - The attribute specified (attribute) exceeds the maximum length
• ORA-28078 - A regular expression parameter is missing or invalid
• ORA-28082 - The parameter parameter is invalid (where the possible values are function_parameters, column_description, policy_name and policy_description)

• ORA-28085 - The input and output lengths of the redaction do not match.

Usage Notes

See Operating Procedures for more information regarding function types and function parameters with related examples.

A named Data Redaction policy expression that has been applied to a redacted column takes precedence over the expression defined in the expression parameter. To find redacted columns that are affected by named policy expressions, query the REDACTION_EXPRESSIONS data dictionary view.

Example

Partial redaction policy:
129.5.2 ALTER_POLICY Procedure

This procedure alters an existing Data Redaction policy for a table or view.

It alters the policy in one or more of the following ways:

- By changing the policy expression
- By changing the type of redaction for a specified column
- By changing the parameters to the redaction function for a specified column
- By adding a column to the redaction policy (the redaction type and any parameters must be specified).
- By removing a column from the redaction policy

Syntax

```sql
DBMS_REDACT.ALTER_POLICY ( 
    object_schema => 'hr', 
    object_name => 'employees', 
    policy_name => 'mask_emp_id_nums', 
    function_type => DBMS_REDACT.PARTIAL, 
    function_parameters => '7,1,5', 
    expression => '1=1');
END;
```

Full redaction policy:

```sql
BEGIN
DBMS_REDACT.ADD_POLICY(
    object_schema => 'hr', 
    object_name => 'employees', 
    column_name => 'employee_id', 
    policy_name => 'mask_emp_ids', 
    function_type => DBMS_REDACT.FULL, 
    expression => 'SYS_CONTEXT(''SYS_SESSION_ROLES'',''CLERK'') = ''FALSE''');
END;
```

129.5.2 ALTER_POLICY Procedure

This procedure alters an existing Data Redaction policy for a table or view.

It alters the policy in one or more of the following ways:

- By changing the policy expression
- By changing the type of redaction for a specified column
- By changing the parameters to the redaction function for a specified column
- By adding a column to the redaction policy (the redaction type and any parameters must be specified).
- By removing a column from the redaction policy

Syntax

```sql
DBMS_REDACT.ALTER_POLICY ( 
    object_schema => 'hr', 
    object_name => 'employees', 
    policy_name => 'mask_emp_id_nums', 
    function_type => DBMS_REDACT.PARTIAL, 
    function_parameters => '7,1,5', 
    expression => '1=1');
END;
```

Full redaction policy:

```sql
BEGIN
DBMS_REDACT.ADD_POLICY(
    object_schema => 'hr', 
    object_name => 'employees', 
    column_name => 'employee_id', 
    policy_name => 'mask_emp_ids', 
    function_type => DBMS_REDACT.FULL, 
    expression => 'SYS_CONTEXT(''SYS_SESSION_ROLES'',''CLERK'') = ''FALSE''');
END;
```
### Parameters

#### Table 129-7  ALTER_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema owning the table, current user if NULL</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view to which to alter a Data Redaction policy</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy limited to 30 bytes</td>
</tr>
<tr>
<td>action</td>
<td>Action to take. For more information see Table 129-2.</td>
</tr>
<tr>
<td>column_name</td>
<td>[Optional] Name of one column to which the redaction policy applies.</td>
</tr>
<tr>
<td>function_type</td>
<td>Type of redaction function to use. Possible values include:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.NONE</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.FULL (default)</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.NULLIY</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.PARTIAL</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.RANDOM</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.REEXP</td>
</tr>
<tr>
<td></td>
<td>- DBMS_REDACT.REGEXP_WIDTH</td>
</tr>
</tbody>
</table>

If the `function_type` is DBMS_REDACT.REEXP or DBMS_REDACT.REGEXP_WIDTH, then you must omit the function_parameters parameter, and use the `regexp_pattern`, `regexp_replace_string`, `regexp_position`, `regexp_occurrence`, and `regexp_match_parameter` to define the Data Redaction policy.

See Table 129-1 for an overview of the meanings of these values, and for some examples of their use.
Table 129-7   (Cont.) ALTER_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| function_parameters | Parameters to the redaction function. The possible values depend on the value of the function_type provided.  
If the function_type is DBMS_REDACT.REGEXP or DBMS_REDACT.REGEXP_WIDTH, then you must omit the function_parameters parameter, and use the regexp_pattern, regexp_replace_string, regexp_position, regexp_occurrence, and regexp_match_parameter to define the Data Redaction policy.  
- If the function_type is DBMS_REDACT.NONE, the function_parameters parameter can be omitted entirely and defaults to NULL.  
- If the function_type is DBMS_REDACT.FULL, the function_parameters parameter can be omitted entirely and defaults to NULL.  
- If the function_type is DBMS_REDACT.PARTIAL, the function_parameters parameter represents the masking parameters for partial masking.  
  • Input format: 'V' for value to be possibly masked, 'F' for formatting character to be ignored  
  • Output format: 'V' for output of masking, any other characters will be treated as formatting characters.  
  • Mask character: a character that will be used to replace the actual values. Examples are '*x' and 'x'.  
  • Starting digit position: specifies the starting (character) position to begin replacing actual values with the masking character. The beginning of the string is position 1. Positions do not include formatting characters.  
  • Ending digit position: specifies the ending (character) position to end masking. An example is 'VVVFVVFVVV,VVV-VV-VVVVV,X,1,5' for masking the first 5 digits of SSN string 123-45-6789, and adding dashes back to format it like an SSN, resulting in XXX-XX-6789.  
For number datatypes, a comma-separated list containing these fields:  
  • Mask character: this is a character between '0' to '9' that will be used to replace the actual values.  
  • Starting digit position: specifies the starting (digit) position to begin replacing actual values with the masking character. The beginning of the string is position 1. Positions do not include the decimal point.  
  • Ending digit position: this specifies the ending digit position to end masking. An example is '9,1,5' for masking the first 5 digits of a Social Security number number 123456789, resulting in 999996789.  
For datetime datatypes, the format is a packed string (no spaces or commas) containing the following sequence of fields. Please note that each field can consist of one or more characters, and the field length depends on whether masking is required. The one-character fields are used to specify that no
Table 129-7  (Cont.) ALTER_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>redaction of that component of the datetime value is to take place. The longer fields indicate a specific time or date to use as the redacted value of that component of the datetime value.</td>
</tr>
<tr>
<td></td>
<td>- Month: 'M' (no masking of month) or 'm#' (mask month to a specific month, if possible), where # (the month specified by its number) is between 1 and 12</td>
</tr>
<tr>
<td></td>
<td>- Day: 'D' (no masking of date) or 'd#' (mask day to #, if possible), # between 1 and 31</td>
</tr>
<tr>
<td></td>
<td>- Year: 'Y' (no masking of year) or 'y#' (mask year to #, if possible), # between 1 and 9999</td>
</tr>
<tr>
<td></td>
<td>- Hour: 'H' (no masking of hour) or 'h#' (mask hour to #, if possible), # between 0 and 23</td>
</tr>
<tr>
<td></td>
<td>- Minute: 'M' (no masking of minute) or 'm#' (mask minute to #, if possible), # between 0 and 59</td>
</tr>
<tr>
<td></td>
<td>- Second: 'S' (no masking of second) or 's#' (mask second to #, if possible), # between 0 and 59</td>
</tr>
<tr>
<td></td>
<td>An example is 'm12DYHMS', which changes 01-May-01 01:01:01 to 01-Dec-01 01:01:01</td>
</tr>
<tr>
<td></td>
<td>For partial character and number-masking shortcuts, see Oracle Database Advanced Security Guide.</td>
</tr>
<tr>
<td></td>
<td>expression Default boolean expression for the table or view. If this expression is used, then redaction takes place only if this policy expression evaluates to TRUE.</td>
</tr>
<tr>
<td></td>
<td>The following functions are supported:</td>
</tr>
<tr>
<td></td>
<td>- SYS_CONTEXT</td>
</tr>
<tr>
<td></td>
<td>- NV(APEX_UTIL.GET_NUMERIC_SESSION_STATE)</td>
</tr>
<tr>
<td></td>
<td>- V(APEX_UTIL.GET_SESSION_STATE)</td>
</tr>
<tr>
<td></td>
<td>- OLS_LABEL_DOMINATES</td>
</tr>
<tr>
<td></td>
<td>- DOMINATES</td>
</tr>
<tr>
<td></td>
<td>- OLS_DOMINATES</td>
</tr>
<tr>
<td></td>
<td>- OLS_DOM</td>
</tr>
<tr>
<td></td>
<td>- DOM</td>
</tr>
<tr>
<td></td>
<td>- OLS STRICTLY DOMINATES</td>
</tr>
<tr>
<td></td>
<td>- STRICTLY DOMINATES</td>
</tr>
<tr>
<td></td>
<td>- S_DOM</td>
</tr>
<tr>
<td></td>
<td>- SA_UTL.DOMINATES</td>
</tr>
<tr>
<td></td>
<td>- SA_UTL.CHECK_READ</td>
</tr>
<tr>
<td></td>
<td>- SA_UTL.NUMERIC_LABEL</td>
</tr>
<tr>
<td></td>
<td>- CHAR_TO_LABEL</td>
</tr>
<tr>
<td></td>
<td>- SA_SESSION.LABEL</td>
</tr>
<tr>
<td></td>
<td>regexp_pattern Regular expression pattern up to 512 bytes.</td>
</tr>
<tr>
<td></td>
<td>Use only if the function_type parameter is DBMS_REDACT.REGEXP. Also, do not specify the function_parameters parameter when function_type is DBMS_REDACT.REGEXP.</td>
</tr>
<tr>
<td></td>
<td>See Oracle Database SQL Language Reference for more information and examples on using regular expression patterns</td>
</tr>
</tbody>
</table>
Table 129-7  (Cont.) ALTER_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regexp_replace_string</td>
<td>Replacement string (up to 4000 characters in length) with up to 500 back-references to subexpressions in the form \n, where n is a number between 1 and 9. Use only if the function_type parameter is DBMS_REDACT.REGEXP</td>
</tr>
<tr>
<td>regexp_position</td>
<td>Integer counting from 1, specifies the position where the search must begin. Use only if the function_type parameter is DBMS_REDACT.REGEXP</td>
</tr>
</tbody>
</table>
| regexp_occurrence      | • Use 0 to replace all occurrences of the match  
• Use positive integer n to replace the n-th occurrence of the match. Use only if the function_type parameter is DBMS_REDACT.REGEXP |
| regexp_match_parameter | Changes the default matching behavior, possible values are a combination of 'i', 'c', 'n', 'm', 'x' Use only if the function_type parameter is DBMS_REDACT.REGEXP  
See Oracle Database SQL Language Reference for more information and examples on using regular expression match parameters. |
| policy_description     | Description of redaction policy |
| column_description     | Description of the column being redacted |

Exceptions

- ORA-28062 - The policy expression is too long.
- ORA-28063 - The policy expression is empty.
- ORA-28064 - The type of redaction function is not valid.
- ORA-28066 - Invalid column column
- ORA-28067 - Missing or invalid column name
- ORA-28068 - The object object does not have a Data Redaction policy.
- ORA-28070 - The column column does not have a Data Redaction policy.
- ORA-28071 - The action is not valid.
- ORA-28072 - The specified policy name is incorrect.
- ORA-28073 - The column column_name has an unsupported datatypes.
- ORA-28074 - The field field_name of the masking parameters is not valid

The field can be any of the following:
- REDACT_PARTIAL_INPUT_FORMAT
- REDACT_PARTIAL_OUTPUT_FORMAT
See Table 129-3 and Table 129-4 for examples of the field contents and field ordering.

- ORA-28075 - The policy expression has unsupported functions.
- ORA-28076 - An attribute was not specified for SYS_SESSION_ROLES.
- ORA-28077 - The attribute specified (attribute) exceeds the maximum length.
- ORA-28078 - A regular expression parameter is missing or invalid.
- ORA-28082 - The parameter parameter is invalid (where the possible values are function_parameters, column_description, policy_name and policy_description)
- ORA-28085 - The input and output lengths of the redaction do not match.

Usage Notes

See Operating Procedures for more information regarding Function Types and Function Parameters with related examples.

A named Data Redaction policy expression that has been applied to a redacted column takes precedence over the expression defined in the expression parameter. To find redacted columns that are affected by named policy expressions, query the REDACTION_EXPRESSIONS data dictionary view.

Examples

BEGIN
DBMS_REDACT.ALTER_POLICY(
    object_schema => 'HR',
    object_name => 'EMPLOYEES',
    policy_name => 'mask_emp_id_nums',
    action => DBMS_REDACT.DROP_COLUMN,
    column_name => 'EMAIL');
END;
129.5.3 APPLY_POLICY_EXPR_TO_COL Procedure

This procedure associates a named Oracle Data Redaction policy expression with a redacted column from a table or view.

Syntax

```sql
DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL (
    object_schema    IN VARCHAR2 := NULL,
    object_name      IN VARCHAR2,
    column_name      IN VARCHAR2,
    policy_expression_name IN VARCHAR2 := NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Name of the schema that contains the redacted column</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object (table or view) that contains the redacted column</td>
</tr>
<tr>
<td>column_name</td>
<td>Name of the redacted column to which the policy expression is applied</td>
</tr>
<tr>
<td>policy_expression_name</td>
<td>Name of the policy expression</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-28068 - The object `object` does not have a Data Redaction policy.
- ORA-28082 - The parameter `parameter` is invalid.
- ORA-28092 - The parameter `parameter` with value `value` has an error.

Usage Notes

You can find existing Data Redaction policy expressions by querying the `REDACTION_EXPRESSIONS` data dictionary view. To find columns that have been redacted, query the `REDACTION_COLUMNS` data dictionary view.

Example

```sql
BEGIN
    DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL(
        object_schema => 'OE',
        object_name   => 'CUSTOMERS',
        column_name   => 'INCOME_LEVEL',
        policy_expression_name => 'oe_redact_pol');
END;
```
129.5.4 CREATE_POLICY_EXPRESSION Procedure

This procedure creates a named Oracle Data Redaction policy expression.

Syntax

```sql
DBMS_REDACT.CREATE_POLICY_EXPRESSION (    
  policy_expression_name    IN    VARCHAR2,    
  expression                IN    VARCHAR2,    
  policy_expression_description IN VARCHAR2 := NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_expression_name</td>
<td>Name of the policy expression</td>
</tr>
<tr>
<td>expression</td>
<td>Definition of the policy expression</td>
</tr>
<tr>
<td>policy_expression_description</td>
<td>Description of the policy expression</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-28082 - The parameter `parameter` is invalid.
- ORA-28092 - The parameter `parameter` with value `value` has an error.

Usage Notes

See Operating Procedures for more information regarding function types and function parameters with related examples.

After you create a policy expression, you can associate it with a redacted table or view column by running the `DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL` procedure. To find existing redacted columns, query the `REDACTION_COLUMNS` data dictionary view.

Example

```sql
BEGIN
  DBMS_REDACT.CREATE_POLICY_EXPRESSION (    
    policy_expression_name => 'oe_redact_pol',    
    expression => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') = ''OE'''),    
    policy_expression_description => 'Enables policy for user OE ');
END;
```

129.5.5 DISABLE_POLICY Procedure

This procedure disables a Data Redaction policy.

Syntax

```sql
DBMS_REDACT.DISABLE_POLICY (    
  object_schema     IN    VARCHAR2 := NULL,    
  ...)```

object_name                  IN    VARCHAR2,
policy_name                  IN    VARCHAR2);

Parameters

Table 129-10  DISABLE_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema owning the table or view, current user if NULL</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view for which to disable a Data Redaction policy</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be disabled</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-28068 - The object object does not have a Data Redaction policy.
- ORA-28072 - The specified policy name is incorrect.
- ORA-28080 - The policy was already disabled.

Examples

```
BEGIN
    DBMS_REDACT.DISABLE_POLICY (object_schema => 'hr',
                                object_name => 'employees',
                                policy_name => 'mask_emp_ids');
END;
```

129.5.6 DROP_POLICY Procedure

This procedure drops a Data Redaction policy by removing a masking policy from the table or view.

Syntax

```
DBMS_REDACT.DROP_POLICY (object_schema    IN    VARCHAR2 := NULL,
                         object_name      IN    VARCHAR2,
                         policy_name      IN    VARCHAR2);
```

Parameters

Table 129-11  DROP_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema owning the table or view, current user if NULL</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view from which to drop a Data Redaction policy</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be dropped</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-28068 - The object object does not have a Data Redaction policy.
• ORA-28072 - The specified policy name is incorrect.

**Examples**

```sql
BEGIN
    DBMS_REDACT.DROP_POLICY (
        object_schema => 'hr',
        object_name   => 'employees',
        policy_name   => 'mask_emp_ids');
END;
```

### 129.5.7 DROP_POLICY_EXPRESSION Procedure

This procedure drops a named policy expression.

**Syntax**

```sql
DBMS_REDACT.DROP_POLICY_EXPRESSION (
    policy_expression_name IN VARCHAR2);
```

**Parameters**

**Table 129-12**  
**DROP_POLICY_EXPRESSION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_expression_name</td>
<td>Name of the policy expression</td>
</tr>
</tbody>
</table>

**Exceptions**

- ORA-28082 - The parameter `policy_expression_name` is invalid.
- ORA-28092 - The parameter `policy_expression_name` with value `value` has an error.

**Usage Notes**

You can find existing Data Redaction policy expressions by querying the `REDACTION_EXPRESSIONS` data dictionary view.

**Example**

```sql
BEGIN
    DBMS_REDACT.DROP_POLICY_EXPRESSION(
        policy_expression_name => 'oe_redact_pol');
END;
```

### 129.5.8 ENABLE_POLICY Procedure

This procedure re-enables a Data Redaction policy.

**Syntax**

```sql
DBMS_REDACT.ENABLE_POLICY (
    object_schema            IN VARCHAR2 := NULL,
    object_name              IN VARCHAR2,
    policy_name              IN VARCHAR2);
```
Parameters

Table 129-13  ENABLE_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema owning the table or view, current user if NULL</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table or view to which to enable a Data Redaction policy</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be enabled</td>
</tr>
</tbody>
</table>

Exceptions

• ORA-28068 - The object object does not have a Data Redaction policy.
• ORA-28071 - The action is not valid.
• ORA-28072 - The specified policy name is incorrect.
• ORA-28079 - The policy was already enabled.

Examples

BEGIN
    DBMS_REDACT.ENABLE_POLICY (
        object_schema => 'hr',
        object_name   => 'employees',
        policy_name   => 'mask_emp_ids');
END;

129.5.9 UPDATE_FULL_REDACTION_VALUES Procedure

This procedure modifies the default displayed values for a Data Redaction policy for full redaction.

Syntax

DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES (  
    number_val IN NUMBER := NULL,  
    binfloat_val IN BINARY_FLOAT := NULL,  
    bindouble_val IN BINARY_DOUBLE := NULL,  
    char_val IN CHAR := NULL,  
    varchar_val IN VARCHAR2 := NULL,  
    nchar_val IN NCHAR := NULL,  
    nvarchar_val IN NVARCHAR2 := NULL,  
    date_val IN DATE := NULL,  
    ts_val IN TIMESTAMP := NULL,  
    tswtz_val IN TIMESTAMP WITH TIME ZONE := NULL,  
    blob_val IN BLOB := NULL,  
    clob_val IN CLOB := NULL,  
    nclob_val IN NCLOB := NULL);


Parameters

Table 129-14  UPDATE_FULL_REDACTION_VALUES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number_val</td>
<td>Modifies the default value for columns of the NUMBER datatype</td>
</tr>
<tr>
<td>binfloat_val</td>
<td>Modifies the default value for columns of the BINARY_FLOAT datatype</td>
</tr>
<tr>
<td>bindouble_val</td>
<td>Modifies the default value for columns of the BINARY_DOUBLE datatype</td>
</tr>
<tr>
<td>char_val</td>
<td>Modifies the default value for columns of the CHAR datatype</td>
</tr>
<tr>
<td>varchar_val</td>
<td>Modifies the default value for columns of the VARCHAR2 datatype</td>
</tr>
<tr>
<td>nchar_val</td>
<td>Modifies the default value for columns of the NCHAR datatype</td>
</tr>
<tr>
<td>nvarchar_val</td>
<td>Modifies the default value for columns of the NVARCHAR2 datatype</td>
</tr>
<tr>
<td>date</td>
<td>Modifies the default value for columns of the DATE datatype</td>
</tr>
<tr>
<td>ts_val</td>
<td>Modifies the default value for columns of the TIMESTAMP datatype</td>
</tr>
<tr>
<td>tswtz_val</td>
<td>Modifies the default value for columns of the TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>blob_val</td>
<td>Modifies the default value for columns of the BLOB datatype</td>
</tr>
<tr>
<td>clob_val</td>
<td>Modifies the default value for columns of the CLOB datatype</td>
</tr>
<tr>
<td>nclob_val</td>
<td>Modifies the default value for columns of the NCLOB datatype</td>
</tr>
</tbody>
</table>

Exceptions

ORA-28082 - The parameter parameter is invalid (where the possible values are char_val, nchar_val, varchar_val and nvarchar_val)

129.5.10 UPDATE_POLICY_EXPRESSION Procedure

This procedure updates a named Oracle Data Redaction policy expression.

Syntax

```sql
DBMS_REDACT.UPDATE_POLICY_EXPRESSION (  
  policy_expression_name    IN    VARCHAR2,  
  expression               IN    VARCHAR2,  
  policy_expression_description IN VARCHAR2 := NULL);
```

Parameters

Table 129-15  UPDATE_POLICY_EXPRESSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_expression_name</td>
<td>Name of the policy expression</td>
</tr>
<tr>
<td>expression</td>
<td>Definition of the policy expression</td>
</tr>
<tr>
<td>policy_expression_description</td>
<td>Description of the policy expression</td>
</tr>
</tbody>
</table>
Exceptions

- ORA-28082 - The parameter *parameter* is invalid.
- ORA-28092 - The parameter *parameter* with value *value* has an error.

Usage Notes

You can find existing policy expressions by querying the `REDACTION_EXPRESSIONS` data dictionary view.

Example

```sql
BEGIN
   DBMS_REDACT.UPDATE_POLICY_EXPRESSION(
      policy_expression_name => 'oe_redact_pol',
      expression => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') != ''OE'''),
      policy_expression_description => 'Disables policy for user OE ');
END;
```
130

DBMS_REDEFINITION

The DBMS_REDEFINITION package provides an interface to perform an online redefinition of tables.

This chapter contains the following topics:

• Overview
• Security Model
• Constants
• Operational Notes
• Rules and Limits
• Examples
• Summary of DBMS_REDEFINITION Subprograms

See Also:
Oracle Database Administrator's Guide for more information about online redefinition of tables

130.1 DBMS_REDEFINITION Overview

To achieve online redefinition, incrementally maintainable local materialized views are used. These logs keep track of the changes to the master tables and are used by the materialized views during refresh synchronization.

130.2 DBMS_REDEFINITION Security Model

Subprograms in the DBMS_REDEFINITION package are run with invokers' rights (with the privileges of the current user).

There are two modes:

• In USER mode, the user who has the CREATE TABLE and CREATE MVIEW privileges may redefine a table residing in his own schema.
• In FULL mode, the user who has the ANY privilege may redefine tables in any schema.

130.3 DBMS_REDEFINITION Constants

The DBMS_REDEFINITION package defines several constants for specifying parameter values.
### Table 130-1  DBMS_REDEFINITION Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONS_CONSTRAINT</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td>Used to specify that dependent object type is a constraint</td>
</tr>
<tr>
<td>CONS_INDEX</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>Used to specify that dependent object type is an index</td>
</tr>
<tr>
<td>CONS_MVLOG</td>
<td>PLS_INTEGER</td>
<td>10</td>
<td>Used to (un)register a materialized view log, as a dependent object of the table, through the REGISTER_DEPENDENT_OBJECT Procedure and the UNREGISTER_DEPENDENT_OBJECT Procedure.</td>
</tr>
<tr>
<td>CONS_ORIG_PARAMS</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Used to specify that indexes should be cloned with their original storage parameters</td>
</tr>
<tr>
<td>CONS_TRIGGER</td>
<td>PLS_INTEGER</td>
<td>4</td>
<td>Used to specify that dependent object type is a trigger</td>
</tr>
<tr>
<td>CONS_USE_PK</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Used to indicate that the redefinition should be done using primary keys or pseudo-primary keys (unique keys with all component columns having NOT NULL constraints)</td>
</tr>
<tr>
<td>CONS_USE_ROWID</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Used to indicate that the redefinition should be done using rowids</td>
</tr>
<tr>
<td>CONS_VPD_AUTO</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Used to indicate to copy VPD policies automatically</td>
</tr>
<tr>
<td>CONS_VPD_MANUAL</td>
<td>BINARY_INTEGER</td>
<td>4</td>
<td>Used to indicate to copy VPD policies manually</td>
</tr>
<tr>
<td>CONS_VPD_NONE</td>
<td>BINARY_INTEGER</td>
<td>1</td>
<td>Used to indicate that there are no VPD policies on the original table</td>
</tr>
</tbody>
</table>

### 130.4 DBMS_REDEFINITION Operational Notes

The following operational notes apply to DBMS_REDEFINITION.

- CONS_USE_PK and CONS_USE_ROWID are constants used as input to the "options_flag" parameter in both the START_REDEF_TABLE Procedure and CAN_REDEF_TABLE Procedure. CONS_USE_ROWID is used to indicate that the redefinition should be done using rowids while CONS_USE_PK implies that the redefinition should be done using primary keys or pseudo-primary keys (which are unique keys with all component columns having NOT NULL constraints).

- CONS_INDEX, CONS_MVLOG, CONS_TRIGGER and CONS_CONSTRAINT are used to specify the type of the dependent object being (un)registered in REGISTER_DEPENDENT_OBJECT Procedure and UNREGISTER_DEPENDENT_OBJECT Procedure (parameter "dep_type").

CONS_INDEX ==> dependent object is of type INDEX
CONS_TRIGGER ==> dependent object is of type TRIGGER
CONS_CONSTRAINT===> dependent object type is of type CONSTRAINT
CONS_MVLOG => dependent object is of type MATERIALIZED VIEW LOG

- CONS_ORIG_PARAMS as used as input to the "copy_indexes" parameter in COPY_TABLE_DEPENDENTS Procedure. Using this parameter implies that the indexes on the original table be copied onto the interim table using the same storage parameters as that of the original index.

130.5 DBMS_REDEFINITION Rules and Limits

Various rules and limits apply to implementation of the DBMS_REDEFINITION package.

For more information about these, see the Oracle Database Administrator's Guide.

130.6 DBMS_REDEFINITION Examples

The following examples demonstrate DBMS_REDEFINITION functionality.

We create two tables EMP and EMP_INT as the original and the interim tables, respectively:

```sql
CREATE TABLE "EMP"
( "EMPNO" NUMBER(4,0) PRIMARY KEY,
  "ENAME" VARCHAR2(10),
  "JOB" VARCHAR2(9),
  "MGR" NUMBER(4,0),
  "HIREDATE" DATE,
  "SAL" NUMBER(7,2),
  "COMM" NUMBER(7,2),
  "DEPTNO" NUMBER(2,0))
TABLESPACE myts;

CREATE TABLE "EMP_INT"
( "EMPNO" NUMBER(4,0) PRIMARY KEY,
  "ENAME" VARCHAR2(10),
  "JOB" VARCHAR2(9),
  "MGR" NUMBER(4,0),
  "HIREDATE" DATE,
  "SAL" NUMBER(7,2),
  "COMM" NUMBER(7,2),
  "DEPTNO" NUMBER(2,0))
TABLESPACE compressed_ts;
```

Regular Multi-Step Redefinition

```sql
DBMS_REDEFINITION.START_REDEF_TABLE('SCOTT', 'EMP', 'EMP_INT',
ENABLE_ROLLBACK => TRUE);
DBMS_REDEFINITION.FINISH_REDEF_TABLE('SCOTT', 'EMP', 'EMP_INT');
```

Assume the DBA wants to evaluate the performance of the application for 2 days, after moving the table EMP from tablespace myts to compressed_ts. One can run
sync_interim_table  

**SYNC_INTERIM_TABLE Procedure**

To keep both the tables in sync (say, every hour).

```sql
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('SCOTT', 'EMP', 'EMP_INT');
```

**Case 1 — DBA is not happy with the performance, so decides to rollback.**

```sql
DBMS_REDEFINITION.ROLLBACK('SCOTT', 'EMP', 'EMP_INT');
```

**Case 2 — DBA is happy with the performance, so decides not to rollback.**

```sql
DBMS_REDEFINITION.ABORT_ROLLBACK('SCOTT', 'EMP', 'EMP_INT');
```

This terminates the possibility of rollback.

**Single-Step Redefinition**

```sql
DBMS_REDEFINITION.REDEF_TABLE('SCOTT', 'EMP', 'ROW STORE COMPRESS ADVANCED', enable_rollback => TRUE);
```

---

**Note:**

Online table redefinition rollback is not supported when the **REDEF_TABLE** procedure is used to redefine a table.

### 130.7 Summary of DBMS_REDEFINITION Subprograms

This table lists the **DBMS_REDEFINITION** subprograms and briefly describes them.

**Table 130-2  DBMS_REDEFINITION Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT_REDEF_TABLE</td>
<td>Cleans up errors that occur during the redefinition process and removes all temporary objects created by the reorganization process</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>ABORT_ROLLBACK</td>
<td>Aborts rollback</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>ABORT_UPDATE</td>
<td>Aborts an update started with the <strong>EXECUTE_UPDATE</strong> procedure</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>CAN_REDEF_TABLE</td>
<td>Determines if a given table can be redefined online</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>COPY_TABLE_DEPENDENTS</td>
<td>Copies the dependent objects of the original table onto the interim table</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>EXECUTE_UPDATE</td>
<td>Optimizes the performance of bulk updates to a table</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>FINISH_REDEF_TABLE</td>
<td>Completes the redefinition process</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
</tbody>
</table>
### Table 130-2  (Cont.) DBMS_REDEFINITION Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REDEF_TABLE Procedure</td>
<td>Provides a single push-button interface that integrates several redefinition steps</td>
</tr>
<tr>
<td>REGISTER_DEPENDENT_OBJECT Procedure</td>
<td>Registers a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table</td>
</tr>
<tr>
<td>ROLLBACK Procedure</td>
<td>Performs rollback</td>
</tr>
<tr>
<td>SET_PARAM Procedure</td>
<td>Sets a new value for a specified parameter used by the redefinition process identified by a redefinition ID</td>
</tr>
<tr>
<td>START_REDEF_TABLE Procedure</td>
<td>Initiates the redefinition process</td>
</tr>
<tr>
<td>SYNC_INTERIM_TABLE Procedure</td>
<td>Keeps the interim table synchronized with the original table</td>
</tr>
<tr>
<td>UNREGISTER_DEPENDENT_OBJECT Procedure</td>
<td>Unregisters a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table</td>
</tr>
</tbody>
</table>

#### 130.7.1 ABORT_REDEF_TABLE Procedure

This procedure cleans up errors that occur during the redefinition process.

This procedure can also be used to terminate the redefinition process any time after the START_REDEF_TABLE Procedure has been called and before the FINISH_REDEF_TABLE Procedure is called. This process will remove the temporary objects that are created by the redefinition process such as materialized view logs.

### Syntax

```
DBMS_REDEFINITION.ABORT_REDEF_TABLE (  
    uname                   IN  VARCHAR2,  
    orig_table              IN  VARCHAR2,  
    int_table               IN  VARCHAR2,  
    part_name               IN  VARCHAR2 := NULL);
```

### Parameters

#### Table 130-3  ABORT_REDEF_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the tables</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table. Can take a comma-delimited list of interim table names.</td>
</tr>
<tr>
<td>part_name</td>
<td>Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined. Can take a comma-delimited list of partition names to be redefined.</td>
</tr>
</tbody>
</table>
130.7.2 ABORT_ROLLBACK Procedure

This procedure aborts rollback for a table that was redefined.

When online redefinition of a table is started with the `START_REDEF_TABLE` procedure, rollback can be enabled for the changes performed by online redefinition of a table by setting the `enable_rollback` parameter to `TRUE`. If you want to retain the changes made by online redefinition, you can abort the rollback to clean up the database objects that enable rollback.

**Syntax**

```sql
DBMS_REDEFINITION.ABORT_ROLLBACK (
    uname           IN  VARCHAR2,
    orig_table      IN  VARCHAR2,
    int_table       IN  VARCHAR2 := NULL,
    part_name       IN  VARCHAR2 := NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the tables</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table</td>
</tr>
<tr>
<td>part_name</td>
<td>Name of the partition being redefined</td>
</tr>
</tbody>
</table>

130.7.3 ABORT_UPDATE Procedure

This procedure can aborts an update started with the `EXECUTE_UPDATE` procedure in the `RDBMS_REDEFINITION` package.

**Syntax**

```sql
DBMS_REDEFINITION.ABORT_UPDATE (    
    update_stmt  IN  CLOB);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update_stmt</td>
<td>The SQL <code>UPDATE</code> statement to be aborted</td>
</tr>
<tr>
<td></td>
<td>The SQL statement must exactly match the SQL statement in the <code>EXECUTE_UPDATE</code> procedure.</td>
</tr>
</tbody>
</table>
130.7.4 CAN_REDEF_TABLE Procedure

This procedure determines if a given table can be redefined online. This is the first step of the online redefinition process. If the table is not a candidate for online redefinition, an error message is raised.

Syntax

```sql
DBMS_REDEFINITION.CAN_REDEF_TABLE (
    uname  IN  VARCHAR2,
    tname  IN  VARCHAR2,
    options_flag  IN  PLS_INTEGER := 1,
    part_name     IN  VARCHAR2 := NULL);
```

Parameters

Table 130-6  CAN_REDEF_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the table</td>
</tr>
<tr>
<td>tname</td>
<td>Name of the table to be re-organized</td>
</tr>
<tr>
<td>options_flag</td>
<td>Indicates the type of redefinition method to use.</td>
</tr>
<tr>
<td></td>
<td>• If dbms_redefinition.cons_use_pk, the redefinition is done using primary</td>
</tr>
<tr>
<td></td>
<td>keys or pseudo-primary keys (unique keys with all component columns having</td>
</tr>
<tr>
<td></td>
<td>NOT NULL constraints). The default method of redefinition is using primary</td>
</tr>
<tr>
<td></td>
<td>keys.</td>
</tr>
<tr>
<td></td>
<td>• If dbms_redefinition.cons_use_rowid, the redefinition is done using rowids.</td>
</tr>
<tr>
<td>part_name</td>
<td>Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.</td>
</tr>
</tbody>
</table>

Exceptions

If the table is not a candidate for online redefinition, an error message is raised.

130.7.5 COPY_TABLE_DEPENDENTS Procedure

This procedure clones the dependent objects of the table being redefined onto the interim table and registers the dependent objects. This procedure does not clone the already registered dependent objects.

This subprogram is used to clone the dependent objects like grants, triggers, constraints and privileges from the table being redefined to the interim table (which represents the post-redefinition table).
Syntax

```sql
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname                    IN  VARCHAR2,
    orig_table               IN  VARCHAR2,
    int_table                IN  VARCHAR2,
    copy_indexes             IN  PLS_INTEGER := 1,
    copy_triggers            IN  BOOLEAN     := TRUE,
    copy_constraints         IN  BOOLEAN     := TRUE,
    copy_privileges          IN  BOOLEAN     := TRUE,
    ignore_errors            IN  BOOLEAN     := FALSE,
    num_errors               OUT PLS_INTEGER,
    copy_statistics          IN  BOOLEAN     := FALSE,
    copy_mvlog               IN  BOOLEAN     := FALSE);
```

Parameters

Table 130-7  COPY_TABLE_DEPENDENTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the tables</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table being redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table</td>
</tr>
<tr>
<td>copy_indexes</td>
<td>Flag indicating whether to copy the indexes</td>
</tr>
<tr>
<td></td>
<td>• 0 - do not copy any index</td>
</tr>
<tr>
<td></td>
<td>• dbms_redefinition.cons_orig_params - copy the indexes using the physical parameters of the source indexes</td>
</tr>
<tr>
<td>copy_triggers</td>
<td>TRUE = clone triggers, FALSE = do nothing</td>
</tr>
<tr>
<td>copy_constraints</td>
<td>TRUE = clone constraints, FALSE = do nothing. If compatibility setting is 10.2 or higher, then clone CHECK and NOT NULL constraints</td>
</tr>
<tr>
<td>copy_privileges</td>
<td>TRUE = clone privileges, FALSE = do nothing</td>
</tr>
<tr>
<td>ignore_errors</td>
<td>TRUE = if an error occurs while cloning a particular dependent object, then skip that object and continue cloning other dependent objects. FALSE = that the cloning process should stop upon encountering an error.</td>
</tr>
<tr>
<td>num_errors</td>
<td>Number of errors that occurred while cloning dependent objects</td>
</tr>
<tr>
<td>copy_statistics</td>
<td>TRUE = copy statistics, FALSE = do nothing</td>
</tr>
<tr>
<td>copy_mvlog</td>
<td>TRUE = copy materialized view log, FALSE = do nothing</td>
</tr>
</tbody>
</table>

Usage Notes

- The user must check the column `num_errors` before proceeding to ensure that no errors occurred during the cloning of the objects.
- In case of an error, the user should fix the cause of the error and call the `COPY_TABLE_DEPENDENTS Procedure` again to clone the dependent object. Alternatively the user can manually clone the dependent object and then register the manually cloned dependent object using the `REGISTERDEPENDENT_OBJECT Procedure`.
• All cloned referential constraints involving the interim tables will be created disabled (they will be automatically enabled after the redefinition) and all triggers on interim tables will not fire till the redefinition is completed. After the redefinition is complete, the cloned objects will be renamed to the corresponding pre-redefinition names of the objects (from which they were cloned from).

• It is the user’s responsibility that the cloned dependent objects are unaffected by the redefinition. All the triggers will be cloned and it is the user’s responsibility that the cloned triggers are unaffected by the redefinition.

130.7.6 EXECUTE_UPDATE Procedure

This procedure can optimize the performance of bulk updates to a table. Performance is optimized because the updates are not logged in the redo log.

The EXECUTE_UPDATE procedure automatically uses the components of online table redefinition, such an interim table, a materialized view, and a materialized view log, to enable optimized bulk updates to a table. The EXECUTE_UPDATE procedure also removes fragmentation of the affected rows and ensures that the update is atomic. If the bulk updates raise any errors, then you can use the ABORT_UPDATE procedure to undo the changes made by the EXECUTE_UPDATE procedure.

Syntax

```
DBMS_REDEFINITION.EXECUTE_UPDATE (  
  update_stmt  IN  CLOB);
```

Parameters

Table 130-8  EXECUTE_UPDATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update_stmt</td>
<td>The SQL UPDATE statement</td>
</tr>
</tbody>
</table>

See Also:

Oracle Database Administrator's Guide

130.7.7 FINISH_REDEF_TABLE Procedure

This procedure completes the redefinition process.

Before this step, you can create new indexes, triggers, grants, and constraints on the interim table. The referential constraints involving the interim table must be disabled. After completing this step, the original table is redefined with the attributes and data of the interim table. The original table is locked briefly during this procedure.

Syntax

```
DBMS_REDEFINITION.FINISH_REDEF_TABLE (  
  uname                   IN   VARCHAR2,  
  orig_table              IN   VARCHAR2,  
```

### Parameters

**Table 130-9 FINISH_REDEF_TABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the tables</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table. Can take a comma-delimited list of interim table names.</td>
</tr>
<tr>
<td>part_name</td>
<td>Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined. Can take a comma-delimited list of partition names to be redefined.</td>
</tr>
<tr>
<td>dml_lock_timeout</td>
<td>Specifies the number of seconds the procedure waits for its required locks before failing. The permissible range of values for timeout is 0 to 1,000,000. The default is NULL (wait mode).</td>
</tr>
<tr>
<td>continue_after_errors</td>
<td>When redefining multiple partitions allows operation execution to continue on the next partition (applies only to batched partition redefinition).</td>
</tr>
<tr>
<td>disable_rollback</td>
<td>When set to TRUE, disables the rollback option if it was enabled in the START_REDEF_TABLE procedure. Specifying TRUE cleans up the database objects that enable rollback.</td>
</tr>
</tbody>
</table>

**Examples**

Wait up to 600 seconds for required locks on SH.SALES:

```sql
EXECUTE DBMS_REDEFINITION.FINISH_REDEF_TABLE (
    'SH', 'SALES', 'INT_SALES', 600);
```

### 130.7.8 REDEF_TABLE Procedure

This procedure provides a single interface that integrates several redefinition steps including the CAN_REDEF_TABLE Procedure, the START_REDEF_TABLE Procedure, the COPY_TABLE_DEPENDENTS Procedure and the FINISH_REDEF_TABLE Procedure.

This procedure can change data storage properties including tablespaces (for table, partition, subpartition, index, LOB column), compress type (for table, partition, subpartition, index, LOB column) and STORE AS clause for the LOB column.

**Syntax**

```sql
DBMS_REDEFINITION.REDEF_TABLE (
    uname IN VARCHAR2,
    tname IN VARCHAR2,
    table_compression_type IN VARCHAR2 := NULL,
    table_part_tablespace IN VARCHAR2 := NULL,
    index_key_compression_type IN VARCHAR2 := NULL,
    int_table IN VARCHAR2,
    part_name IN VARCHAR2 := NULL,
    dml_lock_timeout IN PLS_INTEGER := NULL,
    continue_after_errors IN BOOLEAN := FALSE,
    disable_rollback IN PLS_INTEGER := FALSE);
```
Parameters

Table 130-10  REDEF_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the table</td>
</tr>
<tr>
<td>tname</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>table_compression_type</td>
<td>Text string of the table compression clause. NULL means there is no change.</td>
</tr>
<tr>
<td>table_part_tablespace</td>
<td>Tablespace name for the entire table or partitions. NULL means there is no change.</td>
</tr>
<tr>
<td>index_key_compression_type</td>
<td>Text string of the compression clause for all indexes on the table. NULL means there is no change.</td>
</tr>
<tr>
<td>index_tablespace</td>
<td>Tablespace name for all indexes on the table. NULL means there is no change.</td>
</tr>
<tr>
<td>lob_compression_type</td>
<td>Text string of the compression clause for all LOBs in the entire table. NULL means there is no change.</td>
</tr>
<tr>
<td>lob_tablespace</td>
<td>Tablespace name for all LOBs in the table. NULL means there is no change.</td>
</tr>
<tr>
<td>lob_store_as</td>
<td>Specifies LOB store as 'SECUREFILE' or 'BASICFILE'. NULL means there is no change.</td>
</tr>
<tr>
<td>refresh_dep_mviews</td>
<td>When set to 'Y', fast refresh of dependent materialized views is performed once at the end of the redefinition operation.</td>
</tr>
<tr>
<td>dml_lock_timeout</td>
<td>Specifies the number of seconds the procedure waits for its required locks before failing. The permissible range of values for timeout is 0 to 1,000,000. The default is NULL (wait mode).</td>
</tr>
</tbody>
</table>

Examples

BEGIN
DBMS_REDEFINITION.REDEF_TABLE(
    uname                        => 'TABOWNER2',
    tname                        => 'EMP2',
    table_compression_type       => 'ROW STORE COMPRESS ADVANCED',
    table_part_tablespace        => 'NEWTBS',
    index_key_compression_type   => 'COMPRESS 1',
    index_tablespace             => 'NEWIDXTBS',
    lob_compression_type         => 'COMPRESS HIGH',
    lob_tablespace               => 'SLOBTBS',
    lob_store_as                 => 'SECUREFILE');
END;
Related Topics

- **CAN_REDEF_TABLE Procedure**
  This procedure determines if a given table can be redefined online. This is the first step of the online redefinition process. If the table is not a candidate for online redefinition, an error message is raised.

- **START_REDEF_TABLE Procedure**
  This procedure starts a table redefinition.

- **COPY_TABLE_DEPENDENTS Procedure**
  This procedure clones the dependent objects of the table being redefined onto the interim table and registers the dependent objects. This procedure does not clone the already registered dependent objects.

- **FINISH_REDEF_TABLE Procedure**
  This procedure completes the redefinition process.

**See Also:**
*Oracle Database Administrator's Guide* regarding "Performing Online Redefinition with the REDEF_TABLE Procedure"

### 130.7.9 REGISTER_DEPENDENT_OBJECT Procedure

This procedure registers a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table.

This can be used to have the same object on each table but with different attributes. For example: for an index, the storage and tablespace attributes could be different but the columns indexed remain the same.

**Syntax**

```sql
DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(
    uname                  IN    VARCHAR2,
    orig_table             IN    VARCHAR2,
    int_table              IN    VARCHAR2,
    dep_type               IN    PLS_INTEGER,
    dep_owner              IN    VARCHAR2,
    dep_orig_name          IN    VARCHAR2,
    dep_int_name           IN    VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the tables</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table</td>
</tr>
</tbody>
</table>
### Table 130-11 (Cont.) REGISTER_DEPENDENT_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dep_type</td>
<td>Type of the dependent object (see Constants and Operational Notes)</td>
</tr>
<tr>
<td>dep_owner</td>
<td>Owner of the dependent object</td>
</tr>
<tr>
<td>dep_orig_name</td>
<td>Name of the original dependent object</td>
</tr>
<tr>
<td>dep_int_name</td>
<td>Name of the interim dependent object</td>
</tr>
</tbody>
</table>

#### Usage Notes
- Attempting to register an already registered object will raise an error.
- Registering a dependent object will automatically remove that object from DBA_REDEFINITION_ERRORS if an entry exists for that object.

### 130.7.10 ROLLBACK Procedure

This procedure rolls back changes to a table after online table redefinition to return the table to its original definition and preserve DML changes made to the table.

#### Syntax

```sql
DBMS_REDEFINITION.ROLLBACK (
    uname                   IN  VARCHAR2,
    orig_table              IN  VARCHAR2,
    int_table               IN  VARCHAR2 := NULL,
    part_name               IN  VARCHAR2 := NULL,
    dml_lock_timeout        IN  PLS_INTEGER := NULL,
    continue_after_errors   IN  BOOLEAN := FALSE);
```

#### Parameters

### Table 130-12 ROLLBACK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the table to be redefined</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table.</td>
</tr>
<tr>
<td>part_name</td>
<td>Name of the partition being redefined.</td>
</tr>
<tr>
<td>dml_lock_timeout</td>
<td>Specifies the number of seconds the procedure waits for its required locks before failing. The permissible range of values for timeout is 0 to 1,000,000. The default is NULL (wait mode).</td>
</tr>
<tr>
<td>continue_after_errors</td>
<td>When rolling back redefinition changes on multiple partitions, allows operation execution to continue on the next partition (applies only to batched partition redefinition).</td>
</tr>
</tbody>
</table>
130.7.11 SET_PARAM Procedure

This procedure sets a new value for a specified parameter used by the redefinition process identified by a redefinition ID.

**Note:**
Currently, the only value that can be changed by this procedure is the value for the refresh_dep_mviews parameter that is specified in the REDEF_TABLE procedure or the START_REDEF_TABLE procedure. You can determine the redefinition ID and check the value of the refresh_dep_mviews parameter for an online table redefinition operation by querying the DBA_REDEFINITION_STATUS view.

**Syntax**

```sql
DBMS_REDEFINITION.SET_PARAM (  
    redefinition_id  IN  VARCHAR2,  
    param_name       IN  VARCHAR2,  
    param_value      IN  VARCHAR2);  
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>redefinition_id</td>
<td>The redefinition ID that identifies the redefinition process</td>
</tr>
<tr>
<td>param_name</td>
<td>The parameter name</td>
</tr>
<tr>
<td>param_value</td>
<td>The new parameter value</td>
</tr>
</tbody>
</table>

**See Also:**

Oracle Database Administrator's Guide

130.7.12 START_REDEF_TABLE Procedure

This procedure starts a table redefinition.

Prior to calling this procedure, you must manually create an empty interim table (in the same schema as the table to be redefined) with the desired attributes of the post-redefinition table, and then call this procedure to initiate the redefinition.

**Syntax**

```sql
DBMS_REDEFINITION.START_REDEF_TABLE (  
    uname                   IN  VARCHAR2,  
    orig_table              IN  VARCHAR2),  
```
Parameters

Table 130-14  START_REDEF_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the tables</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table. Can take a comma-delimited list of interim table names.</td>
</tr>
<tr>
<td>col_mapping</td>
<td>Mapping information from the columns in the original table to the columns in the interim table. (This is similar to the column list on the SELECT clause of a query.) If NULL, all the columns in the original table are selected and have the same name after redefinition.</td>
</tr>
<tr>
<td>options_flag</td>
<td>Indicates the type of redefinition method to use:</td>
</tr>
<tr>
<td></td>
<td>· If dbms_redefinition.cons_use_pk, the redefinition is done using primary keys or pseudo-primary keys (unique keys with all component columns having NOT NULL constraints). The default method of redefinition is using primary keys.</td>
</tr>
<tr>
<td></td>
<td>· If dbms_redefinition.cons_use_rowid, the redefinition is done using rowids.</td>
</tr>
<tr>
<td>orderby_cols</td>
<td>This optional parameter accepts the list of columns (along with the optional keyword(s) ascending/descending) with which to order by the rows during the initial instantiation of the interim table (the order by is only done for the initial instantiation and not for subsequent synchronizations)</td>
</tr>
<tr>
<td>part_name</td>
<td>Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined. Can take a comma-delimited list of partition names to be redefined.</td>
</tr>
<tr>
<td>continue_after_errors</td>
<td>When redefining multiple partitions allows operation execution to continue on the next partition (applies only to batched partition redefinition)</td>
</tr>
<tr>
<td>copy_vpd_opt</td>
<td>Specifies how VPD policies are handled in online redefinition</td>
</tr>
<tr>
<td>refresh_dep_mviews</td>
<td>When set to 'Y', fast refresh of dependent materialized views is performed when the START_REDEF_TABLE procedure is run, each time the SYNC_INTERIM_TABLE procedure is run, and when the FINISH_REDEF_TABLE procedure is run.</td>
</tr>
</tbody>
</table>
Table 130-14  (Cont.) START_REDEF_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_rollback</td>
<td>When set to TRUE, enables the rollback option. When this parameter is set to true, Oracle Database maintains the interim table created during redefinition after redefinition is complete. You can run the SYNC_INTERIM_TABLE procedure to synchronize the interim table periodically to apply DML changes made to the redefined table to the interim table. An internal materialized view and materialized view log enables maintenance of the interim table. If you decide to roll back the online table redefinition with the ROLLBACK procedure, then the interim table is synchronized, and Oracle Database switches back to it so that the table has its original definition.</td>
</tr>
</tbody>
</table>

Examples

Start redefinition of three partitions (sal03q1,sal03q2,sal03q3) in table 'STEVE.salestable' using three interim tables of int_salestable1, int_salestable2 and int_salestable3, respectively. The operation will continue on sal03q3 even if it fails on sal03q1.

```sql
DBMS_REDEFINITION.START_REDEF_TABLE(
  uname                 => 'STEVE',
  orig_table            => 'salestable',
  int_table             => 'int_salestable1, int_salestable2, int_salestable3',
  col_mapping           => NULL,
  options_flag          => DBMS_REDEFINITION.CONS_USE_ROWID,
  part_name             => 'sal03q1,sal03q2,sal03q3',
  continue_after_errors => TRUE);
```

Specify to copy VPD policies automatically:

```sql
EXECUTE DBMS_REDEFINITION.START_REDEF_TABLE (
  uname                  => 'SCOTT',
  orig_table             => 'T',
  int_table              => 'INT_T',
  copy_vpd_opt           => DBMS_REDEFINITION.CONS_VPD_AUTO);
```

130.7.13 SYNC_INTERIM_TABLE Procedure

This procedure keeps the interim table synchronized with the original table.

Syntax

```sql
DBMS_REDEFINITION.SYNC_INTERIM_TABLE (   
  uname           IN VARCHAR2,   
  orig_table      IN VARCHAR2,   
  int_table       IN VARCHAR2,   
  part_name       IN VARCHAR2 := NULL,   
  continue_after_errors IN BOOLEAN := FALSE);
```
Parameters

Table 130-15  SYNC_INTERIM_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the table</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table. Can take a comma-delimited list of interim table names.</td>
</tr>
<tr>
<td>part_name</td>
<td>Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined. Can take a comma-delimited list of partition names to be redefined.</td>
</tr>
<tr>
<td>continue_after_errors</td>
<td>When redefining multiple partitions allows operation execution to continue on the next partition (applies only to batched partition redefinition)</td>
</tr>
</tbody>
</table>

Usage Notes

- This step is useful in minimizing the amount of synchronization needed to be done by the FINISH_REDEF_TABLE Procedure before completing the online redefinition.
- This procedure can be called between long running operations (such as CREATE INDEX) on the interim table to sync it up with the data in the original table and speed up subsequent operations.

130.7.14 UNREGISTER_DEPENDENT_OBJECT Procedure

This procedure unregisters a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table.

Syntax

```sql
DBMS_REDEFINITION.UNREGISTERDEPENDENT_OBJECT(
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    dep_type IN PLS_INTEGER,
    dep_owner IN VARCHAR2,
    dep_orig_name IN VARCHAR2,
    dep_int_name IN VARCHAR2);
```

Parameters

Table 130-16  UNREGISTERDEPENDENT_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
<td>Schema name of the tables</td>
</tr>
<tr>
<td>orig_table</td>
<td>Name of the table to be redefined</td>
</tr>
<tr>
<td>int_table</td>
<td>Name of the interim table</td>
</tr>
</tbody>
</table>
Table 130-16  (Cont.) UNREGISTER_DEPENDENT_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dep_type</td>
<td>Type of the dependent object</td>
</tr>
<tr>
<td>dep_owner</td>
<td>Owner of the dependent object</td>
</tr>
<tr>
<td>dep_orig_name</td>
<td>Name of the original dependent object</td>
</tr>
<tr>
<td>dep_int_name</td>
<td>Name of the interim dependent object</td>
</tr>
</tbody>
</table>
The DBMS_REFRESH package enables you to create groups of materialized views that can be refreshed together to a transactionally consistent point in time. These groups are called refresh groups.

This chapter contains the following topics:

- DBMS_REFRESH Overview
- DBMS_REFRESH Security Model

### 131.1 DBMS_REFRESH Overview

When it is important for materialized views to be transactionally consistent with each other, you can organize them into refresh groups.

By refreshing the refresh group, you can ensure that the data in all of the materialized views in the refresh group correspond to the same transactionally consistent point in time. A materialized view in a refresh group still can be refreshed individually, but doing so nullifies the benefits of the refresh group because refreshing the materialized view individually does not refresh the other materialized views in the refresh group.

### 131.2 DBMS_REFRESH Security Model

Users must have the EXECUTE privilege to run the procedures of DBMS_REFRESH package.

### 131.3 Summary of DBMS_REFRESH Subprograms

This table lists the DBMS_REFRESH subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD Procedure</td>
<td>Adds materialized views to a refresh group</td>
</tr>
<tr>
<td>CHANGE Procedure</td>
<td>Changes the refresh interval for a refresh group</td>
</tr>
<tr>
<td>DESTROY Procedure</td>
<td>Removes all of the materialized views from a refresh group and deletes the refresh group</td>
</tr>
<tr>
<td>MAKE Procedure</td>
<td>Specifies the members of a refresh group and the time interval used to determine when to refresh the members of this group</td>
</tr>
<tr>
<td>REFRESH Procedure</td>
<td>Manually refreshes a refresh group</td>
</tr>
<tr>
<td>SUBTRACT Procedure</td>
<td>Removes materialized views from a refresh group</td>
</tr>
</tbody>
</table>
131.3.1 ADD Procedure

This procedure adds materialized views to a refresh group.

Syntax

```
DBMS_REFRESH.ADD (
    name     IN VARCHAR2,
    { list   IN VARCHAR2,
      | tab    IN DBMS_UTILITY.UNCL_ARRAY, }
    lax      IN BOOLEAN := FALSE);
```

**Note:**

This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

**Table 131-2  ADD Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group to which you want to add members, specified as <code>[schema_name]refresh_group_name</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>list</td>
<td>Comma-delimited list of materialized views that you want to add to the refresh group. Synonyms are not supported. Each materialized view is specified as <code>[schema_name]materialized_view_name</code>. If the schema is not specified, then the refresh group owner is the default.</td>
</tr>
<tr>
<td>tab</td>
<td>Instead of a comma-delimited list, you can supply a PL/SQL associative array of type <code>DBMS_UTILITY.UNCL_ARRAY</code>, where each element is the name of a materialized view. The first materialized view should be in position 1. The last position must be <code>NULL</code>. Each materialized view is specified as <code>[schema_name]materialized_view_name</code>. If the schema is not specified, then the refresh group owner is the default.</td>
</tr>
<tr>
<td>lax</td>
<td>A materialized view can belong to only one refresh group at a time. If you are moving a materialized view from one group to another, then you must set the <code>lax</code> flag to <code>TRUE</code> to succeed. Oracle then automatically removes the materialized view from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to <code>ADD</code> generates an error message.</td>
</tr>
</tbody>
</table>
131.3.2 CHANGE Procedure

This procedure changes the refresh interval for a refresh group.

Syntax

```sql
DBMS_REFRESH.CHANGE (
    name                  IN VARCHAR2,
    next_date             IN DATE           := NULL,
    interval              IN VARCHAR2       := NULL,
    implicit_destroy      IN BOOLEAN        := NULL,
    rollback_seg          IN VARCHAR2       := NULL,
    push_deferred_rpc     IN BOOLEAN        := NULL,
    refresh_after_errors  IN BOOLEAN        := NULL,
    purge_option          IN BINARY_INTEGER := NULL,
    parallelism           IN BINARY_INTEGER := NULL,
    heap_size             IN BINARY_INTEGER := NULL);
```

Parameters

Table 131-3  CHANGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group for which you want to alter the refresh interval.</td>
</tr>
<tr>
<td>next_date</td>
<td>Next date that you want a refresh to occur. By default, this date remains</td>
</tr>
<tr>
<td></td>
<td>unchanged.</td>
</tr>
<tr>
<td>interval</td>
<td>Function used to calculate the next time to refresh the materialized views</td>
</tr>
<tr>
<td></td>
<td>in the refresh group. This interval is evaluated immediately before the refresh. This interval is evaluated immediately before the refresh. Thus, select an interval that is greater than the time it takes to perform a refresh. By default, the interval remains unchanged.</td>
</tr>
<tr>
<td>implicit_destroy</td>
<td>Allows you to reset the value of the implicit_destroy flag. If this flag is set, then Oracle automatically deletes the group if it no longer contains any members. By default, this flag remains unchanged.</td>
</tr>
<tr>
<td>rollback_seg</td>
<td>Allows you to change the rollback segment used. By default, the rollback segment remains unchanged. To reset this parameter to use the default rollback segment, specify NULL, including the quotes. Specifying NULL without quotes indicates that you do not want to change the rollback segment currently being used.</td>
</tr>
<tr>
<td>push_deferred_rpc</td>
<td>Starting with Oracle Database 12c Release 2 (12.2), this parameter is</td>
</tr>
<tr>
<td>refresh_after_errors</td>
<td>Starting with Oracle Database 12c Release 2 (12.2), this parameter is</td>
</tr>
<tr>
<td>purg_e_option</td>
<td>Starting with Oracle Database 12c Release 2 (12.2), this parameter is</td>
</tr>
<tr>
<td>parallelism</td>
<td>• 0 specifies serial propagation.</td>
</tr>
<tr>
<td></td>
<td>• n &gt; 1 specifies parallel propagation with n parallel processes.</td>
</tr>
<tr>
<td></td>
<td>• 1 specifies parallel propagation using only one parallel process.</td>
</tr>
</tbody>
</table>
Table 131-3 (Cont.) CHANGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap_size</td>
<td>Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance.</td>
</tr>
</tbody>
</table>

**Note:**
Do not set this parameter unless directed to do so by Oracle Support Services.

### 131.3.3 DESTROY Procedure

This procedure removes all of the materialized views from a refresh group and delete the refresh group.

**Syntax**

```sql
DBMS_REFRESH.DESTROY (name IN VARCHAR2);
```

**Parameters**

Table 131-4 DESTROY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group that you want to destroy.</td>
</tr>
</tbody>
</table>

### 131.3.4 MAKE Procedure

This procedure specifies the members of a refresh group and the time interval used to determine when to refresh the members of this group.

**Syntax**

```sql
DBMS_REFRESH.MAKE (name IN VARCHAR2, { list IN VARCHAR2, | tab IN DBMS_UTILITY.UNCL_ARRAY, } next_date IN DATE, interval IN VARCHAR2, implicit_destroy IN BOOLEAN := FALSE, lax IN BOOLEAN := FALSE, job IN BINARY_INTEGER := 0, rollback_seg IN VARCHAR2 := NULL, push_deferred_rpc IN BOOLEAN := TRUE, refresh_after_errors IN BOOLEAN := FALSE, purge_option IN BINARY_INTEGER := NULL, parallelism IN BINARY_INTEGER := NULL, heap_size IN BINARY_INTEGER := NULL)
```
Note:

This procedure is overloaded. The list and tab parameters are mutually exclusive.

Parameters

Table 131-5  MAKE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Unique name used to identify the refresh group, specified as [schema_name.].refresh_group_name. If the schema is not specified, then the current user is the default. Refresh groups must follow the same naming conventions as tables. Each materialized view is specified as [schema_name.].materialized_view_name. If the schema is not specified, then the refresh group owner is the default.</td>
</tr>
<tr>
<td>list</td>
<td>Comma-delimited list of materialized views that you want to refresh. Synonyms are not supported. These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your current database. Each materialized view is specified as [schema_name.].materialized_view_name. If the schema is not specified, then the refresh group owner is the default.</td>
</tr>
<tr>
<td>tab</td>
<td>Instead of a comma-delimited list, you can supply a PL/SQL associative array of names of materialized views that you want to refresh using the data type DBMS_UTILITY.UNCL_ARRAY. If the table contains the names of n materialized views, then the first materialized view should be in position 1 and the n + 1 position should be set to NULL. Each materialized view is specified as [schema_name.].materialized_view_name. If the schema is not specified, then the refresh group owner is the default.</td>
</tr>
<tr>
<td>next_date</td>
<td>Next date that you want a refresh to occur.</td>
</tr>
<tr>
<td>interval</td>
<td>Function used to calculate the next time to refresh the materialized views in the group. This field is used with the next_date value. For example, if you specify NEXT_DAY(SYSDATE+1, &quot;MONDAY&quot;) as your interval, and if your next_date evaluates to Monday, then Oracle refreshes the materialized views every Monday. This interval is evaluated immediately before the refresh. Thus, select an interval that is greater than the time it takes to perform a refresh.</td>
</tr>
<tr>
<td>implicit_destroy</td>
<td>Set this to TRUE to delete the refresh group automatically when it no longer contains any members. Oracle checks this flag only when you call the SUBTRACT procedure. That is, setting this flag still enables you to create an empty refresh group.</td>
</tr>
<tr>
<td>lax</td>
<td>A materialized view can belong to only one refresh group at a time. If you are moving a materialized view from an existing group to a new refresh group, then you must set this to TRUE to succeed. Oracle then automatically removes the materialized view from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to MAKE generates an error message.</td>
</tr>
<tr>
<td>job</td>
<td>Needed by the Import utility. Use the default value, 0.</td>
</tr>
</tbody>
</table>
### Table 131-5  (Cont.) MAKE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rollback_seg</td>
<td>Name of the rollback segment to use while refreshing materialized views. The default, NULL, uses the default rollback segment.</td>
</tr>
<tr>
<td>push_deferred_rpc</td>
<td>Starting with Oracle Database 12c Release 2 (12.2), this parameter is ignored.</td>
</tr>
<tr>
<td>refresh_after.Errs</td>
<td>Starting with Oracle Database 12c Release 2 (12.2), this parameter is ignored.</td>
</tr>
<tr>
<td>purge_option</td>
<td>Starting with Oracle Database 12c Release 2 (12.2), this parameter is ignored.</td>
</tr>
<tr>
<td>parallelism</td>
<td>• 0 specifies serial propagation.</td>
</tr>
<tr>
<td></td>
<td>• $n &gt; 1$ specifies parallel propagation with $n$ parallel processes.</td>
</tr>
<tr>
<td></td>
<td>• 1 specifies parallel propagation using only one parallel process.</td>
</tr>
<tr>
<td>heap_size</td>
<td>Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance.</td>
</tr>
</tbody>
</table>

**Note:**
Do not set this parameter unless directed to do so by Oracle Support Services.

| job_name          | This parameter is needed by the import utility. User should use the default value, NULL.                                                   |
| auto_commit       | Supported values are NULL, TRUE, and FALSE.                                                                                               |
|                   | • NULL—allows the user to continue using DBMS_JOB.                                                                                         |
|                   | • TRUE—commit statement will be automatically issued after the job of the refresh group are created by DBMS_REFRESH.MAKE.                  |
|                   | • FALSE—user must issue a commit statement to finish the transaction after calling DBMS_REFRESH.MAKE.                                    |
|                   | The default value is NULL.                                                                                                                |

**Usage Notes**

Import utility and export utility need CREATE JOB privilege if DBMS_SCHEDULER jobs are used.

### 131.3.5 REFRESH Procedure

This procedure manually refreshes a refresh group.

**Syntax**

```sql
DBMS_REFRESH.REFRESH (name IN VARCHAR2);
```
131.3.6 SUBTRACT Procedure

This procedure removes materialized views from a refresh group.

Syntax

```sql
DBMS_REFRESH.SUBTRACT (
    name IN VARCHAR2,
    { list IN VARCHAR2,
      | tab IN DBMS_UTILITY.UNCL_ARRAY, }
    lax IN BOOLEAN := FALSE);
```

**Note:**

This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

**Table 131-6** REFRESH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group that you want to refresh manually.</td>
</tr>
</tbody>
</table>

**Table 131-7** SUBTRACT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the refresh group from which you want to remove members, specified as <code>[schema_name].refresh_group_name</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
| list      | Comma-delimited list of materialized views that you want to remove from the refresh group. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your current database.
  Each materialized view is specified as `[schema_name].materialized_view_name`. If the schema is not specified, then the refresh group owner is the default. |
| tab       | Instead of a comma-delimited list, you can supply a PL/SQL associative array of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a materialized view. The first materialized view should be in position 1. The last position must be NULL.
  Each materialized view is specified as `[schema_name].materialized_view_name`. If the schema is not specified, then the refresh group owner is the default. |
| lax        | Set this to `FALSE` if you want Oracle to generate an error message if the materialized view you are attempting to remove is not a member of the refresh group. |
The `DBMS_REPAIR` package contains data corruption repair procedures that enable you to detect and repair corrupt blocks in tables and indexes. You can address corruptions where possible and continue to use objects while you attempt to rebuild or repair them.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Operating Notes
- Exceptions
- Examples
- Summary of `DBMS_REPAIR` Subprograms

See Also:

For detailed information about using the `DBMS_REPAIR` package, see *Oracle Database Administrator's Guide*.

132.1 DBMS_REPAIR Overview

The `DBMS_REPAIR` package is intended for use by database administrators only. It is not intended for use by application developers.

132.2 DBMS_REPAIR Security Model

The package is owned by `SYS`. Execution privilege is not granted to other users.

132.3 DBMS_REPAIR Constants

The `DBMS_REPAIR` package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, `DBMS_REPAIR.TABLE_OBJECT`.

The following table lists the parameters and the enumerated constants.
Table 132-1   DBMS_REPAIR Parameters with Enumerated Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>· TABLE_OBJECT</td>
<td>BINARY_INTEG</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>· INDEX_OBJECT</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· CLUSTER_OBJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>action</td>
<td>· CREATE_ACTION</td>
<td>BINARY_INTEG</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>· DROP_ACTION</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· PURGE_ACTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>table_type</td>
<td>· REPAIR_TABLE</td>
<td>BINARY_INTEG</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>· ORPHAN_TABLE</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>flags</td>
<td>· SKIP_FLAG</td>
<td>BINARY_INTEG</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>· NOSKIP_FLAG</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>object_id</td>
<td>· ALL_INDEX_ID := 0</td>
<td>BINARY_INTEG</td>
<td>Clean up all objects that qualify</td>
</tr>
<tr>
<td>wait_for_lock</td>
<td>· LOCK_WAIT := 1</td>
<td>BINARY_INTEG</td>
<td>Specifies whether to try getting DML locks on</td>
</tr>
<tr>
<td></td>
<td>· LOCK_NOWAIT := 0</td>
<td>R</td>
<td>underlying table [sub]partition object</td>
</tr>
</tbody>
</table>

**Note:**
The default table name will be REPAIR_TABLE when table_type is REPAIR_TABLE, and will be ORPHAN_KEY_TABLE when table_type is ORPHAN_TABLE.

132.4 DBMS_REPAIR Operating Notes

The procedure to create the ORPHAN_KEY_TABLE is similar to the one used to create the REPAIR_TABLE.

CONNECT / AS SYSDBA;
EXEC DBMS_REPAIR.ADMIN_TABLES('ORPHAN_KEYS_TABLE', DBMS_REPAIR.ORPHAN_TABLE, DBMS_REPAIR.CREATE_ACTION);
EXEC DBMS_REPAIR.ADMIN_TABLES('REPAIR_TABLE', DBMS_REPAIR.REPAIR_TABLE, DBMS_REPAIR.CREATE_ACTION);
DESCRIBE ORPHAN_KEYS_TABLE;
DESCRIBE REPAIR_TABLE;
SELECT * FROM ORPHAN_KEYS_TABLE;
SELECT * FROM REPAIR_TABLE;

The DBA would create the repair and orphan keys tables once. Subsequent executions of the CHECK_OBJECT Procedure would add rows into the appropriate table indicating the types of errors found.

The name of the repair and orphan keys tables can be chosen by the user, with the following restriction: the name of the repair table must begin with the 'REPAIR_' prefix, and the name of the orphan keys table must begin with the 'ORPHAN_' prefix. The following code is also legal:
When invoking the `CHECK_OBJECT` Procedure the name of the repair and orphan keys tables that were created should be specified correctly, especially if the default values were not used in the `ADMIN_TABLES` Procedure or `CREATE_ACTION`.

Other actions in the `ADMIN_TABLES` Procedure can be used to purge/delete the `REPAIR_TABLE` and the `ORPHAN_KEYS_TABLE`.

### 132.5 DBMS_REPAIR Exceptions

The table in this topic describes the exceptions raised by the DDBMS_REPAIR subprograms.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>942</td>
<td>Reported by <code>DBMS_REPAIR.ADMIN_TABLES</code> during a <code>DROP_ACTION</code> when the specified table doesn't exist.</td>
<td>-</td>
</tr>
<tr>
<td>955</td>
<td>Reported by <code>DBMS_REPAIR.CREATE_ACTION</code> when the specified table already exists.</td>
<td>-</td>
</tr>
<tr>
<td>24120</td>
<td>An invalid parameter was passed to the specified <code>DBMS_REPAIR</code> procedure.</td>
<td>Specify a valid parameter value or use the parameter's default.</td>
</tr>
<tr>
<td>24122</td>
<td>An incorrect block range was specified.</td>
<td>Specify correct values for the <code>BLOCK_START</code> and <code>BLOCK_END</code> parameters.</td>
</tr>
<tr>
<td>24123</td>
<td>An attempt was made to use the specified feature, but the feature is not yet implemented.</td>
<td>Do not attempt to use the feature.</td>
</tr>
<tr>
<td>24124</td>
<td>An invalid <code>ACTION</code> parameter was specified.</td>
<td>Specify <code>CREATE_ACTION</code>, <code>PURGE_ACTION</code> or <code>DROP_ACTION</code> for the <code>ACTION</code> parameter.</td>
</tr>
<tr>
<td>24125</td>
<td>An attempt was made to fix corrupt blocks on an object that has been dropped or truncated since <code>DBMS_REPAIR.CHECK_OBJECT</code> was run.</td>
<td>Use <code>DBMS_REPAIR.ADMIN_TABLES</code> to purge the repair table and run <code>DBMS_REPAIR.CHECK_OBJECT</code> to determine whether there are any corrupt blocks to be fixed.</td>
</tr>
<tr>
<td>24127</td>
<td><code>TABLESPACE</code> parameter specified with an <code>ACTION</code> other than <code>CREATE_ACTION</code>.</td>
<td>Do not specify <code>TABLESPACE</code> when performing actions other than <code>CREATE_ACTION</code>.</td>
</tr>
<tr>
<td>24128</td>
<td>A partition name was specified for an object that is not partitioned.</td>
<td>Specify a partition name only if the object is partitioned.</td>
</tr>
</tbody>
</table>
Table 132-2  (Cont.) DBMS_REPAIR Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>24129</td>
<td>An attempt was made to pass a table name parameter without the specified prefix.</td>
<td>Pass a valid table name parameter.</td>
</tr>
<tr>
<td>24130</td>
<td>An attempt was made to specify a repair or orphan table that does not exist.</td>
<td>Specify a valid table name parameter.</td>
</tr>
<tr>
<td>24131</td>
<td>An attempt was made to specify a repair or orphan table that does not have a correct definition.</td>
<td>Specify a table name that refers to a properly created table.</td>
</tr>
<tr>
<td>24132</td>
<td>An attempt was made to specify a table name is greater than 30 characters long.</td>
<td>Specify a valid table name parameter.</td>
</tr>
</tbody>
</table>

132.6 DBMS_REPAIR Examples

This topic shows examples of DBMS_REPAIR usage.

/* Fix the bitmap status for all the blocks in table mytab in schema sys */
EXECUTE DBMS_REPAIR.SEGMENT_FIX_STATUS('SYS', 'MYTAB');

/* Mark block number 45, filenumber 1 for table mytab in sys schema as FULL.*/
EXECUTE DBMS_REPAIR.SEGMENT_FIX_STATUS('SYS', 'MYTAB', TABLE_OBJECT,1, 45, 1);

132.7 Summary of DBMS_REPAIR Subprograms

This table lists the DBMS_REPAIR subprograms and briefly describes them.

Table 132-3  DBMS_REPAIR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMIN_TABLES Procedure</td>
<td>Provides administrative functions for the DBMS_REPAIR package repair and orphan key tables, including create, purge, and drop functions</td>
</tr>
<tr>
<td>CHECK_OBJECT Procedure</td>
<td>Detects and reports corruptions in a table or index</td>
</tr>
<tr>
<td>DUMP_ORPHAN_KEYS Procedure</td>
<td>Reports on index entries that point to rows in corrupt data blocks</td>
</tr>
<tr>
<td>FIX_CORRUPT_BLOCKS Procedure</td>
<td>Marks blocks software corrupt that have been previously detected as corrupt by CHECK_OBJECT</td>
</tr>
<tr>
<td>ONLINE_INDEX_CLEAN Function</td>
<td>Performs a manual cleanup of failed or interrupted online index builds or rebuilds</td>
</tr>
<tr>
<td>REBUILD_FREELISTS Procedure</td>
<td>Rebuilds an object’s freelists</td>
</tr>
<tr>
<td>SEGMENT_FIX_STATUS Procedure</td>
<td>Fixes the corrupted state of a bitmap entry</td>
</tr>
</tbody>
</table>
Table 132-3  (Cont.) DBMS_REPAIR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKIP_CORRUPT_BLOCKS</td>
<td>Sets whether to ignore blocks marked corrupt during table and index scans or to report ORA-1578 when blocks marked corrupt are encountered</td>
</tr>
</tbody>
</table>

132.7.1 ADMIN_TABLES Procedure

This procedure provides administrative functions for the DBMS_REPAIR package repair and orphan key tables.

Syntax

`DBMS_REPAIR.ADMIN_TABLES (``
  table_name  IN   VARCHAR2,
  table_type  IN   BINARY_INTEGER,
  action      IN   BINARY_INTEGER,
  tablespace  IN   VARCHAR2   DEFAULT NULL);`

Parameters

Table 132-4  ADMIN_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table to be processed. Defaults to ORPHAN_KEY_TABLE or REPAIR_TABLE based on the specified table_type. When specified, the table name must have the appropriate prefix: ORPHAN_ or REPAIR_.</td>
</tr>
<tr>
<td>table_type</td>
<td>Type of table; must be either ORPHAN_TABLE or REPAIR_TABLE. See &quot;Constants&quot;.</td>
</tr>
<tr>
<td>action</td>
<td>Indicates what administrative action to perform. Must be either CREATE_ACTION, PURGE_ACTION, or DROP_ACTION. If the table already exists, and if CREATE_ACTION is specified, then an error is returned. PURGE_ACTION indicates to delete all rows in the table that are associated with non-existent objects. If the table does not exist, and if DROP_ACTION is specified, then an error is returned. When CREATE_ACTION and DROP_ACTION are specified, an associated view named DBA &lt;table_name&gt; is created and dropped respectively. The view is defined so that rows associated with non-existent objects are eliminated. Created in the SYS schema. See &quot;Constants&quot;.</td>
</tr>
<tr>
<td>tablespace</td>
<td>Indicates the tablespace to use creating a table. By default, the SYS default tablespace is used. An error is returned if the tablespace is specified and if the action is not CREATE_ACTION.</td>
</tr>
</tbody>
</table>
132.7.2 CHECK_OBJECT Procedure

This procedure checks the specified objects and populates the repair table with information about corruptions and repair directives.

Validation consists of block checking all blocks in the object.

Syntax

```sql
DBMS_REPAIR.CHECK_OBJECT(
    schema_name       IN  VARCHAR2,
    object_name       IN  VARCHAR2,
    partition_name    IN  VARCHAR2       DEFAULT NULL,
    object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
    repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
    flags             IN  BINARY_INTEGER DEFAULT NULL,
    relative_fno      IN  BINARY_INTEGER DEFAULT NULL,
    block_start       IN  BINARY_INTEGER DEFAULT NULL,
    block_end         IN  BINARY_INTEGER DEFAULT NULL,
    corrupt_count     OUT BINARY_INTEGER);
```

Parameters

Table 132-5  CHECK_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name of the object to be checked.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table or index to be checked.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name to be checked.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. Must be either TABLE_OBJECT (default) or INDEX_OBJECT.</td>
</tr>
<tr>
<td>repair_table_name</td>
<td>Name of the repair table to be populated.</td>
</tr>
<tr>
<td>flags</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Relative file number: Used when specifying a block range.</td>
</tr>
<tr>
<td>block_start</td>
<td>First block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition.</td>
</tr>
<tr>
<td>block_end</td>
<td>Last block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition. If only one of block_start or block_end is specified, then the other defaults to the first or last block in the file respectively.</td>
</tr>
<tr>
<td>corrupt_count</td>
<td>Number of corruptions reported.</td>
</tr>
</tbody>
</table>
Usage Notes
You may optionally specify a DBA range, partition name, or subpartition name when you want to check a portion of an object.

132.7.3 DUMP_ORPHAN_KEYS Procedure

This procedure reports on index entries that point to rows in corrupt data blocks. For each such index entry encountered, a row is inserted into the specified orphan table.

If the repair table is specified, then any corrupt blocks associated with the base table are handled in addition to all data blocks that are marked software corrupt. Otherwise, only blocks that are marked corrupt are handled.

This information may be useful for rebuilding lost rows in the table and for diagnostic purposes.

Syntax

```
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
    schema_name       IN  VARCHAR2,
    object_name       IN  VARCHAR2,
    partition_name    IN  VARCHAR2       DEFAULT NULL,
    object_type       IN  BINARY_INTEGER DEFAULT INDEX_OBJECT,
    repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
    orphan_table_name IN  VARCHAR2       DEFAULT 'ORPHAN_KEYS_TABLE',
    flags             IN  BINARY_INTEGER DEFAULT NULL,
    key_count         OUT BINARY_INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name.</td>
</tr>
<tr>
<td>object_name</td>
<td>Object name.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name to be processed.</td>
</tr>
<tr>
<td></td>
<td>If this is a partitioned object, and if partition_name is not specified,</td>
</tr>
<tr>
<td></td>
<td>then all partitions and subpartitions are processed. If this is a partitioned</td>
</tr>
<tr>
<td></td>
<td>object, and if the specified partition contains subpartitions, then all</td>
</tr>
<tr>
<td></td>
<td>subpartitions are processed.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. The default is INDEX_OBJECT. See &quot;Constants&quot;.</td>
</tr>
<tr>
<td>repair_table_name</td>
<td>Name of the repair table that has information regarding corrupt blocks in</td>
</tr>
<tr>
<td></td>
<td>the base table. The specified table must exist in the SYS schema. The</td>
</tr>
<tr>
<td></td>
<td>ADMIN_TABLES Procedure is used to create the table.</td>
</tr>
<tr>
<td>orphan_table_name</td>
<td>Name of the orphan key table to populate with information regarding each</td>
</tr>
<tr>
<td></td>
<td>index entry that refers to a row in a corrupt data block. The specified</td>
</tr>
<tr>
<td></td>
<td>table must exist in the SYS schema. The ADMIN_TABLES Procedure is used to</td>
</tr>
<tr>
<td></td>
<td>create the table.</td>
</tr>
<tr>
<td>flags</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

Chapter 132
Summary of DBMS_REPAIR Subprograms
132.7.4 FIX_CORRUPT_BLOCKS Procedure

This procedure fixes the corrupt blocks in specified objects based on information in the repair table that was previously generated by the CHECK_OBJECT Procedure.

Prior to effecting any change to a block, the block is checked to ensure the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is effected, the associated row in the repair table is updated with a fix timestamp.

Syntax

```
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
    schema_name       IN  VARCHAR2,
    object_name       IN  VARCHAR2,
    partition_name    IN  VARCHAR2       DEFAULT NULL,
    object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
    repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
    flags             IN  BINARY_INTEGER DEFAULT NULL,
    fix_count         OUT BINARY_INTEGER);
```

Parameters

Table 132-7 FIX_CORRUPT_BLOCKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object with corrupt blocks to be fixed.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name to be processed.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See &quot;Constants&quot;.</td>
</tr>
<tr>
<td>repair_table_name</td>
<td>Name of the repair table with the repair directives.</td>
</tr>
<tr>
<td>flags</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>fix_count</td>
<td>Number of blocks fixed.</td>
</tr>
</tbody>
</table>

Related Topics

- CHECK_OBJECT Procedure
  This procedure checks the specified objects and populates the repair table with information about corruptions and repair directives.
132.7.5 ONLINE_INDEX_CLEAN Function

This function performs a manual cleanup of failed or interrupted online index builds or rebuilds.

This action is also performed periodically by SMON, regardless of user-initiated cleanup.

This function returns TRUE if all indexes specified were cleaned up and FALSE if one or more indexes could not be cleaned up.

Syntax

DBMS_REPAIR.ONLINE_INDEX_CLEAN (
   object_id      IN BINARY_INTEGER DEFAULT ALL_INDEX_ID,
   wait_for_lock  IN BINARY_INTEGER DEFAULT LOCK_WAIT
) RETURN BOOLEAN;

Parameters

Table 132-8 ONLINE_INDEX_CLEAN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>Object id of index to be cleaned up. The default cleans up all object ids that qualify.</td>
</tr>
<tr>
<td>wait_for_lock</td>
<td>This parameter specifies whether to try getting DML locks on underlying table [sub]partition object. The default retries up to an internal retry limit, after which the lock get will give up. If LOCK_NOWAIT is specified, then the lock get does not retry.</td>
</tr>
</tbody>
</table>

132.7.6 REBUILD_FREELISTS Procedure

This procedure rebuilds the freelists for the specified object.

All free blocks are placed on the master freelist. All other freelists are zeroed.

If the object has multiple freelist groups, then the free blocks are distributed among all freelists, allocating to the different groups in round-robin fashion.

Syntax

DBMS_REPAIR.REBUILD_FREELISTS (
   schema_name    IN VARCHAR2,
   object_name    IN VARCHAR2,
   partition_name IN VARCHAR2 DEFAULT NULL,
   object_type    IN BINARY_INTEGER DEFAULT TABLE_OBJECT);

Parameters

Table 132-9 REBUILD_FREELISTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object whose freelists are to be rebuilt.</td>
</tr>
</tbody>
</table>
Table 132-9  (Cont.) REBUILD_FREELISTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition_name</td>
<td>Partition or subpartition name whose freelists are to be rebuilt. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See “Constants”.</td>
</tr>
</tbody>
</table>

132.7.7 SEGMENT_FIX_STATUS Procedure

With this procedure you can fix the corrupted state of a bitmap entry. The procedure either recalculates the state based on the current contents of the corresponding block or sets the state to a specific value.

Syntax

```sql
DBMS_REPAIR.SEGMENT_FIX_STATUS (
    segment_owner   IN VARCHAR2,
    segment_name    IN VARCHAR2,
    segment_type    IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
    file_number     IN BINARY_INTEGER DEFAULT NULL,
    block_number    IN BINARY_INTEGER DEFAULT NULL,
    status_value    IN BINARY_INTEGER DEFAULT NULL,
    partition_name  IN VARCHAR2 DEFAULT NULL,);
```

Parameters

Table 132-10  SEGMENT_FIX_STATUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_owner</td>
<td>Schema name of the segment.</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Optional. Name of an individual partition. NULL for nonpartitioned objects. Default is NULL.</td>
</tr>
<tr>
<td>segment_type</td>
<td>Optional Type of the segment (for example, TABLE_OBJECT or INDEX_OBJECT). Default is NULL.</td>
</tr>
<tr>
<td>file_number</td>
<td>(optional) The tablespace-relative file number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.</td>
</tr>
<tr>
<td>block_number</td>
<td>(optional) The file-relative block number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.</td>
</tr>
</tbody>
</table>
### SEGMENT_FIX_STATUS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status_value</td>
<td>(optional) The value to which the block status described by the file_number and block_number will be set. If omitted, the status will be set based on the current state of the block. This is almost always the case, but if there is a bug in the calculation algorithm, the value can be set manually. Status values: 1 = block is full, 2 = block is 0-25% free, 3 = block is 25-50% free, 4 = block is 50-75% free, 5 = block is 75-100% free. The status for bitmap blocks, segment headers, and extent map blocks cannot be altered. The status for blocks in a fixed hash area cannot be altered. For index blocks, there are only two possible states: 1 = block is full and 3 = block has free space.</td>
</tr>
</tbody>
</table>

### SKIP_CORRUPT_BLOCKS Procedure

This procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object.

When the object is a table, skip applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

**Note:**

When Oracle performs an index range scan on a corrupt index after DBMS_REPAIR.SKIP_CORRUPT_BLOCKS has been set for the base table, corrupt branch blocks and root blocks are not skipped. Only corrupt non-root leaf blocks are skipped.

#### Syntax

```sql
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (    schema_name IN VARCHAR2,    object_name IN VARCHAR2,    object_type IN BINARY_INTEGER DEFAULT TABLE_OBJECT,    flags IN BINARY_INTEGER DEFAULT SKIP_FLAG);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name of the object to be processed.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object to be processed. This must be either TABLE_OBJECT (default) or CLUSTER_OBJECT.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Constants&quot;.</td>
</tr>
<tr>
<td>flags</td>
<td>If SKIP_FLAG is specified, then it turns on the skip of software corrupt blocks for the object during index and table scans. If NOSKIP_FLAG is specified, then scans that encounter software corrupt blocks return an ORA-1578.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Constants&quot;.</td>
</tr>
</tbody>
</table>
The **DBMS_RESCONFIG** package provides an interface to operate on the resource configuration list, and to retrieve listener information for a resource.

This chapter contains the following topics:

- **Overview**
- **Summary of DBMS_RESCONFIG Subprograms**

![See Also:](Oracle_XML_DB_Developer's_Guide_for_more_information_about_'Resource_Configuration'.)
Table 133-1  (Cont.) DBMS_RESCONFIG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETREPOSITORYRESCONFIGPATHS</td>
<td>Returns a list of resource configuration paths defined for the repository</td>
</tr>
<tr>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>GETRESCONFIG Function</td>
<td>Returns the resource configuration at the specified position of the target resource©s configuration list</td>
</tr>
<tr>
<td>GETRESCONFIGPATHS Function</td>
<td>Returns a list of resource configuration paths defined in the target resource©s configuration list</td>
</tr>
<tr>
<td>PATCHREPOSITORYRESCONFIGLIST</td>
<td>Removes invalid references from the repository resource configuration list, and makes the repository available</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
</tbody>
</table>

133.2.1 ADDREPOSITORYRESCONFIG Procedure

This procedure inserts the resource configuration specified by absolute path of the resource configuration at the specified position of the repository's configuration list. It shifts the element currently at that position (if any) and any subsequent elements to the right.

Syntax

DBMS_RESCONFIG.ADDREPOSITORYRESCONFIG(
  rcpath     IN   VARCHAR2,
  pos        IN   PLS_INTEGER := NULL);

Parameters

Table 133-2  ADDREPOSITORYRESCONFIG Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcpath</td>
<td>Absolute path of the resource configuration to be inserted. An exception is raised if rcpath already exists in the target's configuration list.</td>
</tr>
<tr>
<td>pos</td>
<td>Index at which the new configuration is to be inserted. If this parameter is not specified then the new configuration is appended to the end of the list. An exception is raised if the index is out of range (pos &lt; 0 or pos &gt; the size of the target resource's configuration list).</td>
</tr>
</tbody>
</table>

Usage Notes

- An error is raised if the document referenced by rcpath is not based on XDBResConfig.xsd schema.
- Users must have XDBADMIN role and READ privilege on the resource configuration to be inserted; otherwise, an error is returned.

133.2.2 ADDRESCONFIG Procedure

This procedure inserts the resource configuration specified by the absolute path of the resource configuration at the given position in the target resource’s configuration list. It
shifts the element currently at that position (if any) and any subsequent elements to the right.

**Syntax**

```sql
DBMS_RESCONFIG.ADDRESCONFIG(
    respath    IN   VARCHAR2,
    rcpath     IN   VARCHAR2,
    pos        IN   PLS_INTEGER := NULL);
```

**Parameters**

**Table 133-3  ADDRESCONFIG Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>respath</td>
<td>Absolute path of the target resource</td>
</tr>
<tr>
<td>rcpath</td>
<td>Absolute path of the resource configuration to be inserted. An exception is raised if rcpath already exists in the target's configuration list.</td>
</tr>
<tr>
<td>pos</td>
<td>Index at which the new configuration is to be inserted. If this parameter is not specified then the new configuration is appended to the end of the list. An exception is raised if the index is out of range (pos &lt; 0 or pos &gt; the size of the target resource's configuration list).</td>
</tr>
</tbody>
</table>

**Usage Notes**

- An error is raised if the document referenced by `rcpath` is not based on `XDBResConfig.xsd` schema.
- Users must have `WRITE-CONFIG` privilege on the target resource and read privilege on the resource configuration to be inserted; otherwise, an error is returned.

### 133.2.3 APPENDRESCONFIG Procedure

This procedure appends the resource configuration specified by `rcpath` to the target resource's configuration list if it is not already included in the list.

**Syntax**

```sql
DBMS_RESCONFIG.ADDRESCONFIG(
    respath       IN   VARCHAR2,
    rcpath        IN   VARCHAR2,
    appendOption  IN   PLS_INTEGER);
```

**Parameters**

**Table 133-4  ADDRESCONFIG Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>respath</td>
<td>Absolute path of the target resource</td>
</tr>
<tr>
<td>rcpath</td>
<td>Absolute path of the resource configuration to be appended at the end of the target's configuration list. If <code>rcpath</code> already exists in the list then nothing is appended.</td>
</tr>
</tbody>
</table>
Table 133-4  (Cont.) ADDRESCONFIG Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendOption</td>
<td>Either APPEND_RESOURCE or APPEND_RECURSIVE. If APPEND_RESOURCE is specified then only the target resource is affected. If APPEND_RECURSIVE is specified then the target resource and all its descendents will be affected.</td>
</tr>
</tbody>
</table>

Usage Notes

• An error is raised if the document referenced by rcpath is not based on XDBResConfig.xsd schema.

• Users must have WRITE-CONFIG privilege on all affected resources and required read privilege on the resource configuration to be inserted; otherwise, an error is returned.

133.2.4 DELETEREPOSITORYRESCONFIG Procedure

This procedure removes the configuration at the given position in the repository's configuration list. It shifts any subsequent elements to the left.

Syntax

DBMS_RESCONFIG.DELETEREPOSITORYRESCONFIG(oui PLS_INTEGER);

Parameters

Table 133-5  DELETEREPOSITORYRESCONFIG Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The index of the configuration to be removed. An exception is raised if the index is out of range (pos &lt; 0 or pos &gt;= the size of the target resource's configuration list).</td>
</tr>
</tbody>
</table>

Usage Notes

• Users must have XDBADMIN role to execute this.

• This statement is treated as if it is a DDL statement. This means the system will implicitly commit before and after this statement.
133.2.5 DELETERESCONFIG Procedures

This procedure removes the configuration at the given position in the target resource's configuration list. It shifts any subsequent elements to the left. Users can use the overloaded for recursive deletion.

Syntax

DBMS_RESCONFIG.DELETERESCONFIG(
  respath       IN   VARCHAR2,
  pos           IN   PLS_INTEGER);

DBMS_RESCONFIG.DELETERESCONFIG(
  respath        IN  VARCHAR2,
  rcpath         IN  VARCHAR2,
  deleteOption   IN  PLS_INTEGER);

Parameters

Table 133-6  DELETERESCONFIG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>respath</td>
<td>Absolute path of the target resource</td>
</tr>
<tr>
<td>pos</td>
<td>The index of the configuration to be removed. An exception is raised if the index is out of range (pos &lt; 0 or pos &gt;= the size of the target resource's configuration list).</td>
</tr>
<tr>
<td>rcpath</td>
<td>Absolute path of the resource configuration to be deleted if found in list.</td>
</tr>
<tr>
<td>deleteOption</td>
<td>Either DELETERESOURCE or DELETERECURSIVE. If DELETERESOURCE is specified then only the configuration list of the target resource is affected. If DELETERECURSIVE is specified then the configuration list of the target resource and all its descendents will be affected.</td>
</tr>
</tbody>
</table>

Usage Notes

Users must have WRITE-CONFIG privilege on the target resource to execute this.

133.2.6 GETLISTENERS Function

This function returns the list of listeners applicable for a given resource.

The value returned by this function is an XML document containing the <event-listeners> element of the XDBResconfig.xsd schema. It contains all the listeners applicable to the target resource, including repository-level listeners. From the returned XML document users can use the EXTRACT operator to retrieve the listeners defined for a specific event.

Syntax

DBMS_RESCONFIG.GETLISTENERS(
  path IN VARCHAR2)
RETURN XMLTYPE;
Parameters

Table 133-7   GETLISTENERS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Absolute path of the target resource</td>
</tr>
</tbody>
</table>

Usage Notes

Users must have the required access privilege on all resource configurations referenced by the repository and the target resource; otherwise, an error is returned.

133.2.7 GETREPOSITORYRESCONFIG Function

This function returns the resource configuration at the specified position of the repository's configuration list.

Syntax

```
DBMS_RESCONFIG.GETREPOSITORYRESCONFIG(
   pos    IN   PLS_INTEGER)
RETURN XMLTYPE;
```

Parameters

Table 133-8   GETREPOSITORYRESCONFIG Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>Index of element to return. An exception is raised if the index is out of range (pos &lt; 0 or pos &gt;= the size of the repository's configuration list).</td>
</tr>
</tbody>
</table>

Usage Notes

Users must have the required read privilege on the requested resource configuration; otherwise, an error is returned.

133.2.8 GETREPOSITORYRESCONFIGPATHS Function

This function returns a list of resource configuration paths defined for the repository.

Syntax

```
DBMS_RESCONFIG.GETREPOSITORYRESCONFIGPATHS
RETURN XDB#STRING_LIST_T;
```

Usage Notes

Users must be able to access all the referenced resource configurations; otherwise, an error is returned.
133.2.9 GETRESCONFIG Function

This function returns the resource configuration at the specified position of the target resource's configuration list.

**Syntax**

```sql
DBMS_RESCONFIG.GETRESCONFIG(
    respath IN VARCHAR2,
    pos IN PLS_INTEGER
) RETURN XMLTYPE;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>respath</td>
<td>Absolute path of the target resource</td>
</tr>
<tr>
<td>pos</td>
<td>Index of element to return. An exception is raised if the index is out of range (pos &lt; 0 or pos &gt;= the size of the target resource's configuration list).</td>
</tr>
</tbody>
</table>

**Usage Notes**

Users must have the required read privilege on the requested resource configuration; otherwise, an error is returned.

133.2.10 GETRESCONFIGPATHS Function

This function returns a list of resource configuration paths defined in the target resource's configuration list.

**Syntax**

```sql
DBMS_RESCONFIG.GETRESCONFIGPATHS(
    respath IN VARCHAR2
) RETURN XDB$STRING_LIST_T;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>respath</td>
<td>Absolute path of the target resource</td>
</tr>
</tbody>
</table>

**Usage Notes**

Users must be able to access all the referenced resource configurations; otherwise, an error is returned.
133.2.11 PATCHREPOSITORYRESCONFIGLIST Procedure

This procedure removes invalid references from the repository resource configuration list, and makes the repository available.

Under normal circumstances, deletion of a resource configuration resource cannot be performed if it is part of the repository resource configuration list. If, for some reason, the deletion of a resource configuration resource that is part of the repository resource configuration list succeeds, then any repository operation results in a ‘dangling reference’ error. This procedure removes those invalid references.

This procedure must be run as SYS.

Syntax

```
DBMS_RESCONFIG.PATCHREPOSITORYRESCONFIGLIST;
```
The `DBMS_RESOURCE_MANAGER` package maintains plans, consumer groups, and plan directives. It also provides semantics so that you may group together changes to the plan schema.

This chapter contains the following topics:

- Deprecated Subprograms
- Security Model
- Constants
- Summary of `DBMS_RESOURCE_MANAGER` Subprograms

See Also:

For more information on using the Database Resource Manager, see Oracle Database Administrator’s Guide

134.1 DBMS_RESOURCE_MANAGER Deprecated Subprograms

The `SET_INITIAL_CONSUMER_GROUP` Procedure has been deprecated with Oracle Database 11g.

- `SET_INITIAL_CONSUMER_GROUP` Procedure

Note:

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

134.2 DBMS_RESOURCE_MANAGER Security Model

The invoker must have the `ADMINISTER_RESOURCE_MANAGER` system privilege to execute these procedures.

The procedures to grant and revoke this privilege are in the package `DBMS_RESOURCE_MANAGER_PRIVS`. 
134.3 DBMS_RESOURCE_MANAGER Constants

The DBMS_RESOURCE_MANAGER package defines several constants for specifying parameter values.

These are shown in the following table.

Table 134-1 DBMS_RESOURCE_MANAGER Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_ID</td>
<td>VARCHAR2(30)</td>
<td>CLIENT_ID</td>
<td>Client identifier of the session</td>
</tr>
<tr>
<td>CLIENT_MACHINE</td>
<td>VARCHAR2(30)</td>
<td>CLIENT_MACHIN</td>
<td>Name of the computer from which the client is making the connection</td>
</tr>
<tr>
<td>CLIENT_OS_USER</td>
<td>VARCHAR2(30)</td>
<td>CLIENT_OS_USE</td>
<td>Operating system user name of the client that is logging in</td>
</tr>
<tr>
<td>CLIENT_PROGRAM</td>
<td>VARCHAR2(30)</td>
<td>CLIENT_PROGRA</td>
<td>Name of the client program used to log in to the server</td>
</tr>
<tr>
<td>MODULE_NAME</td>
<td>VARCHAR2(30)</td>
<td>MODULE_NAME</td>
<td>Module name in the currently running application as set by the SET_MODULE Procedure in the DBMS_APPLICATION_INFO package, or the equivalent OCI attribute setting</td>
</tr>
<tr>
<td>MODULE_NAME_ACTION</td>
<td>VARCHAR2(30)</td>
<td>MODULE_NAME_A</td>
<td>A combination of the current module and the action being performed as set by either of the following procedures in the DBMS_APPLICATION_INFO package, or their equivalent OCI attribute setting:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTION</td>
<td>• SET_MODULE Procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• SET_ACTION Procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The attribute is specified as the module name followed by a period (.), followed by the action name (module_name.action_name).</td>
</tr>
<tr>
<td>ORACLE_FUNCTION</td>
<td>VARCHAR2(30)</td>
<td>ORACLE_FUNCTI</td>
<td>Function the session is currently executing. Valid functions are the BACKUP, COPIY, DATALOAD, and INMEMORY. BACKUP is set for sessions that are doing backup operations using RMAN. COPY is set for sessions that are doing image copies using RMAN. DATALOAD is set for sessions that are loading data using Oracle Data Pump.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>ORACLE_USER</td>
<td>VARCHAR2(30)</td>
<td>ORACLE_USER</td>
<td>Oracle Database user name</td>
</tr>
</tbody>
</table>
Table 134-1  (Cont.) DBMS_RESOURCE_MANAGER Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_MODULE</td>
<td>VARCHAR2(30)</td>
<td>SERVICE_MODULE</td>
<td>Combination of service and module names in this form: service_name.module_name</td>
</tr>
<tr>
<td>SERVICE_MODULE_ACTION</td>
<td>VARCHAR2(30)</td>
<td>SERVICE_MODULE_ACTION</td>
<td>Combination of service name, module name, and action name, in this form: service_name.module_name.action_name</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>VARCHAR2(30)</td>
<td>SERVICE_NAME</td>
<td>Service name used by the client to establish a connection</td>
</tr>
</tbody>
</table>

134.4 Summary of DBMSRESOURCE_MANAGER Subprograms

This table lists the DBMSRESOURCE_MANAGER subprograms and briefly describes them.

Table 134-2  DBMSRESOURCE_MANAGER Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_SQL_BLOCK Procedure</td>
<td>Indicates the start of a block of SQL statements to be treated as a group by resource manager</td>
</tr>
<tr>
<td>CALIBRATE_IO Procedure</td>
<td>Calibrates the I/O capabilities of storage</td>
</tr>
<tr>
<td>CLEAR_PENDING_AREA Procedure</td>
<td>Clears the work area for the resource manager</td>
</tr>
<tr>
<td>CREATECATEGORY Procedure</td>
<td>Creates a new resource consumer group category</td>
</tr>
<tr>
<td>CREATE_CDB_PLAN Procedure</td>
<td>Creates entries which define consolidation resource plans.</td>
</tr>
<tr>
<td>CREATE_CDB_PLAN_DIRECTIVE Procedure</td>
<td>Creates the plan directives of the consolidation resource plan</td>
</tr>
<tr>
<td>CREATE_CDB_PROFILE_DIRECTIVE Procedure</td>
<td>Creates the performance profile directives of the consolidation resource plan</td>
</tr>
<tr>
<td>CREATE_CONSUMER_GROUP Procedure</td>
<td>Creates entries which define resource consumer groups</td>
</tr>
<tr>
<td>CREATE_PENDING_AREA Procedure</td>
<td>Creates a work area for changes to resource manager objects</td>
</tr>
<tr>
<td>CREATE_PLAN Procedure</td>
<td>Creates entries which define resource plans</td>
</tr>
<tr>
<td>CREATE_PLAN_DIRECTIVE Procedure</td>
<td>Creates resource plan directives</td>
</tr>
<tr>
<td>CREATE_SIMPLE_PLAN Procedure</td>
<td>Creates a single-level resource plan containing up to eight consumer groups in one step</td>
</tr>
<tr>
<td>DELETECATEGORY Procedure</td>
<td>Deletes an existing resource consumer group category</td>
</tr>
<tr>
<td>DELETECDB_PLAN Procedure</td>
<td>Deletes the consolidation resource plan</td>
</tr>
<tr>
<td>DELETE_CDB_PLAN_DIRECTIVE Procedure</td>
<td>Deletes the plan directive of the consolidation resource plan</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>DELETE_CDB_PROFILE_DIRECTIVE Procedure</code></td>
<td>Deletes the performance profile directive of the consolidation resource plan</td>
</tr>
<tr>
<td><code>DELETE_CONSUMER_GROUP Procedure</code></td>
<td>Deletes entries which define resource consumer groups</td>
</tr>
<tr>
<td><code>DELETE_PLAN Procedure</code></td>
<td>Deletes the specified plan as well as all the plan directives it refers to</td>
</tr>
<tr>
<td><code>DELETE_PLAN CASCADE Procedure</code></td>
<td>Deletes the specified plan as well as all its descendants (plan directives, subplans, consumer groups)</td>
</tr>
<tr>
<td><code>DELETE_PLAN_DIRECTIVE Procedure</code></td>
<td>Deletes resource plan directives</td>
</tr>
<tr>
<td><code>DEQUEUE_PARALLEL_STATEMENT Procedure</code></td>
<td>Dequeues a parallel statement from the parallel statement queue</td>
</tr>
<tr>
<td><code>END_SQL_BLOCK Procedure</code></td>
<td>Indicates the end of a block of SQL statements that should be treated as a group by resource manager</td>
</tr>
<tr>
<td><code>SET_CONSUMER_GROUP_MAPPING Procedure</code></td>
<td>Adds, deletes, or modifies entries for the login and run-time attribute mappings</td>
</tr>
<tr>
<td><code>SET_CONSUMER_GROUP_MAPPING_PRI Procedure</code></td>
<td>Creates the session attribute mapping priority list</td>
</tr>
<tr>
<td><code>SET_INITIAL_CONSUMER_GROUP Procedure</code></td>
<td>Assigns the initial resource consumer group for a user (Caution: Deprecated Subprogram)</td>
</tr>
<tr>
<td><code>SUBMIT_PENDING_AREA Procedure</code></td>
<td>Submits pending changes for the resource manager</td>
</tr>
<tr>
<td><code>SWITCH_CONSUMER_GROUP_FOR_SESS Procedure</code></td>
<td>Changes the resource consumer group of a specific session</td>
</tr>
<tr>
<td><code>SWITCH_CONSUMER_GROUP_FOR_USER Procedure</code></td>
<td>Changes the resource consumer group for all sessions with a given user name</td>
</tr>
<tr>
<td><code>SWITCH_PLAN Procedure</code></td>
<td>Sets the current resource manager plan</td>
</tr>
<tr>
<td><code>UPDATE_CATEGORY Procedure</code></td>
<td>Updates an existing resource consumer group category</td>
</tr>
<tr>
<td><code>UPDATE_CDB_AUTOTASK_DIRECTIVE Procedure</code></td>
<td>Updates the plan directives with regard to automated maintenance tasks</td>
</tr>
<tr>
<td><code>UPDATE_CDB_DEFAULT_DIRECTIVE Procedure</code></td>
<td>Updates the default values for a consolidation plan</td>
</tr>
<tr>
<td><code>UPDATE_CDB_PLAN Procedure</code></td>
<td>Updates the consolidation resource plan</td>
</tr>
<tr>
<td><code>UPDATE_CDB_PLAN_DIRECTIVE Procedure</code></td>
<td>Updates the plan directives for a consolidation resource plan</td>
</tr>
<tr>
<td><code>UPDATE_CDB_PROFILE_DIRECTIVE Procedure</code></td>
<td>Updates the performance profile directives of the consolidation resource plan</td>
</tr>
<tr>
<td><code>UPDATE_CONSUMER_GROUP Procedure</code></td>
<td>Updates entries which define resource consumer groups</td>
</tr>
<tr>
<td><code>UPDATE_PLAN Procedure</code></td>
<td>Updates entries which define resource plans</td>
</tr>
<tr>
<td><code>UPDATE_PLAN_DIRECTIVE Procedure</code></td>
<td>Updates resource plan directives</td>
</tr>
<tr>
<td><code>VALIDATE_PENDING_AREA Procedure</code></td>
<td>Validates pending changes for the resource manager</td>
</tr>
</tbody>
</table>
134.4.1 BEGIN_SQL_BLOCK Procedure

This procedure, to be used with parallel statement queuing, indicates the start of a block of SQL statements that should be treated as a group by resource manager.

Syntax

```sql
DBMS_RESOURCE_MANAGER.BEGIN_SQL_BLOCK;
```

Usage Notes

For more information, see "Parallel Statement Queuing" and "Managing Parallel Statement Queuing with Resource Manager" in Oracle Database VLDB and Partitioning Guide.

134.4.2 CALIBRATE_IO Procedure

This procedure calibrates the I/O capabilities of storage. Calibration status is available from the $IO_CALIBRATION_STATUS view and results for a successful calibration run are located in DBA_RSRC_IO_CALIBRATE table.

Syntax

```sql
DBMS_RESOURCE_MANAGER.CALIBRATE_IO (num_physical_disks IN PLS_INTEGER DEFAULT 1, max_latency IN PLS_INTEGER DEFAULT 20, max_iops OUT PLS_INTEGER, max_mbps OUT PLS_INTEGER, actual_latency OUT PLS_INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_physical_disks</td>
<td>Approximate number of physical disks in the database storage. This parameter is used to determine the initial I/O load for the calibration run.</td>
</tr>
<tr>
<td>max_latency</td>
<td>Maximum tolerable latency in milliseconds for database-block-sized I/O requests</td>
</tr>
<tr>
<td>max_iops</td>
<td>Maximum number of I/O requests per second that can be sustained. The I/O requests are randomly-distributed, database-block-sized reads.</td>
</tr>
<tr>
<td>max_mbps</td>
<td>Maximum throughput of I/O that can be sustained, expressed in megabytes per second. The I/O requests are randomly-distributed, 1 megabyte reads.</td>
</tr>
<tr>
<td>actual_latency</td>
<td>Average latency of database-block-sized I/O requests at max_iops rate, expressed in milliseconds</td>
</tr>
</tbody>
</table>

Usage Notes

- Only users with the SYSDBA privilege can run this procedure. Qualified users must also turn on timed_statistics, and ensure asynch_io is enabled for datafiles. This can be
achieved by setting `filesystemio_options` to either `ASYNCH` or `SETALL`. One can also query the `asynch_io` status by means of the following SQL statement:

```sql
col name format a50
SELECT name, asynch_io FROM v$datafile f, v$iostat_file i
WHERE f.file# = i.file_no
AND   filetype_name = 'Data File'
```

- Only one calibration can be run at a time. If another calibration is initiated at the same time, it will fail.
- For an Oracle Real Application Clusters (Oracle RAC) database, the workload is simultaneously generated from all instances.
- In a multitenant container database (CDB), calibration can only be run from the CDB root (`CDB$ROOT`).
- Calibration is extremely disruptive to the database performance. It is strongly recommended to run calibration only when database users can tolerate severe deterioration to database performance.
- For optimal calibration results, no other database workloads should be running.

**See Also:**

*Oracle Database Performance Tuning Guide* for more information about calibration

### Examples

#### Example of using I/O Calibration procedure

```sql
SET SERVEROUTPUT ON
DECLARE
  lat  INTEGER;
  iops INTEGER;
  mbps INTEGER;
BEGIN
  -- DBMS_RESOURCE_MANAGER.CALIBRATE_IO (<DISKS>, <MAX_LATENCY>, iops, mbps, lat);
  DBMS_RESOURCE_MANAGER.CALIBRATE_IO (2, 10, iops, mbps, lat);
  DBMS_OUTPUT.PUT_LINE ('max_iops = ' || iops);
  DBMS_OUTPUT.PUT_LINE ('latency  = ' || lat);
  DBMS_OUTPUT.PUT_LINE ('max_mbps = ' || mbps);
END;
/
```

#### View for I/O calibration results

```sql
SQL> desc v$io_calibration_status
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
STATUS                                             VARCHAR2(13)
CALIBRATION_TIME                                   TIMESTAMP(3)

SQL> desc gv$io_calibration_status
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
```
### Column explanation:

#### STATUS:
- **IN PROGRESS**: Calibration in Progress (Results from previous calibration run displayed, if available)
- **READY**: Results ready and available from earlier run
- **NOT AVAILABLE**: Calibration results not available.

#### CALIBRATION_TIME: End time of the last calibration run

### DBA table that stores I/O Calibration results

```sql
SQL> desc DBA_RSRC_IO_CALIBRATE
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
START_TIME                                         TIMESTAMP(6)
END_TIME                                           TIMESTAMP(6)
MAX_IOPS                                           NUMBER
MAX_MBPS                                           NUMBER
MAX_PMBPS                                          NUMBER
LATENCY                                            NUMBER
NUM_PHYSICAL_DISKS                                 NUMBER
```

- **comment on table DBA_RSRC_IO_CALIBRATE is**
  'Results of the most recent I/O calibration'
- **comment on column DBA_RSRC_IO_CALIBRATE.START_TIME is**
  'start time of the most recent I/O calibration'
- **comment on column DBA_RSRC_IO_CALIBRATE.END_TIME is**
  'end time of the most recent I/O calibration'
- **comment on column DBA_RSRC_IO_CALIBRATE.MAX_IOPS is**
  'maximum number of data-block read requests that can be sustained per second'
- **comment on column DBA_RSRC_IO_CALIBRATE.MAX_MBPS is**
  'maximum megabytes per second of maximum-sized read requests that can be sustained'
- **comment on column DBA_RSRC_IO_CALIBRATE.MAX_PMBPS is**
  'maximum megabytes per second of large I/O requests that can be sustained by a single process'
- **comment on column DBA_RSRC_IO_CALIBRATE.LATENCY is**
  'latency for data-block read requests'
- **comment on column DBA_RSRC_IO_CALIBRATE.NUM_PHYSICAL_DISKS is**
  'number of physical disks in the storage subsystem (as specified by user)'
```
134.4.3 CLEAR_PENDING_AREA Procedure

This procedure clears pending changes for the resource manager.

Syntax

```
DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;
```

134.4.4 CREATE_CATEGORY Procedure

This procedure creates a new consumer group category. The primary purpose of this attribute is to support Exadata I/O Resource Manager category plans.

The view `DBA_RSRC_CATEGORIES` defines the currently defined categories. The `ADMINISTRATIVE`, `INTERACTIVE`, `BATCH`, `MAINTENANCE`, and `OTHER` categories are available.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_CATEGORY (    
    category    IN    VARCHAR2,    
    comment     IN    VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 134-4 CREATE_CATEGORY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>Name of consumer group category</td>
</tr>
<tr>
<td>comment</td>
<td>User comment</td>
</tr>
</tbody>
</table>

134.4.5 CREATE_CDB_PLAN Procedure

Creates entries which define consolidation resource plans.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN (    
    plan                    IN    VARCHAR2(32),    
    comment                 IN    VARCHAR2(2000) DEFAULT NULL);
```

### Parameters

**Table 134-5 CREATE_CDB_PLAN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>comment</td>
<td>User comment</td>
</tr>
</tbody>
</table>
Usage Notes

This procedure can be run only from the CDB root (CDB$ROOT).

134.4.6 CREATE_CDB_PLAN_DIRECTIVE Procedure

This procedure creates the plan directives of the consolidation resource plan. Plan directives specify the resource allocation policy for pluggable databases (PDBs).

Syntax

```sql
DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE (
    plan                    IN    VARCHAR2,
    pluggable_database      IN    VARCHAR2,
    comment                 IN    VARCHAR2 (2000) DEFAULT '',
    shares                  IN    NUMBER          DEFAULT NULL,
    utilization_limit       IN    NUMBER          DEFAULT NULL,
    parallel_server_limit   IN    NUMBER          DEFAULT NULL,
    memory_limit            IN    NUMBER          DEFAULT 100,
    memory_min              IN    NUMBER          DEFAULT 0);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>pluggable_database</td>
<td>Name of the PDB</td>
</tr>
<tr>
<td>comment</td>
<td>User comment</td>
</tr>
<tr>
<td>shares</td>
<td>Specifies the share of resource allocation for the PDB. CPU Resource Manager and Exadata I/O Resource Manager are enabled by specifying shares for each PDB. The shares parameter is also used for Parallel Statement Queuing. If no share is specified, the default is obtained from the default directive, specified through UPDATE_CDB_DEFAULT_DIRECTIVE Procedure.</td>
</tr>
<tr>
<td>utilization_limit</td>
<td>Specifies the maximum percentage of the CDB’s CPU and Exadata I/O resources that the PDB can utilize. CPU Resource Manager and Exadata I/O Resource Manager can also be limited by setting the CPU_COUNT parameter for the PDB.</td>
</tr>
</tbody>
</table>
Table 134-6  (Cont.) CREATE_CDB_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel_server_limit</td>
<td>Parallel servers that the PDB can use after which parallel statements are queued. Alternatively, you can set the parallel_servers_target at the PDB level.</td>
</tr>
<tr>
<td></td>
<td>A PDB can set a lower limit for parallel execution servers than the limit specified in the CDB resource plan. When the PARALLEL_SERVERS_TARGET initialization parameter is set in a PDB, and parallel execution server limit is specified for a PDB in the CDB resource plan, then the lower limit is used. For example, assume that the parallel_servers_target initialization parameter is set to 100 in the CDB root and parallel_server_limit is set to 70 for hrpdb in the CDB resource plan. Also, assume that hrpdb has its parallel_servers_target initialization parameter set to 50. In this case, the limit for parallel execution servers for hrpdb is 50, because 50 is lower than the CDB resource plan limit of 70 for hrpdb.</td>
</tr>
<tr>
<td>memory_limit</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Exadata Smart Flash Cache and Exadata PMEM Cache.</td>
</tr>
<tr>
<td>memory_min</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Exadata Smart Flash Cache and Exadata PMEM Cache.</td>
</tr>
</tbody>
</table>

Usage Notes

- The default value for shares, utilization_limit, and parallel_server_limit is NULL. When a user specifies NULL, or does not specify a value, this indicates that the default value should be used.
- This procedure can be run only from the CDB root (CDB$ROOT).

134.4.7 CREATE_CDB_PROFILE_DIRECTIVE Procedure

This procedure creates the performance profile directives of the consolidation resource plan. The directives specify the resource allocation policy for pluggable databases (PDBs) that use the performance profile.

For a PDB to use the new performance profile, the PDB must have the DB_PERFORMANCE_PROFILE initialization parameter set to the profile name.
This procedure provides an easy way to specify the directive for a large number of PDBs with the same resource requirements. Each PDB with a `DB_PERFORMANCE_PROFILE` initialization parameter set to the performance profile name inherits the settings specified by this directive, including the shares, utilization limit, and so on.

### Syntax

```sql
DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE (  
    plan                    IN    VARCHAR2,  
    profile                 IN    VARCHAR2,  
    comment                 IN    VARCHAR2 (2000) DEFAULT '',  
    shares                  IN    NUMBER          DEFAULT NULL,  
    utilization_limit       IN    NUMBER          DEFAULT NULL,  
    parallel_server_limit   IN    NUMBER          DEFAULT NULL,  
    memory_limit            IN    NUMBER          DEFAULT 100,  
    memory_min              IN    NUMBER          DEFAULT 0);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>profile</td>
<td>Name of the performance profile</td>
</tr>
<tr>
<td>comment</td>
<td>User comment</td>
</tr>
<tr>
<td>shares</td>
<td>Specifies the share of resource allocation for PDBs that use the performance profile. CPU Resource Manager and Exadata I/O Resource Manager are enabled by specifying shares for each PDB. The <code>shares</code> parameter is also used for Parallel Statement Queuing. If no share is specified, the default is obtained from the default directive, specified through <code>UPDATE_CDB_DEFAULT_DIRECTIVE</code> Procedure.</td>
</tr>
<tr>
<td>utilization_limit</td>
<td>Specifies the maximum percentage of CPU Resource Manager and Exadata I/O Resource Manager that PDBs that use the performance profile can utilize.</td>
</tr>
<tr>
<td>parallel_server_limit</td>
<td>Specifies the maximum percentage of parallel servers that PDBs that use the performance profile can use.</td>
</tr>
<tr>
<td>memory_limit</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Exadata Smart Flash Cache and Exadata PMEM Cache.</td>
</tr>
<tr>
<td>memory_min</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Exadata Smart Flash Cache and Exadata PMEM Cache.</td>
</tr>
</tbody>
</table>

### 134.4.8 CREATE_CONSUMER_GROUP Procedure

This procedure creates entries which define resource consumer groups.

### Syntax

```sql
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (  
    consumer_group  IN VARCHAR2,  
    comment         IN VARCHAR2 DEFAULT NULL,  
```
Parameters

Table 134-8  CREATE_CONSUMER_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_group</td>
<td>Name of the consumer group</td>
</tr>
<tr>
<td>comment</td>
<td>User comment</td>
</tr>
<tr>
<td>cpu_mth</td>
<td>Name of CPU resource allocation method (deprecated)</td>
</tr>
<tr>
<td>mgmt_mth</td>
<td>Name of CPU resource allocation method</td>
</tr>
<tr>
<td>category</td>
<td>Describes the category of the consumer group. The primary purpose of this attribute is to support Exadata I/O Resource Manager category plans. The view <code>DBA_RSRC_CATEGORIES</code> defines the currently defined categories. Categories can be modified, using the <code>CREATE_CATEGORY Procedure</code>, <code>UPDATE_CATEGORY Procedure</code>, and <code>DELETE_CATEGORY Procedure</code>.</td>
</tr>
</tbody>
</table>

134.4.9 CREATE_PENDING_AREA Procedure

This procedure makes changes to resource manager objects.

All changes to the plan schema must be done within a pending area. The pending area can be thought of as a "scratch" area for plan schema changes. The administrator creates this pending area, makes changes as necessary, possibly validates these changes, and only when the submit is completed do these changes become active.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

Usage Notes

You may, at any time while the pending area is active, view the current plan schema with your changes by selecting from the appropriate user views.

At any time, you may clear the pending area if you want to stop the current changes. You may also call the `VALIDATE` procedure to confirm whether the changes you have made are valid. You do not have to perform your changes in a given order to maintain a consistent group of entries. These checks are also implicitly done when the pending area is submitted.

Note:

Oracle allows "orphan" consumer groups (in other words, consumer groups that have no plan directives that refer to them). This is in anticipation that an administrator may want to create a consumer group that is not currently being used, but will be used in the future.
For resource plans, the following rules must be adhered to, and they are checked whenever the validate or submit procedures are executed:

- No plan schema may contain any loops.
- All plans and consumer groups referred to by plan directives must exist.
- All plans must have plan directives that refer to either plans or consumer groups.
- All percentages in any given level must not add up to greater than 100 for the emphasis resource allocation method.
- No plan may be deleted that is currently being used as a top plan by an active instance.
- The plan directive parameter, `parallel_degree_limit_p1`, may only appear in plan directives that refer to consumer groups (that is, not at subplans).
- There cannot be more than 28 plan directives coming from any given plan (that is, no plan can have more than 28 children).
- There cannot be more than 28 consumer groups in any active plan schema.
- Plans and consumer groups use the same namespace; therefore, no plan can have the same name as any consumer group.
- There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan schema. This ensures that a session not covered by the currently active plan is allocated resources as specified by the `OTHER_GROUPS` directive.

**Note:**

These rules are not applicable for CDB resource plans.

If any of the preceding rules are broken when checked by the `VALIDATE` or `SUBMIT` procedures, then an informative error message is returned. You may then make changes to fix one or more problems and reissue the validate or submit procedures.

### 134.4.10 CREATE_PLAN Procedure

This procedure creates entries which define resource plans.

**Syntax**

```sql
DBMS_RESOURCE_MANAGER.CREATE_PLAN (
    plan                       IN   VARCHAR2,
    comment                    IN   VARCHAR2 DEFAULT NULL,
    cpu_mth                    IN   VARCHAR2 DEFAULT NULL, -- deprecated
    active_sess_pool_mth       IN   VARCHAR2 DEFAULT 'ACTIVE_SESS_POOL_ABSOLUTE',
    parallel_degree_limit_mth  IN   VARCHAR2 DEFAULT 'PARALLEL_DEGREE_LIMIT_ABSOLUTE',
    queueing_mth               IN   VARCHAR2 DEFAULT 'FIFO_TIMEOUT',
    mgmt_mth                   IN   VARCHAR2 DEFAULT 'EMPHASIS',
    sub_plan                   IN   BOOLEAN DEFAULT FALSE,
    max_iops                   IN   NUMBER DEFAULT NULL,
    max_mbps                   IN   NUMBER DEFAULT NULL);
```
Parameters

Table 134-9 CREATE_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the resource plan</td>
</tr>
<tr>
<td>comment</td>
<td>User comment</td>
</tr>
<tr>
<td>cpu_mth</td>
<td>Allocation method for CPU resources (deprecated)</td>
</tr>
<tr>
<td>active_sess_pool_mth</td>
<td>Active session pool resource allocation method. Limits the number of active sessions. All other sessions are inactive and wait in a queue to be activated. ACTIVE_SESS_POOL_ABSOLUTE is the default and only method available.</td>
</tr>
<tr>
<td>parallel_degree_limit_mth</td>
<td>Resource allocation method for specifying a limit on the degree of parallelism of any operation. PARALLEL_DEGREE_LIMIT_ABSOLUTE is the default and only method available.</td>
</tr>
<tr>
<td>queueing_mth</td>
<td>Queuing resource allocation method. Controls order in which queued inactive sessions will execute. FIFO_TIMEOUT is the default and only method available</td>
</tr>
<tr>
<td>mgmt_mth</td>
<td>Resource allocation method for specifying how much resources (for example, CPU or I/O) each consumer group or sub-plan gets</td>
</tr>
<tr>
<td></td>
<td>• EMPHASIS - for multilevel plans that use percentages to specify how I/O resources are distributed among consumer groups</td>
</tr>
<tr>
<td></td>
<td>• RATIO - for single-level plans that use ratios to specify how I/O resources are distributed</td>
</tr>
<tr>
<td>sub_plan</td>
<td>If TRUE, indicates that this plan is only intended for use as a sub-plan. Sub-plans are not required to have an OTHER_GROUPS directive. Default is FALSE.</td>
</tr>
<tr>
<td>max_iops</td>
<td>Nonoperative</td>
</tr>
<tr>
<td>max_mbps</td>
<td>Nonoperative</td>
</tr>
</tbody>
</table>

Usage Notes

If you want to use any default resource allocation method, then you do not need to specify it when creating or updating a plan.

134.4.11 CREATE_PLAN_DIRECTIVE Procedure

This procedure creates resource plan directives.

Note:

The parameters max_utilization_limit and parallel_target_percentage are deprecated with Oracle Database 11g Release 1 (11.1.0.1), and are replaced by utilization_limit and parallel_server_limit.
Syntax

DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (  
  plan                         IN VARCHAR2,  
  group_or_subplan             IN VARCHAR2,  
  comment                      IN VARCHAR2 DEFAULT NULL,  
  cpu_p1                       IN NUMBER DEFAULT NULL, -- deprecated  
  cpu_p2                       IN NUMBER DEFAULT NULL, -- deprecated  
  cpu_p3                       IN NUMBER DEFAULT NULL, -- deprecated  
  cpu_p4                       IN NUMBER DEFAULT NULL, -- deprecated  
  cpu_p5                       IN NUMBER DEFAULT NULL, -- deprecated  
  cpu_p6                       IN NUMBER DEFAULT NULL, -- deprecated  
  cpu_p7                       IN NUMBER DEFAULT NULL, -- deprecated  
  cpu_p8                       IN NUMBER DEFAULT NULL, -- deprecated  
  active_sess_pool_p1          IN NUMBER DEFAULT NULL,  
  queueing_p1                  IN NUMBER DEFAULT NULL,  
  parallel_degree_limit_p1     IN NUMBER DEFAULT NULL,  
  switch_group                 IN VARCHAR2 DEFAULT NULL,  
  switch_time                  IN NUMBER DEFAULT NULL,  
  switch_estimate              IN BOOLEAN DEFAULT FALSE,  
  max_est_exec_time            IN NUMBER DEFAULT NULL,  
  undo_pool                    IN NUMBER DEFAULT NULL,  
  max_idle_time                IN NUMBER DEFAULT NULL,  
  max_idle_blocker_time        IN NUMBER DEFAULT NULL,  
  switch_time_in_call          IN NUMBER DEFAULT NULL, -- deprecated  
  mgmt_p1                      IN NUMBER DEFAULT NULL,  
  mgmt_p2                      IN NUMBER DEFAULT NULL,  
  mgmt_p3                      IN NUMBER DEFAULT NULL,  
  mgmt_p4                      IN NUMBER DEFAULT NULL,  
  mgmt_p5                      IN NUMBER DEFAULT NULL,  
  mgmt_p6                      IN NUMBER DEFAULT NULL,  
  mgmt_p7                      IN NUMBER DEFAULT NULL,  
  mgmt_p8                      IN NUMBER DEFAULT NULL,  
  switch_io_megabytes          IN NUMBER DEFAULT NULL,  
  switch_io_reqs               IN NUMBER DEFAULT NULL,  
  switch_for_call              IN BOOLEAN DEFAULT FALSE,  
  max_utilization_limit        IN NUMBER DEFAULT NULL, -- deprecated  
  parallel_target_percentage   IN NUMBER DEFAULT NULL, -- deprecated  
  parallel_server_limit        IN NUMBER DEFAULT NULL,  
  utilization_limit            IN NUMBER DEFAULT NULL,  
  switch_io_logical            IN NUMBER DEFAULT NULL,  
  switch_elapsed_time          IN NUMBER DEFAULT NULL,  
  shares                       IN NUMBER DEFAULT NULL,  
  parallel_stmt_critical       IN VARCHAR2 DEFAULT NULL,  
  session_pga_limit            IN NUMBER DEFAULT NULL,  
  pq_timeout_action            IN NUMBER DEFAULT NULL,  
  parallel_queue_timeout       IN NUMBER DEFAULT NULL,);  

Note:

Oracle recommends that you use shares instead of mgmt_p*.
## Parameters

### Table 134-10  CREATE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the resource plan</td>
</tr>
<tr>
<td>group_or_subplan</td>
<td>Name of the consumer group or subplan</td>
</tr>
<tr>
<td>comment</td>
<td>Comment for the plan directive</td>
</tr>
<tr>
<td>cpu_p1</td>
<td>-- deprecated: use mgmt_p1 or, even better, shares instead</td>
</tr>
<tr>
<td>cpu_p2</td>
<td>-- deprecated: use mgmt_p2 or, even better, shares instead</td>
</tr>
<tr>
<td>cpu_p3</td>
<td>-- deprecated: use mgmt_p3 or, even better, shares instead</td>
</tr>
<tr>
<td>cpu_p4</td>
<td>-- deprecated: use mgmt_p4 or, even better, shares instead</td>
</tr>
<tr>
<td>cpu_p5</td>
<td>-- deprecated: use mgmt_p5 or, even better, shares instead</td>
</tr>
<tr>
<td>cpu_p6</td>
<td>-- deprecated: use mgmt_p6 or, even better, shares instead</td>
</tr>
<tr>
<td>cpu_p7</td>
<td>-- deprecated: use mgmt_p7 or, even better, shares instead</td>
</tr>
<tr>
<td>cpu_p8</td>
<td>-- deprecated: use mgmt_p8 or, even better, shares instead</td>
</tr>
<tr>
<td>active_sess_pool_p1</td>
<td>Specifies maximum number of sessions that can currently have an active call</td>
</tr>
<tr>
<td>queueing_p1</td>
<td>Specified time (in seconds) after which a call in the inactive session queue (waiting for execution) will time out. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>parallel_degree_limit_p1</td>
<td>Specifies a limit on the degree of parallelism for any operation. Default is NULL, which means unlimited. If the value is 0, then all operations will be serial.</td>
</tr>
<tr>
<td>switch_group</td>
<td>Specifies consumer group to switch to, once a switch condition is met. If the group name is CANCEL_SQL, then the current call is canceled when the switch condition is met. If the group name is KILL_SESSION, then the session is killed when the switch condition is met. If the group name is LOG_ONLY, then no action is taken other than recording this event via SQL monitor. Default is NULL.</td>
</tr>
<tr>
<td>switch_time</td>
<td>Specifies the time on CPU (not elapsed time) that a session can execute before an action is taken. Default is NULL, which means unlimited. As with other switch directives, if switch_for_call is TRUE, the number of CPUs is accumulated from the start of a call. Otherwise, the number of CPUs is accumulated for the length of the session.</td>
</tr>
<tr>
<td>switch_estimate</td>
<td>If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. This is used in conjunction with the switch_time directive. Default value is FALSE.</td>
</tr>
<tr>
<td>max_est_exec_time</td>
<td>Specifies the maximum execution time (in CPU seconds) allowed for a session. If the optimizer estimates that an operation will take longer than MAX_EST_EXEC_TIME, the operation is not started and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>undo_pool</td>
<td>Limits the size in kilobytes of the undo records corresponding to uncommitted transactions by this consumer group.</td>
</tr>
<tr>
<td>max_idle_time</td>
<td>Indicates the maximum session idle time. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>max_idle_blocker_time</td>
<td>Maximum amount of time in seconds that a session can be idle while blocking another session’s acquisition of a resource.</td>
</tr>
<tr>
<td>switch_time_in_call</td>
<td>Deprecated. If this parameter is specified, switch_time is set to switch_time_in_call (in seconds) and switch_for_call is effectively set to TRUE. It is better to use switch_time and switch_for_call.</td>
</tr>
</tbody>
</table>
| mgmt_p1         | Resource allocation value for level 1 (replaces cpu_p1):  
  - EMPHASIS - specifies the resource percentage at the first level  
  - RATIO - specifies the weight of resource usage                                                                                                                                                                                                                                                                                                                |
| mgmt_p2         | Resource allocation value for level 2 (replaces cpu_p2)  
  - EMPHASIS - specifies the resource percentage at the second level  
  - RATIO - non-applicable                                                                                                                                                                                                                                                                                                                                       |
| mgmt_p3         | Resource allocation value for level 3 (replaces cpu_p3)  
  - EMPHASIS - specifies the resource percentage at the third level  
  - RATIO - non-applicable                                                                                                                                                                                                                                                                                                                                       |
| mgmt_p4         | Resource allocation value for level 4 (replaces cpu_p4)  
  - EMPHASIS - specifies the resource percentage at the fourth level  
  - RATIO - non-applicable                                                                                                                                                                                                                                                                                                                                       |
| mgmt_p5         | Resource allocation value for level 5 (replaces cpu_p5)  
  - EMPHASIS - specifies the resource percentage at the fifth level  
  - RATIO - non-applicable                                                                                                                                                                                                                                                                                                                                       |
| mgmt_p6         | Resource allocation value for level 6 (replaces cpu_p6)  
  - EMPHASIS - specifies the resource percentage at the sixth level  
  - RATIO - non-applicable                                                                                                                                                                                                                                                                                                                                       |
| mgmt_p7         | Resource allocation value for level 7 (replaces cpu_p7)  
  - EMPHASIS - specifies the resource percentage at the seventh level  
  - RATIO - non-applicable                                                                                                                                                                                                                                                                                                                                       |
| mgmt_p8         | Resource allocation value for level 8 (replaces cpu_p8)  
  - EMPHASIS - specifies the resource percentage at the eighth level  
  - RATIO - non-applicable                                                                                                                                                                                                                                                                                                                                       |
### Table 134-10  (Cont.) CREATE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch_io_megabytes</td>
<td>Specifies the amount of I/O (in MB) that a session can issue before an action is taken. Default is NULL, which means unlimited. As with other switch directives, if switch_for_call is TRUE, the number of CPUs is accumulated from the start of a call. Otherwise, the number of CPUs is accumulated for the length of the session.</td>
</tr>
<tr>
<td>switch_io_reqs</td>
<td>Specifies the number of I/O requests that a session can issue before an action is taken. Default is NULL, which means unlimited. As with other switch directives, if switch_for_call is TRUE, the number of CPUs is accumulated from the start of a call. Otherwise, the number of CPUs is accumulated for the length of the session.</td>
</tr>
<tr>
<td>switch_for_call</td>
<td>Specifies that if an action is taken because of the switch_time, switch_io_megabytes, switch_io_reqs, switch_io_logical or switch_elapsed_time parameters, the consumer group is restored to its original consumer group at the end of the top call. Default is NULL, which means that the original consumer group is not restored at the end of the top call.</td>
</tr>
<tr>
<td>max_utilization_limit</td>
<td>-- deprecated: use utilization_limit instead</td>
</tr>
<tr>
<td>parallel_target_percent</td>
<td>-- deprecated: use parallel_sever_limit instead</td>
</tr>
<tr>
<td>parallel_queue_timeout</td>
<td>Specifies the time (in seconds) that a parallel statement may remain in its Consumer Group’s parallel statement queue before it is removed and terminated with an error (ORA- 07454).</td>
</tr>
</tbody>
</table>

**Note:**

You can use the `pq_timeout_action` parameter to specify the action to be taken when a parallel statement is removed from the queue.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel_sever_limit</td>
<td>Specifies the maximum percentage of parallel_servers_target parallel servers that the Consumer Group can use, after which parallel statements are queued.</td>
</tr>
<tr>
<td>utilization_limit</td>
<td>Resource limit. Currently it includes CPU and I/O for Exadata. For CPU, this limits the CPU utilization for the consumer group. For Exadata I/O, this limits the disk utilization for the consumer group. This does not apply to parallel servers.</td>
</tr>
<tr>
<td>switch_io_logical</td>
<td>Number of logical I/Os that will trigger the action specified by switch_group. As with other switch directives, if switch_for_call is TRUE, the number of logical I/Os is accumulated from the start of a call. Otherwise, the number of logical I/Os is accumulated for the length of the session.</td>
</tr>
</tbody>
</table>
### Table 134-10 (Cont.) CREATE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch_elapsed_time</td>
<td>Elapsed time that will trigger the action specified by switch_group. As with other switch directives, if switch_for_call is TRUE, the elapsed time is accumulated from the start of a call. Otherwise, the elapsed time is accumulated for the length of the session.</td>
</tr>
<tr>
<td>shares</td>
<td>Specifies the share of resource allocation for the consumer group. CPU Resource Manager and Exadata I/O Resource Manager are enabled by specifying shares for each consumer group. The shares parameter is also used for Parallel Statement Queuing. If CPU Resource Manager and Exadata I/O Resource Manager are enabled, then the default value is 1.</td>
</tr>
<tr>
<td>parallel_stmt_critical</td>
<td>If set to BYPASS_QUEUE, parallel statements from the Consumer Group are not queued, regardless of the PARALLEL_DEGREE_POLICY parameter value.</td>
</tr>
<tr>
<td></td>
<td>If set to QUEUE, all the parallel statements from the consumer group, irrespective of the parallel_degree_policy parameter value, are eligible for queuing. Default is FALSE, which means that parallel statements are eligible for queuing, based on the parallel_degree_policy parameter value.</td>
</tr>
<tr>
<td>session_pga_limit</td>
<td>Maximum amount of untunable PGA (in MB) that a session in this consumer group can allocate before being terminated. NULL (default) indicates no limit. SQL operations that allocate tunable PGA (operations that can opt to use temp space) are not controlled by this limit.</td>
</tr>
<tr>
<td>pq_timeout_action</td>
<td>Specifies the action to be taken when a parallel statement is removed from the queue due to parallel_queue_timeout. The values are:</td>
</tr>
<tr>
<td></td>
<td>• CANCEL — The parallel statement is terminated with error ORA-7454</td>
</tr>
<tr>
<td></td>
<td>• RUN — The SQL statement runs immediately, and might get downgraded if parallel servers are unavailable</td>
</tr>
<tr>
<td></td>
<td>The default action of this parameter is CANCEL.</td>
</tr>
</tbody>
</table>

#### Usage Notes

- All parameters default to NULL.
- For max_idle_time and max_idle_blocker_time, PMON will check these limits once a minute. If it finds a session that has exceeded one of the limits, it will forcibly kill the session and clean up all its state.
- The parameter switch_for_call is mostly useful for three-tier applications where the mid-tier server is implementing session pooling. By using switch_for_call, the resource usage of one client will not affect a future client that happens to be executed on the same session.
- An error is thrown if PQ_TIMEOUT_ACTION is specified, but PARALLEL_QUEUE_TIMEOUT is not specified.
- Specifies the action to be taken when a parallel statement is removed from the queue.
134.4.12 CREATE_SIMPLE_PLAN Procedure

This procedure creates a single-level resource plan containing up to eight consumer groups in one step. You do not need to create a pending area manually before creating a resource plan, or use the CREATE_CONSUMER_GROUP and CREATE_RESOURCE_PLAN_DIRECTIVES procedures separately.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN (
  simple_plan      IN  VARCHAR2  DEFAULT NULL,
  consumer_group1  IN  VARCHAR2  DEFAULT NULL,
  group1_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  consumer_group2  IN  VARCHAR2  DEFAULT NULL,
  group2_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  consumer_group3  IN  VARCHAR2  DEFAULT NULL,
  group3_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  consumer_group4  IN  VARCHAR2  DEFAULT NULL,
  group4_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  consumer_group5  IN  VARCHAR2  DEFAULT NULL,
  group5_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  consumer_group6  IN  VARCHAR2  DEFAULT NULL,
  group6_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  consumer_group7  IN  VARCHAR2  DEFAULT NULL,
  group7_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  consumer_group8  IN  VARCHAR2  DEFAULT NULL,
  group8_cpu       IN  NUMBER    DEFAULT NULL,   -- deprecated
  group1_percent   IN  NUMBER    DEFAULT NULL,
  group2_percent   IN  NUMBER    DEFAULT NULL,
  group3_percent   IN  NUMBER    DEFAULT NULL,
  group4_percent   IN  NUMBER    DEFAULT NULL,
  group5_percent   IN  NUMBER    DEFAULT NULL,
  group6_percent   IN  NUMBER    DEFAULT NULL,
  group7_percent   IN  NUMBER    DEFAULT NULL,
  group8_percent   IN  NUMBER    DEFAULT NULL);
```

Parameters

Table 134-11  CREATE_SIMPLE_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple_plan</td>
<td>Name of the resource plan</td>
</tr>
<tr>
<td>consumer_group1</td>
<td>Name of the consumer group</td>
</tr>
<tr>
<td>group1_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>consumer_group2</td>
<td>Name of the consumer group</td>
</tr>
<tr>
<td>group2_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>consumer_group3</td>
<td>Name of the consumer group</td>
</tr>
<tr>
<td>group3_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>consumer_group4</td>
<td>Name of the consumer group</td>
</tr>
<tr>
<td>group4_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>consumer_group5</td>
<td>Name of the consumer group</td>
</tr>
</tbody>
</table>
Table 134-11 (Cont.) CREATE_SIMPLE_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group5_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>consumer_group6</td>
<td>Name of the consumer group</td>
</tr>
<tr>
<td>group6_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>consumer_group7</td>
<td>Name of the consumer group</td>
</tr>
<tr>
<td>group7_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>consumer_group8</td>
<td>OTHER_GROUPS - all sessions that aren't mapped to a consumer group</td>
</tr>
<tr>
<td>group8_cpu</td>
<td>Percentage for group (deprecated)</td>
</tr>
<tr>
<td>group1_percent</td>
<td>Percentage of resources allocated for this consumer group</td>
</tr>
<tr>
<td>group2_percent</td>
<td>Percentage of resources allocated for this consumer group</td>
</tr>
<tr>
<td>group3_percent</td>
<td>Percentage of resources allocated for this consumer group</td>
</tr>
<tr>
<td>group4_percent</td>
<td>Percentage of resources allocated for this consumer group</td>
</tr>
<tr>
<td>group5_percent</td>
<td>Percentage of resources allocated for this consumer group</td>
</tr>
<tr>
<td>group6_percent</td>
<td>Percentage of resources allocated for this consumer group</td>
</tr>
<tr>
<td>group7_percent</td>
<td>Percentage of resources allocated for this consumer group</td>
</tr>
<tr>
<td>group8_percent</td>
<td>Percentage of resources allocated to other groups</td>
</tr>
</tbody>
</table>

134.4.13 DELETE_CATEGORY Procedure

This procedure deletes an existing resource consumer group category.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CATEGORY (category IN VARCHAR2);
```

Parameters

Table 134-12 DELETE_CATEGORY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>Name of consumer group category</td>
</tr>
</tbody>
</table>

134.4.14 DELETE_CDB_PLAN Procedure

This procedure deletes the consolidation resource plan.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN (plan IN VARCHAR2(32) DEFAULT NULL);
```
Parameters

Table 134-13  DELETE_CDB_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure can be run only from the CDB root (CDB$ROOT).

134.4.15 DELETE_CDB_PLAN_DIRECTIVE Procedure

This procedure deletes the plan directives of the consolidation resource plan. Once the plan directive is deleted, the pluggable database will get the default resource allocation.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN_DIRECTIVE (plan IN VARCHAR2(32) DEFAULT NULL,
pluggable_database IN VARCHAR2(32) DEFAULT NULL);
```

Parameters

Table 134-14  DELETE_CDB_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>pluggable_database</td>
<td>Name of the pluggable database in which the plan directive is to be deleted</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure can be run only from the CDB root (CDB$ROOT).

134.4.16 DELETE_CDB_PROFILE_DIRECTIVE Procedure

This procedure deletes the performance profile directive of the consolidation resource plan. Once the directive is deleted, the pluggable databases (PDBs) that use the performance profile use the default resource allocation.

For a PDB to use a performance profile, the PDB must have the DB_PERFORMANCE_PROFILE initialization parameter set to the performance profile name.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CDB_PROFILE_DIRECTIVE (plan IN VARCHAR2(32) DEFAULT NULL,
profile IN VARCHAR2(32) DEFAULT NULL);
```
Parameters

Table 134-15  DELETE_CDB_PROFILE_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>profile</td>
<td>Name of the performance profile directive to be deleted</td>
</tr>
</tbody>
</table>

Usage Notes
This procedure can be run only from the CDB root ($CDB$ROOT).

134.4.17 DELETE_CONSUMER_GROUP Procedure

This procedure deletes entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP (  
    consumer_group IN VARCHAR2);  
```

Parameters

Table 134-16  DELETE_CONSUMER_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_group</td>
<td>Name of the consumer group to be deleted</td>
</tr>
</tbody>
</table>

134.4.18 DELETE_PLAN Procedure

This procedure deletes the specified plan as well as all the plan directives to which it refers.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN (  
    plan IN VARCHAR2);  
```

Parameters

Table 134-17  DELETE_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the resource plan to delete</td>
</tr>
</tbody>
</table>
134.4.19 DELETE_PLAN_CASCADE Procedure

This procedure deletes the specified plan and all of its descendants (plan directives, subplans, consumer groups). Mandatory objects and directives are not deleted.

Syntax

```sql
DBMS_RESOURCE_MANAGER.DELETE_PLAN_CASCADE (plan IN VARCHAR2);
```

Parameters

Table 134-18  DELETE_PLAN_CASCADE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the plan</td>
</tr>
</tbody>
</table>

Usage Notes

If `DELETE_PLAN_CASCADE` encounters any error, then it rolls back the operation, and nothing is deleted.

134.4.20 DELETE_PLAN_DIRECTIVE Procedure

This procedure deletes resource plan directives.

Syntax

```sql
DBMS_RESOURCE_MANAGER.DELETE_PLAN_DIRECTIVE (plan              IN VARCHAR2,
                                                group_or_subplan  IN VARCHAR2);
```

Parameters

Table 134-19  DELETE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the resource plan</td>
</tr>
<tr>
<td>group_or_subplan</td>
<td>Name of the group or subplan</td>
</tr>
</tbody>
</table>

134.4.21 DEQUEUE_PARALLEL_STATEMENT Procedure

This procedure dequeues a parallel statement from the parallel statement queue.

If the `PARALLEL_DEGREE_POLICY` initialization parameter is set to `AUTO` or `ADAPTIVE`, then parallel statement queuing is enabled. If a parallel statement is in the parallel statement queue, then you can use this procedure to dequeue the parallel statement so that it runs immediately.
Syntax

```
DEQUEUE_PARALLEL_STATEMENT ( 
    session_id      IN  PLS_INTEGER,
    session_serial  IN  PLS_INTEGER,
    inst_id         IN  PLS_INTEGER  DEFAULT NULL,
    sql_id          IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>The session id of the session running the parallel statement to be dequeued.</td>
</tr>
<tr>
<td>session_serial</td>
<td>The serial number of the session.</td>
</tr>
<tr>
<td>inst_id</td>
<td>Instance ID where the session is running. If NULL, then the current instance is used.</td>
</tr>
<tr>
<td>sql_id</td>
<td>The SQL ID of the session's statement to dequeue. If the session is running SQL with a different SQL ID, then the statement is not dequeued.</td>
</tr>
</tbody>
</table>

134.4.22 END_SQL_BLOCK Procedure

This procedure, to be used with parallel statement queuing, indicates the end of a block of SQL statements that should be treated as a group by resource manager.

Syntax

```
END_SQL_BLOCK;
```

Usage Notes

For more information, see "Parallel Statement Queuing" and "Managing Parallel Statement Queuing with Resource Manager" in Oracle Database VLDB and Partitioning Guide.

134.4.23 SET_CONSUMER_GROUP_MAPPING Procedure

This procedure adds, deletes, or modifies entries that map sessions to consumer groups, based on the session's login and runtime attributes.

Syntax

```
SET_CONSUMER_GROUP_MAPPING(
    attribute     IN VARCHAR2,
    value         IN VARCHAR2,
    consumer_group IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 134-21  SET_CONSUMER_GROUP_MAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>Mapping attribute to add or modify. It can be one of the Constants listed.</td>
</tr>
<tr>
<td>value</td>
<td>Attribute value to match. This includes both absolute mapping and regular expressions.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of the mapped consumer group, or NULL to delete a mapping</td>
</tr>
</tbody>
</table>

Usage Notes

- If no mapping exists for the given attribute and value, a mapping to the given consumer group will be created. If a mapping already exists for the given attribute and value, the mapped consumer group will be updated to the one given. If the consumer_group argument is NULL, then any mapping from the given attribute and value will be deleted.

- The subprogram supports simple regex expressions for the value parameter. It implements the same semantics as the SQL 'LIKE' operator. Specifically, it uses '%' as a multicharacter wildcard and '_' as a single character wildcard. The '\' character can be used to escape the wildcards. Note that wildcards can only be used if the attribute is one of the following:
  - CLIENT_OS_USER
  - CLIENT_PROGRAM
  - CLIENT_MACHINE
  - MODULE_NAME
  - MODULE_NAME_ACTION
  - SERVICE_MODULE
  - SERVICE_MODULE_ACTION

- Consumer group mapping comparisons for DBMS_RESOURCE_MANAGER.CLIENT_PROGRAM are performed by stripping the @ sign and following characters from V$SESSION.PROGRAM before comparing it to the CLIENT_PROGRAM value supplied.

134.4.24 SET_CONSUMER_GROUP_MAPPING_PRI Procedure

Multiple attributes of a session can be used to map the session to a consumer group. This procedure prioritizes the attribute mappings.

Syntax

```sql
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI(
    explicit               IN NUMBER,
    oracle_user            IN NUMBER,
    service_name           IN NUMBER,
    client_os_user         IN NUMBER,
)```

client_program         IN NUMBER,
client_machine         IN NUMBER,
module_name            IN NUMBER,
module_name_action     IN NUMBER,
service_module         IN NUMBER,
service_module_action  IN NUMBER,
client_id              IN NUMBER DEFAULT 11);

Parameters

Table 134-22  SET_CONSUMER_GROUP_MAPPING_PRI Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit</td>
<td>Priority of the explicit mapping</td>
</tr>
<tr>
<td>oracle_user</td>
<td>Priority of the Oracle user name mapping</td>
</tr>
<tr>
<td>service_name</td>
<td>Priority of the client service name mapping</td>
</tr>
<tr>
<td>client_os_user</td>
<td>Priority of the client operating system user name mapping</td>
</tr>
<tr>
<td>client_program</td>
<td>Priority of the client program mapping</td>
</tr>
<tr>
<td>client_machine</td>
<td>Priority of the client machine mapping</td>
</tr>
<tr>
<td>module_name</td>
<td>Priority of the application module name mapping</td>
</tr>
<tr>
<td>module_name_action</td>
<td>Priority of the application module name and action mapping</td>
</tr>
<tr>
<td>service_module</td>
<td>Priority of the service name and application module name mapping</td>
</tr>
<tr>
<td>module_name_action</td>
<td>Priority of the service name, application module name, and application action mapping</td>
</tr>
<tr>
<td>client_id</td>
<td>Client identifier</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure requires that you include the pseudo-attribute `explicit` as an argument. It must be set to 1. It indicates that explicit consumer group switches have the highest priority. You explicitly switch consumer groups with these package procedures:
  - DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP
  - DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS
  - DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER
- Each priority value must be a unique integer from 1 to 11. Together, they establish an ordering where 1 is the highest priority and 11 is the lowest.

134.4.25 SET_INITIAL_CONSUMER_GROUP Procedure

This deprecated procedure sets the initial resource consumer group for a user.

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs.
This procedure is deprecated in Release 11gR1. While the procedure remains available in the package, Initial Consumer Group is set by the session-to-consumer group mapping rules.

Syntax

```
DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (  
    user             IN   VARCHAR2,  
    consumer_group   IN   VARCHAR2);
```

Parameters

**Table 134-23  SET_INITIAL_CONSUMER_GROUP Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Name of the user</td>
</tr>
<tr>
<td>consumer_group</td>
<td>User's initial consumer group</td>
</tr>
</tbody>
</table>

Usage Notes

- The `ADMINISTER_RESOURCE_MANAGER` or the `ALTER USER` system privilege are required to be able to execute this procedure. The user, or `PUBLIC`, must be directly granted switch privilege to a consumer group before it can be set to be the user's initial consumer group. Switch privilege for the initial consumer group cannot come from a role granted to that user.

- If the initial consumer group for a user has never been set, then the user's initial consumer group is automatically the consumer group: `DEFAULT_CONSUMER_GROUP`.

- `DEFAULT_CONSUMER_GROUP` has switch privileges granted to `PUBLIC`; therefore, all users are automatically granted switch privilege for this consumer group. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group now have `DEFAULT_CONSUMER_GROUP` as their initial consumer group. All currently active sessions belonging to a deleted consumer group are switched to `DEFAULT_CONSUMER_GROUP`.

Note:

These semantics are similar to those for `ALTER USER DEFAULT ROLE`.
134.4.26 SUBMIT_PENDING_AREA Procedure

This procedure submits pending changes for the resource manager. It clears the pending area after validating and committing the changes (if valid).

**Note:**

A call to `SUBMIT_PENDING_AREA` may fail even if `VALIDATE_PENDING_AREA` succeeds. This may happen if a plan being deleted is loaded by an instance after a call to `VALIDATE_PENDING_AREA`, but before a call to `SUBMIT_PENDING_AREA`.

**Syntax**

```sql
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

134.4.27 SWITCH_CONSUMER_GROUP_FOR_SESS Procedure

This procedure changes the resource consumer group of a specific session. It also changes the consumer group of any parallel execution servers that are related to the top user session.

**Syntax**

```sql
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS (
    session_id      IN NUMBER,
    session_serial  IN NUMBER,
    consumer_group  IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>SID column from the view V$SESSION</td>
</tr>
<tr>
<td>session_serial</td>
<td>SERIAL# column from view V$SESSION.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of the consumer group to which to switch</td>
</tr>
</tbody>
</table>

134.4.28 SWITCH_CONSUMER_GROUP_FOR_USER Procedure

This procedure changes the resource consumer group for all sessions with a given user ID. It also changes the consumer group of any parallel execution servers that are related to the top user session.

**Syntax**

```sql
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER (
    user            IN VARCHAR2,
    consumer_group  IN VARCHAR2);
```
Parameters

Table 134-25 SWITCH_CONSUMER_GROUP_FOR_USER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Name of the user</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of the consumer group to which to switch</td>
</tr>
</tbody>
</table>

Usage Notes

- The SWITCH_CONSUMER_GROUP_FOR_SESS Procedure and the SWITCH_CONSUMER_GROUP_FOR_USER procedures let you raise or lower the allocation of CPU resources of certain sessions or users. This provides a functionality similar to the nice command on UNIX.
- These procedures cause the session to be moved into the newly specified consumer group immediately.

134.4.29 SWITCH_PLAN Procedure

This procedure sets the current resource manager plan.

Syntax

```sql
DBMS_RESOURCE_MANAGER.SWITCH_PLAN(
    plan_name                     IN   VARCHAR2,
    sid                           IN   VARCHAR2 DEFAULT '*',
    allow_scheduler_plan_switches IN   BOOLEAN DEFAULT TRUE);
```

Parameters

Table 134-26 SWITCH_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_name</td>
<td>Name of the plan to which to switch. Passing in an empty string (*) for the plan_name, disables the resource manager</td>
</tr>
<tr>
<td>sid</td>
<td>The sid parameter is relevant only in an Oracle Real Application Clusters environment. This parameter lets you change the plan for a particular instance. Specify the sid of the instance where you want to change the plan. Or specify ‘*’ if you want Oracle to change the plan for all instances.</td>
</tr>
<tr>
<td>allow_scheduler_plan_switches</td>
<td>FALSE - disables automated plan switches by the job scheduler at window boundaries. To reenable automated plan switches, switch_plan must be called again by the administrator with allow_scheduler_plan_switches set to TRUE. By default automated plan switches by the job scheduler are enabled.</td>
</tr>
</tbody>
</table>
134.4.30 UPDATE_CATEGORY Procedure

This procedure updates an existing resource consumer group category.

Syntax

```sql
DBMS_RESOURCE_MANAGER.UPDATE_CATEGORY (
    category        IN    VARCHAR2,
    new_comment     IN    VARCHAR2  DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>Name of consumer group category</td>
</tr>
<tr>
<td>new_comment</td>
<td>User comment</td>
</tr>
</tbody>
</table>

134.4.31 UPDATE_CDB_AUTOTASK_DIRECTIVE Procedure

This procedure updates the plan directives with regard to automated maintenance tasks in the CDB root (CDB$ROOT).

By default, all maintenance tasks occur directly in the PDBs themselves.

Syntax

```sql
DBMS_RESOURCE_MANAGER.UPDATE_CDB_AUTOTASK_DIRECTIVE (
    plan                        IN    VARCHAR2,
    new_comment                 IN    VARCHAR2       DEFAULT NULL,
    new_shares                  IN    NUMBER         DEFAULT NULL,
    new_utilization_limit       IN    NUMBER         DEFAULT NULL,
    new_parallel_server_limit   IN    NUMBER         DEFAULT NULL,
    new_memory_limit            IN    NUMBER         DEFAULT NULL,
    new_memory_min              IN    NUMBER         DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user comment</td>
</tr>
<tr>
<td>new_shares</td>
<td>Specifies the new share of resource allocation for CDB root’s automated maintenance tasks</td>
</tr>
<tr>
<td>new_utilization_limit</td>
<td>Specifies the new maximum percentage of CPU that automated maintenance tasks in the CDB root can utilize</td>
</tr>
<tr>
<td>new_parallel_server_limit</td>
<td>Specifies the new maximum percentage of parallel servers that automated maintenance tasks in the CDB root are allowed to use</td>
</tr>
</tbody>
</table>
Table 134-28 (Cont.) UPDATE_CDB_AUTOTASK_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_memory_limit</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
<tr>
<td>new_memory_min</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
</tbody>
</table>

Usage Notes

- By default for automated maintenance tasks, the values are
  - shares: -1
  - utilization_limit: 90
  - parallel_server_limit: 100
- The shares = -1 means that the automated maintenance tasks get an allocation of 20% of the system. If the user specifies the shares, it behaves the same properties as the other CDB plan directive functions. If the user does not change the shares or later changes it back to -1, autotask will get 20% of the system.
- This procedure can be run only from the CDB root.

134.4.32 UPDATE_CDB_DEFAULT_DIRECTIVE Procedure

This procedure updates the plan directives of the consolidation resource plan.

Syntax

DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE (plan                        IN VARCHAR2    DEFAULT NULL,
new_comment                 IN VARCHAR2    DEFAULT NULL,
new_shares                  IN NUMBER      DEFAULT NULL,
new_utilization_limit       IN NUMBER      DEFAULT NULL,
new_parallel_server_limit   IN NUMBER      DEFAULT NULL,
new_memory_limit            IN NUMBER      DEFAULT NULL,
new_memory_min              IN NUMBER      DEFAULT NULL);

Parameters

Table 134-29 UPDATE_CDB_DEFAULT_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user comment</td>
</tr>
<tr>
<td>new_shares</td>
<td>Specifies the share of resource allocation for the pluggable database. CPU Resource Manager and Exadata I/O Resource Manager are enabled by specifying shares for each PDB. The new_shares parameter is also used for Parallel Statement Queuing.</td>
</tr>
<tr>
<td>new_utilization_limit</td>
<td>Specifies the maximum percentage of CPU that the pluggable database can utilize.</td>
</tr>
</tbody>
</table>
### Table 134-29 (Cont.) UPDATE_CDB_DEFAULT_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_parallel_server_limit</td>
<td>Specifies the maximum percentage of parallel_servers_target parallel servers that the pluggable database can use.</td>
</tr>
<tr>
<td>new_memory_limit</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
<tr>
<td>new_memory_min</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
</tbody>
</table>

#### Usage Notes

- By default, the default values are
  - new_shares: 1
  - utilization_limit: 100
  - parallel_server_limit: 100
- Note that the default values are NULL. This has the same meaning as in UPDATE_CDB_PLAN_DIRECTIVE Procedure. If the user does not specify a value, the value will not be modified.
- This procedure can be run only from the CDB root (CDB$ROOT).

### 134.4.33 UPDATE_CDB_PLAN Procedure

This procedure updates the consolidation resource plan.

#### Syntax

```sql
DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN ( plan                        IN VARCHAR2(32),
                                          new_comment                 IN VARCHAR2(2000)   DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>new_comment</td>
<td>User comment</td>
</tr>
</tbody>
</table>

#### Usage Notes

This procedure can be run only from the CDB root (CDB$ROOT).
134.4.34 UPDATE_CDB_PLAN_DIRECTIVE Procedure

Updates the plan directives for a consolidation resource plan. Plan directives specify
the resource allocation policy for pluggable databases (PDBs).

Syntax

```sql
DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN_DIRECTIVE (
    plan                        IN    VARCHAR2 (30),
    pluggable_database          IN    VARCHAR2 (30),
    new_comment                 IN    VARCHAR2 (200)  DEFAULT NULL,
    new_shares                  IN    NUMBER          DEFAULT NULL,
    new_utilization_limit       IN    NUMBER          DEFAULT NULL,
    new_parallel_server_limit   IN    NUMBER          DEFAULT NULL,
    new_memory_limit            IN    NUMBER          DEFAULT NULL,
    new_memory_min              IN    NUMBER          DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>pluggable_database</td>
<td>Name of the pluggable database</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user comment</td>
</tr>
<tr>
<td>new_shares</td>
<td>The share of resource allocation for the pluggable database</td>
</tr>
<tr>
<td></td>
<td>CPU Resource Manager is enabled by specifying shares for each PDB. The shares parameter is also used for Parallel Statement Queuing. If no share is specified, the default is obtained from the default directive, specified through the UPDATE_CDB_DEFAULT_DIRECTIVE Procedure.</td>
</tr>
<tr>
<td>new_utilization_limit</td>
<td>The new maximum percentage of CPU that the pluggable database can utilize</td>
</tr>
<tr>
<td>new_parallel_server_limit</td>
<td>The new maximum percentage of parallel servers that the pluggable database can use</td>
</tr>
<tr>
<td>new_memory_limit</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
<tr>
<td>new_memory_min</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
</tbody>
</table>

Usage Notes

- The default value for the `new_*` parameters is `NULL` which indicates that the existing value is left unchanged. If the user does not specify one of the arguments when calling this function, the value is not modified.
- This procedure can be run only from the CDB root (`CDB$ROOT`).
134.4.35 UPDATE_CDB_PROFILE_DIRECTIVE Procedure

This procedure updates the performance profile directives of the consolidation resource plan. The directives specify the resource allocation policy for pluggable databases (PDBs) that use the performance profile.

For a PDB to use a performance profile, the PDB must have the `DB_PERFORMANCE_PROFILE` initialization parameter set to the performance profile name.

Syntax

```sql
DBMS_RESOURCE_MANAGER.UPDATE_CDB_PROFILE_DIRECTIVE (
    plan                        IN    VARCHAR2,
    profile                     IN    VARCHAR2,
    new_comment                 IN    VARCHAR2 (2000) DEFAULT '',
    new_shares                  IN    NUMBER          DEFAULT NULL,
    new_utilization_limit       IN    NUMBER          DEFAULT NULL,
    new_parallel_server_limit   IN    NUMBER          DEFAULT NULL,
    new_memory_limit            IN    NUMBER          DEFAULT 100,
    new_memory_min              IN    NUMBER          DEFAULT 0);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the consolidation plan</td>
</tr>
<tr>
<td>profile</td>
<td>Name of the performance profile</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user comment</td>
</tr>
<tr>
<td>new_shares</td>
<td>The share of resource allocation for the PDBs that use the performance profile</td>
</tr>
<tr>
<td>new_utilization_limit</td>
<td>The new maximum percentage of CPU that PDBs that use the performance profile can use</td>
</tr>
<tr>
<td>new_parallel_server_limit</td>
<td>The new maximum percentage of parallel servers that PDBs that use the performance profile can use</td>
</tr>
<tr>
<td>new_memory_limit</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
<tr>
<td>new_memory_min</td>
<td>This parameter is only applicable to Oracle Exadata storage for configuring the Database Smart Flash Cache.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure can be run only from the CDB root (`CDB$ROOT`).
134.4.36 UPDATE_CONSUMER_GROUP Procedure

This procedure updates entries which define resource consumer groups.

Syntax

```sql
DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP (  
consumer_group  IN VARCHAR2,
new_comment     IN VARCHAR2 DEFAULT NULL,
new_cpu_mth     IN VARCHAR2 DEFAULT NULL,
new_mgmt_mth    IN VARCHAR2 DEFAULT NULL,
new_category    IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_group</td>
<td>Name of consumer group</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user comment</td>
</tr>
<tr>
<td>new_cpu_mth</td>
<td>Name of new method for CPU resource allocation (deprecated)</td>
</tr>
<tr>
<td>new_mgmt_mth</td>
<td>Name of new method for CPU resource allocation</td>
</tr>
<tr>
<td>new_category</td>
<td>New consumer group category</td>
</tr>
</tbody>
</table>

Usage Notes

If the parameters to the `UPDATE_CONSUMER_GROUP` procedure are not specified, then they remain unchanged in the data dictionary.

134.4.37 UPDATE_PLAN Procedure

This procedure updates entries which define resource plans.

Syntax

```sql
DBMS_RESOURCE_MANAGER.UPDATE_PLAN (  
plan                               IN VARCHAR2,
new_comment                        IN VARCHAR2 DEFAULT NULL,
new_cpu_mth                        IN VARCHAR2 DEFAULT NULL, -- deprecated
new_active_sess_pool_mth           IN VARCHAR2 DEFAULT NULL,
new_parallel_degree_limit_mth      IN VARCHAR2 DEFAULT NULL,
new_queueing_mth                   IN VARCHAR2 DEFAULT NULL,
new_mgmt_mth                       IN VARCHAR2 DEFAULT NULL,
new_sub_plan                       IN BOOLEAN DEFAULT FALSE,
new_max_iops                       IN NUMBER DEFAULT NULL,
new_max_mbps                       IN NUMBER DEFAULT NULL);
```
Parameters

Table 134-34  UPDATE_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of resource plan</td>
</tr>
<tr>
<td>new_comment</td>
<td>New user comment</td>
</tr>
<tr>
<td>new_cpu_mth</td>
<td>Name of new allocation method for CPU resources (deprecated)</td>
</tr>
<tr>
<td>new_active_sess_pool_mth</td>
<td>Name of new method for maximum active sessions</td>
</tr>
<tr>
<td>new_parallel_degree_lmit_mth</td>
<td>Name of new method for degree of parallelism</td>
</tr>
<tr>
<td>new_queueing_mth</td>
<td>Specifies type of queuing policy to use with active session pool feature</td>
</tr>
<tr>
<td>new_mgmt_mth</td>
<td>Resource allocation method for specifying how much resources (for example, CPU or I/O) each consumer group or sub-plan gets</td>
</tr>
<tr>
<td>- EMPHASIS</td>
<td>for multilevel plans that use percentages to specify how I/O resources are distributed among consumer groups.</td>
</tr>
<tr>
<td>- RATIO</td>
<td>for single-level plans that use ratios to specify how I/O resources are distributed.</td>
</tr>
<tr>
<td>new_sub_plan</td>
<td>New setting for whether the plan is only intended for use as a sub-plan</td>
</tr>
<tr>
<td>new_max_iops</td>
<td>Nonoperative</td>
</tr>
<tr>
<td>new_max_mbps</td>
<td>Nonoperative</td>
</tr>
</tbody>
</table>

Usage Notes

- If the parameters to UPDATE_PLAN Procedure are not specified, then they remain unchanged in the data dictionary.
- If you want to use any default resource allocation method, then you do not need to specify it when creating or updating a plan.

134.4.38 UPDATE_PLAN_DIRECTIVE Procedure

This procedure updates resource plan directives.

Note:

The parameters new_max_utilization_limit and new_parallel_target_percentage are deprecated with Oracle Database 11g Release 1 (12.1.0.1), and are replaced by new_utilization_limit and new_parallel_server_limit.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (plan IN VARCHAR2,
                                           group_or_subplan IN VARCHAR2,
                                           new_comment IN VARCHAR2 DEFAULT NULL,
                                           new_parallel_lmit_mth IN VARCHAR2,
                                           new_queueing_mth IN VARCHAR2,
                                           new_mgmt_mth IN VARCHAR2,
                                           new_sub_plan IN VARCHAR2,
                                           new_max_iops IN VARCHAR2,
                                           new_max_mbps IN VARCHAR2)
```
new_cpu_p1                       IN NUMBER   DEFAULT NULL, -- deprecated
new_cpu_p2                       IN NUMBER   DEFAULT NULL, -- deprecated
new_cpu_p3                       IN NUMBER   DEFAULT NULL, -- deprecated
new_cpu_p4                       IN NUMBER   DEFAULT NULL, -- deprecated
new_cpu_p5                       IN NUMBER   DEFAULT NULL, -- deprecated
new_cpu_p6                       IN NUMBER   DEFAULT NULL, -- deprecated
new_cpu_p7                       IN NUMBER   DEFAULT NULL, -- deprecated
new_cpu_p8                       IN NUMBER   DEFAULT NULL, -- deprecated
new_active_sess_pool_p1          IN NUMBER   DEFAULT NULL,
new_queueing_p1                  IN NUMBER   DEFAULT NULL,
new_parallel_degree_limit_p1     IN NUMBER   DEFAULT NULL,
new_switch_group                 IN VARCHAR2 DEFAULT NULL,
new_switch_time                  IN NUMBER   DEFAULT NULL,
new_switch_estimate              IN BOOLEAN  DEFAULT FALSE,
new_max_est_exec_time            IN NUMBER   DEFAULT NULL,
new_undo_pool                    IN NUMBER   DEFAULT NULL,
new_max_idle_time                IN NUMBER   DEFAULT NULL,
new_max_idle_blocker_time        IN NUMBER   DEFAULT NULL,
switch_time_in_call              IN NUMBER   DEFAULT NULL, -- deprecated
new_mgmt_p1                      IN NUMBER   DEFAULT NULL,
new_mgmt_p2                      IN NUMBER   DEFAULT NULL,
new_mgmt_p3                      IN NUMBER   DEFAULT NULL,
new_mgmt_p4                      IN NUMBER   DEFAULT NULL,
new_mgmt_p5                      IN NUMBER   DEFAULT NULL,
new_mgmt_p6                      IN NUMBER   DEFAULT NULL,
new_mgmt_p7                      IN NUMBER   DEFAULT NULL,
new_mgmt_p8                      IN NUMBER   DEFAULT NULL,
new_switch_io_megabytes          IN NUMBER   DEFAULT NULL,
new_switch_io_reqs               IN NUMBER   DEFAULT NULL,
new_switch_for_call              IN BOOLEAN  DEFAULT NULL,
new_max_utilization_limit        IN NUMBER   DEFAULT NULL,
new_parallel_target_percentage   IN NUMBER   DEFAULT NULL,
new_parallel_queue_timeout       IN NUMBER   DEFAULT NULL,
new_parallel_server_limit        IN NUMBER   DEFAULT NULL,
new_utilization_limit            IN NUMBER   DEFAULT NULL,
new_switch_io_logical            IN NUMBER   DEFAULT NULL,
new_switch_elapsed_time          IN NUMBER   DEFAULT NULL,
new_shares                       IN NUMBER   DEFAULT NULL,
new_parallel_stmt_critical       IN VARCHAR2 DEFAULT NULL,
new_session_pga_limit            IN NUMBER   DEFAULT NULL,
new_pq_timeout_action            IN NUMBER   DEFAULT NULL);

Parameters

Table 134-35  UPDATE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>Name of the resource plan</td>
</tr>
<tr>
<td>group_or_subplan</td>
<td>Name of the consumer group or subplan</td>
</tr>
<tr>
<td>new_comment</td>
<td>Comment for the plan directive</td>
</tr>
<tr>
<td>new_cpu_p1</td>
<td>Deprecated - use new_mgmt_p1 instead</td>
</tr>
<tr>
<td>new_cpu_p2</td>
<td>Deprecated - use new_mgmt_p2 instead</td>
</tr>
<tr>
<td>new_cpu_p3</td>
<td>Deprecated - use new_mgmt_p3 instead</td>
</tr>
<tr>
<td>new_cpu_p4</td>
<td>Deprecated - use new_mgmt_p4 instead</td>
</tr>
<tr>
<td>new_cpu_p5</td>
<td>Deprecated - use new_mgmt_p5 instead</td>
</tr>
</tbody>
</table>

Chapter 134  Summary of DBMS_RESOURCE_MANAGER Subprograms
### Table 134-35  UPDATE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_cpu_p6</td>
<td>Deprecated - use new_mgmt_p6 instead</td>
</tr>
<tr>
<td>new_cpu_p7</td>
<td>Deprecated - use new_mgmt_p7 instead</td>
</tr>
<tr>
<td>new_cpu_p8</td>
<td>Deprecated - use new_mgmt_p8 instead</td>
</tr>
<tr>
<td>new_active_sess_pool_p1</td>
<td>Specifies maximum number of concurrently active sessions for a consumer group. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_queueing_p1</td>
<td>Specified time (in seconds) after which a job in the inactive session queue (waiting for execution) will time out. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_parallel_degree_limit_p1</td>
<td>Specifies a limit on the degree of parallelism for any operation. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_switch_group</td>
<td>Specifies consumer group to which this session is switched if other switch criteria are met. Default is NULL. If the group name is 'CANCEL_SQL', the current call will be canceled when other switch criteria are met. If the group name is 'KILL_SESSION', the session will be killed when other switch criteria are met.</td>
</tr>
<tr>
<td>new_switch_time</td>
<td>Specifies time (in CPU seconds) that a session can execute before an action is taken. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_switch_estimate</td>
<td>If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. Default is FALSE.</td>
</tr>
<tr>
<td>new_max_est_exec_time</td>
<td>Specifies the maximum execution time (in CPU seconds) allowed for a session. If the optimizer estimates that an operation will take longer than MAX_EST_EXEC_TIME, the operation is not started and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_undo_pool</td>
<td>Limits the size in kilobytes of the undo records corresponding to uncommitted transactions by this consumer group</td>
</tr>
<tr>
<td>new_max_idle_time</td>
<td>Indicates the maximum session idle time. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_max_idle_blocker_time</td>
<td>Maximum amount of time in seconds that a session can be idle while blocking another session’s acquisition of a resource</td>
</tr>
<tr>
<td>new_switch_time_in_call</td>
<td>Deprecated. If this parameter is specified, new_switch_time will be effectively set to new_switch_time_in_call and new_switch_for_call will be effectively set to TRUE.</td>
</tr>
<tr>
<td>new_mgmt_p1</td>
<td>Resource allocation value for level 1 (replaces new_cpu_p1):</td>
</tr>
<tr>
<td></td>
<td>• EMPHASIS - specifies the resource percentage at the first level</td>
</tr>
<tr>
<td></td>
<td>• RATIO - specifies the weight of resource usage</td>
</tr>
<tr>
<td>new_mgmt_p2</td>
<td>Resource allocation value for level 2 (replaces new_cpu_p2):</td>
</tr>
<tr>
<td></td>
<td>• EMPHASIS - specifies the resource percentage at the second level</td>
</tr>
<tr>
<td></td>
<td>• RATIO - non-applicable</td>
</tr>
</tbody>
</table>
### Table 134-35 (Cont.) UPDATE_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_mgmt_p3</td>
<td>Resource allocation value for level 3 (replaces new_cpu_p3)</td>
</tr>
<tr>
<td></td>
<td>- EMPHASIS - specifies the resource percentage at the third level</td>
</tr>
<tr>
<td></td>
<td>- RATIO - non-applicable</td>
</tr>
<tr>
<td>new_mgmt_p4</td>
<td>Resource allocation value for level 4 (replaces new_cpu_p4)</td>
</tr>
<tr>
<td></td>
<td>- EMPHASIS - specifies the resource percentage at the fourth level</td>
</tr>
<tr>
<td></td>
<td>- RATIO - non-applicable</td>
</tr>
<tr>
<td>new_mgmt_p5</td>
<td>Resource allocation value for level 5 (replaces new_cpu_p5)</td>
</tr>
<tr>
<td></td>
<td>- EMPHASIS - specifies the resource percentage at the fifth level</td>
</tr>
<tr>
<td></td>
<td>- RATIO - non-applicable</td>
</tr>
<tr>
<td>new_mgmt_p6</td>
<td>Resource allocation value for level 6 (replaces new_cpu_p6)</td>
</tr>
<tr>
<td></td>
<td>- EMPHASIS - specifies the resource percentage at the sixth level</td>
</tr>
<tr>
<td></td>
<td>- RATIO - non-applicable</td>
</tr>
<tr>
<td>new_mgmt_p7</td>
<td>Resource allocation value for level 7 (replaces new_cpu_p7)</td>
</tr>
<tr>
<td></td>
<td>- EMPHASIS - specifies the resource percentage at the seventh level</td>
</tr>
<tr>
<td></td>
<td>- RATIO - non-applicable</td>
</tr>
<tr>
<td>new_mgmt_p8</td>
<td>Resource allocation value for level 8 (replaces new_cpu_p8)</td>
</tr>
<tr>
<td></td>
<td>- EMPHASIS - specifies the resource percentage at the eighth level</td>
</tr>
<tr>
<td></td>
<td>- RATIO - non-applicable</td>
</tr>
<tr>
<td>new_switch_io_megabytes</td>
<td>Specifies the amount of I/O (in MB) that a session can issue before an action is taken. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_switch_io_reqs</td>
<td>Specifies the number of I/O requests that a session can issue before an action is taken. Default is NULL, which means unlimited.</td>
</tr>
<tr>
<td>new_switch_for_call</td>
<td>Specifies that if an action is taken because of the new_switch_time, new_switch_io_megabytes, or new_switch_io_reqs parameters, the consumer group is restored to its original consumer group at the end of the top call. Default is FALSE, which means that the original consumer group is not restored at the end of the top call.</td>
</tr>
<tr>
<td>new_max_utilization_limit</td>
<td>Deprecated - use new_utilization_limit instead</td>
</tr>
<tr>
<td>new_parallel_target_percentage</td>
<td>Deprecated - use new_parallel_server_limit instead</td>
</tr>
<tr>
<td>new_parallel_server_limit</td>
<td>Parallel server limit. Setting this overwrites the limit for parallel server set by utilization_limit.</td>
</tr>
<tr>
<td>new_utilization_limit</td>
<td>Resource limit. For CPU, this limits the CPU utilization for the consumer group. For parallel servers, this limits the parallel servers used as a percentage of parallel_servers_target.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>new_switch_elapsed_time</td>
<td>Elapsed time that will trigger the action specified by switch_group. As with other switch directives, if new_switch_for_call is TRUE, the elapsed time is accumulated from the start of a call. Otherwise, the elapsed time is accumulated for the length of the session.</td>
</tr>
<tr>
<td>new_shares</td>
<td>Specifies the share of resource allocation for the pluggable database. CPU Resource Manager and Exadata I/O Resource Manager are enabled by specifying shares for each PDB. The shares parameter is also used for Parallel Statement Queuing. If CPU Resource Manager and Exadata I/O Resource Manager are enabled, then the default value is 1.</td>
</tr>
<tr>
<td>new_parallel Stmt_crit</td>
<td>If set to BYPASS_QUEUE, parallel statements from this consumer group are not queued.                                                                                               If set to QUEUE, all the parallel statements, irrespective of the parallel_degree_policy parameter value, from the consumer group get queued. Default is FALSE, which means that certain parallel statements are eligible for queueing depending upon the parallel_degree_policy parameter value.</td>
</tr>
<tr>
<td>new_session_pga_limit</td>
<td>Maximum amount of PGA in MB that sessions in this consumer group can allocate before being terminated. NULL (default) indicates no change.</td>
</tr>
<tr>
<td>new_parallel_queue_time</td>
<td>Specifies the time (in seconds) that a parallel statement may remain in its Consumer Group's parallel statement queue before it is removed. The default action of this parameter is ERROR. This action can be altered using the new_pq_timeout_action parameter.</td>
</tr>
<tr>
<td>new_pq_timeout_action</td>
<td>Specifies the action to be taken when a parallel statement is removed from the queue due to new_parallel_queue_timeout. The values are:                                                                                           • CANCEL — The SQL statement is terminated with error ORA-7454 • RUN — The SQL statement runs immediately, and might get downgraded if parallel servers are unavailable</td>
</tr>
</tbody>
</table>

**Usage Notes**

- If the parameters for UPDATE_PLAN_DIRECTIVE are left unspecified, then they remain unchanged in the data dictionary.
- For new_max_idle_time and new_max_idle_blocker_time, PMON will check these limits once a minute. If it finds a session that has exceeded one of the limits, it will forcibly kill the session and clean up all its state.
- The parameter new_switch_time_in_call is mostly useful for three-tier applications where the mid-tier server is implementing session pooling. By turning on new_switch_time_in_call, the resource usage of one client will not affect the consumer group of a future client that happens to be executed on the same session.
• To clear (zero or nullify) any numeric parameter in a resource plan directive, set it to -1 using the UPDATE_PLAN_DIRECTIVE Procedure.

134.4.39 VALIDATE_PENDING_AREA Procedure

This procedure validates pending changes for the resource manager.

**Syntax**

```
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```
The **DBMS_RESOURCE_MANAGER_PRIVS** package maintains privileges associated with the Resource Manager.

This chapter contains the following topics:

- Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms

**See Also:**

For more information on using the Database Resource Manager, see *Oracle Database Administrator’s Guide*.

### 135.1 Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms

This table lists the **DBMS_RESOURCE_MANAGER_PRIVS** subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT_SWITCH_CONSUMER_GROUP Procedure</td>
<td>Grants the privilege to switch to resource consumer groups</td>
</tr>
<tr>
<td>GRANT_SYSTEM_PRIVILEGE Procedure</td>
<td>Performs a grant of a system privilege</td>
</tr>
<tr>
<td>REVOKE_SWITCH_CONSUMER_GROUP Procedure</td>
<td>Revokes the privilege to switch to resource consumer groups.</td>
</tr>
<tr>
<td>REVOKE_SYSTEM_PRIVILEGE Procedure</td>
<td>Performs a revoke of a system privilege</td>
</tr>
</tbody>
</table>

### 135.1.1 GRANT_SWITCH_CONSUMER_GROUP Procedure

This procedure grants the privilege to switch to a resource consumer group.

**Syntax**

```sql
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ( 
    grantee_name   IN VARCHAR2, 
    consumer_group IN VARCHAR2, 
    grant_option   IN BOOLEAN);
```
Parameters

Table 135-2   GRANT_SWITCH_CONSUMER_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee_name</td>
<td>Name of the user or role to whom privilege is to be granted.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of consumer group.</td>
</tr>
<tr>
<td>grant_option</td>
<td>TRUE if grantee should be allowed to grant access, FALSE otherwise.</td>
</tr>
</tbody>
</table>

Usage Notes

If you grant permission to switch to a particular consumer group to a user, then that user can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to a role, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to PUBLIC, then any user can switch to that consumer group.

If the grant_option parameter is TRUE, then users granted switch privilege for the consumer group may also grant switch privileges for that consumer group to others.

In order to set the initial consumer group of a user, you must grant the switch privilege for that group to the user.

See Also:

DBMS_RESOURCE_MANAGER

Examples

BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott', 'mail_maintenance_group', true);
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.set_consumer_group_mapping(
    dbms_resource_manager.oracle_user, 'scott', 'mail_maintenance_group');
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/

135.1.2 GRANT_SYSTEM_PRIVILEGE Procedure

This procedure performs a grant of a system privilege to a user or role.

Syntax

DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (grantee_name IN VARCHAR2,
Parameters

Table 135-3  GRANT_SYSTEM_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee_name</td>
<td>Name of the user or role to whom privilege is to be granted.</td>
</tr>
<tr>
<td>privilege_name</td>
<td>Name of the privilege to be granted.</td>
</tr>
<tr>
<td>admin_option</td>
<td>TRUE if the grant is with admin_option, FALSE otherwise.</td>
</tr>
</tbody>
</table>

Usage Notes

Currently, Oracle provides only one system privilege for the Resource Manager: ADMINISTER_RESOURCE_MANAGER. Database administrators have this system privilege with the ADMIN option. The grantee and the revokee can either be a user or a role. Users that have been granted the system privilege with the ADMIN option can also grant this privilege to others.

Examples

The following call grants this privilege to a user called scott without the ADMIN option:

```sql
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (  
    grantee_name => 'scott',  
    privilege_name => 'ADMINISTER_RESOURCE_MANAGER',  
    admin_option => FALSE);  
END;
/
```

135.1.3 REVOKE_SWITCH_CONSUMER_GROUP Procedure

This procedure revokes the privilege to switch to a resource consumer group.

Syntax

```sql
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (  
  revokee_name IN VARCHAR2,  
  consumer_group IN VARCHAR2);  
```

Parameters

Table 135-4  REVOKE_SWITCH_CONSUMER_GROUP Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revokee_name</td>
<td>Name of user/role from which to revoke access.</td>
</tr>
<tr>
<td>consumer_group</td>
<td>Name of consumer group.</td>
</tr>
</tbody>
</table>
Usage Notes

If you revoke a user's switch privilege for a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail.

If you revoke the initial consumer group from a user, then that user will automatically be part of the DEFAULT_CONSUMER_GROUP consumer group when logging in.

If you revoke the switch privilege for a consumer group from a role, then any users who only had switch privilege for the consumer group through that role will not be able to switch to that consumer group.

If you revoke the switch privilege for a consumer group from PUBLIC, then any users who could previously only use the consumer group through PUBLIC will not be able to switch to that consumer group.

Examples

The following example revokes the privileges to switch to mail_maintenance_group from Scott:

BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP ('scott', 'mail_maintenance_group');
END;
/

135.1.4 REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure performs a revoke of a system privilege from a user or role.

Syntax

DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE (revokee_name IN VARCHAR2, privilege_name IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER');

Parameters

Table 135-5  REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revokee_name</td>
<td>Name of the user or role from whom privilege is to be revoked.</td>
</tr>
<tr>
<td>privilege_name</td>
<td>Name of the privilege to be revoked.</td>
</tr>
</tbody>
</table>

Examples

The following call revokes the ADMINISTER_RESOURCE_MANAGER from user scott:

BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE ('scott');
END;
/
The `DBMS_RESULT_CACHE` package provides an interface to allow the DBA to administer that part of the shared pool that is used by the SQL result cache and the PL/SQL function result cache.

Both these caches use the same infrastructure. Therefore, for example, `DBMS_RESULT_CACHE.BYPASS` determines whether both caches are bypassed or both caches are used, and `DBMS_RESULT_CACHE.FLUSH` flushes both all the cached results for SQL queries and all the cached results for PL/SQL functions.

This chapter contains the following topics:

- **Security Model**
- **Constants**
- **Summary of DBMS_RESULT_CACHE Subprograms**

### See Also:

- *Oracle Database Performance Tuning Guide*, for more information about "Result Cache Concepts"
- *Oracle Database PL/SQL Language Reference*, for more information about PL/SQL function Result Cache "Using the Cross-Session PL/SQL Function Result Cache"
- *Database PL/SQL Language Reference*, for more information about Result Cache Management.

### 136.1 DBMS_RESULT_CACHE Security Model

Only database administrators should be granted the `EXECUTE` privilege for this package.

### 136.2 DBMS_RESULT_CACHE Constants

The `DBMS_RESULT_CACHE` package defines several constants for specifying parameter values.

The following table describes these constants.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_BYPAS</td>
<td>CONSTANT VARCHAR2(10) := 'BYPASS';</td>
<td>Cache has been made temporarily unavailable.</td>
</tr>
</tbody>
</table>
### Table 136-1 (Cont.) DBMS_RESULT_CACHE Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_CORR</td>
<td>CONSTANT VARCHAR2(10) := 'CORRUPT';</td>
<td>The result cache is in an unusable state.</td>
</tr>
<tr>
<td>STATUS_DISA</td>
<td>CONSTANT VARCHAR2(10) := 'DISABLED';</td>
<td>Cache is not available.</td>
</tr>
<tr>
<td>STATUS_ENAB</td>
<td>CONSTANT VARCHAR2(10) := 'ENABLED';</td>
<td>Cache is available.</td>
</tr>
<tr>
<td>STATUS_SYNC</td>
<td>CONSTANT VARCHAR2(10) := 'SYNC';</td>
<td>Cache is available, but synchronizing with Oracle RAC nodes.</td>
</tr>
</tbody>
</table>

### 136.3 Summary of DBMS_RESULT_CACHE Subprograms

This table lists the DBMS_RESULT_CACHE subprograms and briefly describes them.

### Table 136-2 DBMS_RESULT_CACHE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK_LIST Function</td>
<td>Returns a pipelined BL_TABT.</td>
</tr>
<tr>
<td>BLACK_LIST_ADD Procedure</td>
<td>Adds a cache_id to the black-list.</td>
</tr>
<tr>
<td>BLACK_LIST_CLEAR Procedure</td>
<td>Removes all cache_ids from the black-list.</td>
</tr>
<tr>
<td>BLACK_LIST_REMOVE Procedure</td>
<td>Removes the cache_id from the black-list.</td>
</tr>
<tr>
<td>BYPASS Procedure</td>
<td>Sets the bypass mode for the Result Cache</td>
</tr>
<tr>
<td>FLUSH Function &amp; Procedure</td>
<td>Attempts to remove all the objects from the Result Cache, and depending on the arguments retains or releases the memory and retains or clears the statistics</td>
</tr>
<tr>
<td>INVALIDATE Functions &amp; Procedures</td>
<td>Invalidates all the result-set objects that dependent upon the specified dependency object</td>
</tr>
<tr>
<td>INVALIDATE_OBJECT Functions &amp; Procedures</td>
<td>Invalidates the specified result-set object(s)</td>
</tr>
<tr>
<td>MEMORY_REPORT Procedure</td>
<td>Produces the memory usage report for the Result Cache</td>
</tr>
<tr>
<td>STATUS Function</td>
<td>Checks the status of the Result Cache</td>
</tr>
</tbody>
</table>

### 136.3.1 BLACK_LIST Function

This table function returns all the blocklisted cache ids of a local instance.

**Syntax**

```sql
DBMS_RESULT_CACHE.BLACK_LIST
RETURN BL_TABT;
```
136.3.2 BLACK_LIST_ADD Procedure

This procedure adds a cache_id to the black-list.

Syntax

DBMS_RESULT_CACHE.BLACK_LIST_ADD (  
  cache_id   IN  VARCHAR2,  
  global     IN  BOOLEAN DEFAULT FALSE);

Parameters

Table 136-3   BLACK_LIST_ADD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache_id</td>
<td>The cache_id is added to the black-list.</td>
</tr>
</tbody>
</table>
| global    | TRUE — applies to all caches in a RAC cluster.  
           | FALSE (default) — applies only to the local instance cache. |

136.3.3 BLACK_LIST_CLEAR Procedure

This procedure removes all cache_ids from the black-list.

Syntax

DBMS_RESULT_CACHE.BLACK_LIST_CLEAR (  
  global   IN  BOOLEAN DEFAULT FALSE);

Parameters

Table 136-4   BLACK_LIST_CLEAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| global    | TRUE — applies to all caches in a RAC cluster.  
           | FALSE (default) — applies only to the local instance cache. |

136.3.4 BLACK_LIST_REMOVE Procedure

This procedures removes the cache_id from the black-list.

Syntax

DBMS_RESULT_CACHE.BLACK_LIST_REMOVE (  
  cache_id   IN  VARCHAR2,  
  global     IN  BOOLEAN DEFAULT FALSE);
Parameters

**Table 136-5  BLACK_LIST_REMOVE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache_id</td>
<td>The cache_id is removed from the black-list.</td>
</tr>
<tr>
<td>global</td>
<td>TRUE — applies to all caches in a RAC cluster.</td>
</tr>
<tr>
<td></td>
<td>FALSE (default) — applies only to the local instance cache.</td>
</tr>
</tbody>
</table>

136.3.5 BYPASS Procedure

This procedure sets the bypass mode for the Result Cache.

It sets one of the following bypass modes:

- When bypass mode is turned on, it implies that cached results are no longer used and that no new results are saved in the cache.
- When bypass mode is turned off, the cache resumes normal operation.

**Syntax**

```sql
DBMS_RESULT_CACHE.BYPASS ( 
  bypass_mode    IN   BOOLEAN, 
  session        IN   BOOLEAN);
```

**Parameters**

**Table 136-6  BYPASS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| bypass_mode | • TRUE => Result Cache usage is bypassed  
|           | • FALSE => Result Cache usage is turned on  |
| session    | • TRUE => Applies to current session  
|           | • FALSE (default) => Applies to all sessions  |

**Usage Notes**

This operation is database instance specific.

**Examples**

This operation can be used when there is a need to hot patch PL/SQL code in a running system. If a code-patch is applied to a PL/SQL module on which a result cached function directly or transitively depends, then the cached results associated with the result cache function are not automatically flushed (if the instance is not restarted/bounced). This must be manually achieved.

To ensure correctness during the patching process follow these steps:

1. Place the result cache in bypass mode, and flush existing result.
BEGIN
  DBMS_RESULT_CACHE.BYPASS(TRUE);
  DBMS_RESULT_CACHE.FLUSH;
END;
/

This step must be performed on each instance if in a Oracle Real Application Clusters environment.

2. Apply the PL/SQL code patches.

3. Resume use of the result cache, by turning off the cache bypass mode.

BEGIN
  DBMS_RESULT_CACHE.BYPASS(FALSE);
END;
/

This step must be performed on each instance if in a Oracle Real Application Clusters environment.

### 136.3.6 FLUSH Function & Procedure

This function and procedure attempts to remove all the objects from the Result Cache, and depending on the arguments retains or releases the memory and retains or clears the statistics.

#### Syntax

```
DBMS_RESULT_CACHE.FLUSH (  
    retainMem  IN  BOOLEAN DEFAULT FALSE,  
    retainSta  IN  BOOLEAN DEFAULT FALSE)  
RETURN BOOLEAN;
```

```
DBMS_RESULT_CACHE.FLUSH (  
    retainMem  IN  BOOLEAN DEFAULT FALSE,  
    retainSta  IN  BOOLEAN DEFAULT FALSE);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retainMem</td>
<td>• TRUE =&gt; retains the free memory in the cache</td>
</tr>
<tr>
<td></td>
<td>• FALSE (default) =&gt; releases the free memory to the system</td>
</tr>
<tr>
<td>retainSta</td>
<td>• TRUE =&gt; retains the existing cache statistics</td>
</tr>
<tr>
<td></td>
<td>• FALSE (default) =&gt; clears the existing cache statistics</td>
</tr>
</tbody>
</table>

#### Return Values

TRUE if successful in removing all the objects.
136.3.7 INVALIDATE Functions & Procedures

This function and procedure invalidates all the result-set objects that dependent upon the specified dependency object.

Syntax

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE}\left(\begin{array}{l}
\text{owner}} \quad \text{IN VARCHAR2,} \\
\text{name} \quad \text{IN VARCHAR2}
\end{array}\right)
\]
\[
\text{RETURN NUMBER;}
\]

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE}\left(\begin{array}{l}
\text{owner}} \quad \text{IN VARCHAR2,} \\
\text{name} \quad \text{IN VARCHAR2}
\end{array}\right);
\]

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE}\left(\begin{array}{l}
\text{object_id}} \quad \text{IN BINARY\_INTEGER}
\end{array}\right)
\]
\[
\text{RETURN NUMBER;}
\]

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE}\left(\begin{array}{l}
\text{object_id}} \quad \text{IN BINARY\_INTEGER}
\end{array}\right);
\]

Parameters

Table 136-8 INVALIDATE Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Schema name</td>
</tr>
<tr>
<td>name</td>
<td>Object name</td>
</tr>
<tr>
<td>object_id</td>
<td>Dictionary object number</td>
</tr>
</tbody>
</table>

Return Values

The number of objects invalidated.

136.3.8 INVALIDATE_OBJECT Functions & Procedures

This function and procedure invalidates the specified result-set object(s).

Syntax

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE\_OBJECT}\left(\begin{array}{l}
id} \quad \text{IN BINARY\_INTEGER}
\end{array}\right)
\]
\[
\text{RETURN NUMBER;}
\]

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE\_OBJECT}\left(\begin{array}{l}
id} \quad \text{IN BINARY\_INTEGER}
\end{array}\right);
\]

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE\_OBJECT}\left(\begin{array}{l}
cache_id} \quad \text{IN VARCHAR2}
\end{array}\right)
\]
\[
\text{RETURN NUMBER;}
\]

\[
\text{DBMS\_RESULT\_CACHE\_INVALIDATE\_OBJECT}\left(\begin{array}{l}
cache_id} \quad \text{IN VARCHAR2}
\end{array}\right);
\]
### 136.3.9 MEMORY_REPORT Procedure

This procedure produces the memory usage report for the Result Cache.

**Syntax**

```sql
DBMS_RESULT_CACHE.MEMORY_REPORT (  
    detailed   IN   BOOLEAN DEFAULT FALSE);
```

**Parameters**

**Table 136-10 MEMORY_REPORT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detailed</td>
<td>· TRUE =&gt; produces a more detailed report</td>
</tr>
<tr>
<td></td>
<td>· FALSE (default) =&gt; produces the standard report</td>
</tr>
</tbody>
</table>

**Usage Notes**

Invoking this procedure from SQL*Plus requires that the serveroutput be turned on.

**Examples**

```sql
SET SERVEROUTPUT ON
EXECUTE DBMS_RESULT_CACHE.MEMORY_REPORT;
```

### 136.3.10 STATUS Function

This function checks the status of the Result Cache.

**Syntax**

```sql
DBMS_RESULT_CACHE.STATUS  
    RETURN VARCHAR2;
```
Note:

For more information on the constants, see DBMS_RESULT_CACHE Constants.
With the `DBMS_RESUMABLE` package, you can suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution. In this way you can write applications without worrying about running into space-related errors.

This chapter contains the following topics:

- Operational Notes
- Summary of `DBMS_RESUMABLE` Subprograms

137.1 DBMS_RESUMABLE Operational Notes

When you suspend a statement, you should log the suspension in the alert log. You should also register a procedure to be executed when the statement is suspended. Using a view, you can monitor the progress of the statement and indicate whether the statement is currently executing or suspended.

Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held during a statement suspend and resume. When the error condition disappears, the suspended statement automatically resumes execution. A resumable space allocation can be suspended and resumed multiple times during execution.

A suspension timeout interval is associated with resumable space allocations. A resumable space allocation that is suspended for the timeout interval (the default is two hours) wakes up and returns an exception to the user. A suspended statement may be forced to throw an exception using the `DBMS_RESUMABLE.ABORT()` procedure.

137.2 Summary of DBMS_RESUMABLE Subprograms

This table lists the `DBMS_RESUMABLE` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT Procedure</td>
<td>Aborts a suspended resumable space allocation</td>
</tr>
<tr>
<td>GET_SESSION_TIMEOUT Function</td>
<td>Returns the current timeout value of the resumable space allocations for a session with <code>session_id</code></td>
</tr>
<tr>
<td>GET_TIMEOUT Function</td>
<td>Returns the current timeout value of resumable space allocations for the current session</td>
</tr>
<tr>
<td>SET_SESSION_TIMEOUT Procedure</td>
<td>Sets the timeout of resumable space allocations for a session with <code>session_id</code></td>
</tr>
<tr>
<td>SET_TIMEOUT Procedure</td>
<td>Sets the timeout of resumable space allocations for the current session</td>
</tr>
</tbody>
</table>
Table 137-1 (Cont.) DBMS_RESUMABLE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACE_ERROR_INFO</td>
<td>Looks for space-related errors in the error stack, otherwise returning FALSE</td>
</tr>
</tbody>
</table>

137.2.1 ABORT Procedure

This procedure aborts a suspended resumable space allocation.

The parameter `session_id` is the session ID in which the statement is executed. For a parallel DML/DDL, `session_id` is any session ID that participates in the parallel DML/DDL. This operation is guaranteed to succeed. The procedure can be called either inside or outside of the `AFTER SUSPEND` trigger.

Syntax

```sql
DBMS_RESUMABLE.ABORT (
    session_id  IN NUMBER);
```

Parameters

Table 137-2 ABORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>The session identifier of the resumable space allocation.</td>
</tr>
</tbody>
</table>

Usage Notes

To call an `ABORT` procedure, you must be the owner of the session with `session_id`, have `ALTER SYSTEM` privileges, or be a DBA.

137.2.2 GET_SESSION_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for a session with `session_id`.

Syntax

```sql
DBMS_RESUMABLE.GET_SESSION_TIMEOUT (  
    session_id  IN NUMBER)  
RETURN NUMBER;
```

Parameters

Table 137-3 GET_SESSION_TIMEOUT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>The session identifier of the resumable space allocation.</td>
</tr>
</tbody>
</table>
Return Values

Table 137-4  GET_SESSION_TIMEOUT Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>The current timeout value of resumable space allocations for a session with session_id. The timeout is returned in seconds.</td>
</tr>
</tbody>
</table>

Usage Notes

If session_id does not exist, the GET_SESSION_TIMEOUT function returns -1.

137.2.3 GET_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for the current session.

Syntax

DBMS_RESUMABLE.GET_TIMEOUT
RETURN NUMBER;

Return Values

Table 137-5  GET_TIMEOUT Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>The current timeout value of resumable space allocations for the current session. The returned value is in seconds.</td>
</tr>
</tbody>
</table>

Usage Notes

If the current session is not resumable enabled, the GET_TIMEOUT function returns -1.

137.2.4 SET_SESSION_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for a session with session_id.

The new timeout setting applies to the session immediately. If session_id does not exist, no operation occurs.

Syntax

DBMS_RESUMABLE.SET_SESSION_TIMEOUT (  
    session_id  IN NUMBER,  
    timeout     IN NUMBER);
Parameters

Table 137-6  SET_SESSION_TIMEOUT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>The session identifier of the resumable space allocation.</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout of the resumable space allocation.</td>
</tr>
</tbody>
</table>

137.2.5 SET_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for the current session. The new timeout setting applies to the session immediately.

Syntax

```
DBMS_RESUMABLE.SET_TIMEOUT (
    timeout  IN NUMBER);
```

Parameters

Table 137-7  SET_TIMEOUT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The timeout of the resumable space allocation.</td>
</tr>
</tbody>
</table>

137.2.6 SPACE_ERROR_INFO Function

This function looks for space-related errors in the error stack.

If it cannot find a space related error, it will return FALSE. Otherwise, TRUE is returned and information about the particular object that causes the space error is returned.

Syntax

```
DBMS_RESUMABLE_SPACE_ERROR_INFO
    error_type        OUT VARCHAR2,
    object_type       OUT VARCHAR2,
    object_owner      OUT VARCHAR2,
    table_space_name  OUT VARCHAR2,
    object_name       OUT VARCHAR2,
    sub_object_name   OUT VARCHAR2)
RETURN BOOLEAN;
```
## Parameters

### Table 137-8  SPACE_ERROR_INFO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_type</td>
<td>The space error type. It will be one of the following:</td>
</tr>
<tr>
<td></td>
<td>- NO MORE SPACE</td>
</tr>
<tr>
<td></td>
<td>- MAX EXTENTS REACHED</td>
</tr>
<tr>
<td></td>
<td>- SPACE QUOTA EXCEEDED</td>
</tr>
<tr>
<td>object_type</td>
<td>The object type. It will be one of the following:</td>
</tr>
<tr>
<td></td>
<td>- TABLE</td>
</tr>
<tr>
<td></td>
<td>- INDEX</td>
</tr>
<tr>
<td></td>
<td>- CLUSTER</td>
</tr>
<tr>
<td></td>
<td>- TABLE SPACE</td>
</tr>
<tr>
<td></td>
<td>- ROLLBACK SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- UNDO SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- LOB SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- TEMP SEGMENT</td>
</tr>
<tr>
<td></td>
<td>- INDEX PARTITION</td>
</tr>
<tr>
<td></td>
<td>- TABLE PARTITION</td>
</tr>
<tr>
<td></td>
<td>- LOB PARTITION</td>
</tr>
<tr>
<td></td>
<td>- TABLE SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>- INDEX SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>- LOB SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>The type can also be <strong>NULL</strong> if it does not apply.</td>
</tr>
<tr>
<td>object_owner</td>
<td>The owner of the object. <strong>NULL</strong> if it cannot be determined.</td>
</tr>
<tr>
<td>table_space_name</td>
<td>The table space where the object resides. <strong>NULL</strong> if it cannot be determined.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of rollback segment, temp segment, table, index, or cluster.</td>
</tr>
<tr>
<td>sub_object_name</td>
<td>The partition name or sub-partition name of LOB, TABLE, or INDEX. <strong>NULL</strong> if it cannot be determined.</td>
</tr>
</tbody>
</table>
The **DBMS_RLS** package contains the fine-grained access control administrative interface, which is used to implement Virtual Private Database (VPD).

**DBMS_RLS** is available with the Enterprise Edition only.

**See Also:**

*Oracle Database Security Guide* for usage information about **DBMS_RLS**

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Operational Notes
- Rules and Limits
- Summary of DBMS_RLS Subprograms

### 138.1 DBMS_RLS Overview

The functionality to support fine-grained access control is based on dynamic predicates, where security rules are not embedded in views, but are acquired at the statement parse time, when the base table or view is referenced in a DML statement.

A dynamic predicate for a table, view, or synonym is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface. For example:

```
DBMS_RLS.ADD_POLICY (
    'hr', 'employees', 'emp_policy', 'hr', 'emp_sec', 'select', 'user_ctx', 'time');
```

Whenever the `EMPLOYEES` table, under the `HR` schema, is referenced in a query or subquery (SELECT), the server calls the `EMP_SEC` function (under the `HR` schema). This function returns a predicate specific to the current schema for the `EMP_POLICY` policy. The policy function may generate the predicates based on the session environment variables available during the function call. These variables usually appear in the form of application contexts. The policy can specify any combination of security-relevant columns and of these statement types: INDEX, SELECT, INSERT, UPDATE, or DELETE.

The server then produces a transient view with the text:

```
SELECT * FROM hr.employees WHERE P1
```
Here, $P_1$ (for example, where $SAL > 10000$, or even a subquery) is the predicate returned from the `EMP_SEC` function. The server treats the `EMPLOYEES` table as a view and does the view expansion just like the ordinary view, except that the view text is taken from the transient view instead of the data dictionary.

If the predicate contains subqueries, then the owner (definer) of the policy function is used to resolve objects within the subqueries and checks security for those objects. In other words, users who have access privilege to the policy-protected objects do not need to know anything about the policy. They do not need to be granted object privileges for any underlying security policy. Furthermore, the users do not require `EXECUTE` privilege on the policy function, because the server makes the call with the function definer's right.

---

**Note:**

The transient view can preserve the updatability of the parent object because it is derived from a single table or view with predicate only; that is, no `JOIN`, `ORDER BY`, `GROUP BY`, and so on.

---

`DBMS_RLS` also provides the interface to drop or enable security policies. For example, you can drop or enable the `EMP_POLICY` with the following PL/SQL statements:

```sql
DBMS_RLS.DROP_POLICY('hr', 'employees', 'emp_policy');
DBMS_RLS.ENABLE_POLICY('hr', 'employees', 'emp_policy', TRUE);
```

### 138.2 DBMS_RLS Security Model

A security check is performed when the transient view is created with a subquery. The schema owning the policy function, which generates the dynamic predicate, is the transient view's definer for security check and object lookup.

### 138.3 DBMS_RLS Constants

The `DBMS_RLS` package includes constants that can be used for specifying parameter values.

**Table 138-1 DBMS_RLS Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADD_ATTRIBUTE_ASSOCIATION</code></td>
<td><code>BINARY_INTEGER</code></td>
<td>1</td>
<td>Used with <code>DBMS_RLS.ALTER_POLICY</code> and <code>DBMS_RLS_ALTER_GROUPED_POLICY</code>; adds the specified namespace and attribute to the policy or grouped policy</td>
</tr>
</tbody>
</table>
Table 138-1  (Cont.) DBMS_RLS Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOVE_ATTRIBUTE_ASSOCIATION</td>
<td>BINARY_INTEGER</td>
<td>2</td>
<td>Used with DBMS_RLS.ALTER_POLICY and DBMS_RLS.ALTER_GROUPED_POLICY: removes the specified namespace and attribute to the policy or grouped policy.</td>
</tr>
</tbody>
</table>

138.4 DBMS_RLS Operational Notes

The DBMS_RLS procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS_RLS procedures are part of the DDL transaction.

For example, you may create a trigger for CREATE TABLE. Inside the trigger, you may add a column through ALTER TABLE, and you can add a policy through DBMS_RLS. All these operations are in the same transaction as CREATE TABLE, even though each one is a DDL statement. The CREATE TABLE succeeds only if the trigger is completed successfully.

Views of current cursors and corresponding predicates are available from V$VPD_POLICIES.

A synonym can reference only a view or a table.

138.5 DBMS_RLS Rules and Limits

Using long identifiers is supported for VPD. The maximum length for arguments such as object_schema, object_name, and policy_name, which apply to objects (table names, policy names, and subprogram names) and views is 128 bytes.

138.6 Summary of DBMS_RLS Subprograms

This table lists and briefly describes the subprograms available in DBMS_RLS.

Table 138-2  DBMS_RLS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_GROUPED_POLICY Procedure</td>
<td>Adds a policy associated with a policy group</td>
</tr>
<tr>
<td>ADD_POLICY Procedure</td>
<td>Adds a fine-grained access control policy to a table, view, or synonym</td>
</tr>
<tr>
<td>ADD_POLICY_CONTEXT Procedure</td>
<td>Adds the context for the active application</td>
</tr>
<tr>
<td>ALTER_POLICY Procedure</td>
<td>Associates an application context attribute with VPD policies</td>
</tr>
<tr>
<td>ALTER_GROUPED_POLICY Procedure</td>
<td>Adds application context related changes</td>
</tr>
<tr>
<td>CREATE_POLICY_GROUP Procedure</td>
<td>Creates a policy group</td>
</tr>
<tr>
<td>DELETE_POLICY_GROUP Procedure</td>
<td>Deletes a policy group</td>
</tr>
</tbody>
</table>
### Table 138-2  (Cont.) DBMS_RLS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE_GROUPED_POLICY Procedure</td>
<td>Disables a row-level group security policy</td>
</tr>
<tr>
<td>DROP_GROUPED_POLICY Procedure</td>
<td>Drops a policy associated with a policy group</td>
</tr>
<tr>
<td>DROP_POLICY Procedure</td>
<td>Drops a fine-grained access control policy from a table, view, or synonym</td>
</tr>
<tr>
<td>DROP_POLICY_CONTEXT Procedure</td>
<td>Drops a driving context from the object so that it will have one less driving context</td>
</tr>
<tr>
<td>ENABLE_GROUPED_POLICY Procedure</td>
<td>Enables or disables a row-level group security policy</td>
</tr>
<tr>
<td>ENABLE_POLICY Procedure</td>
<td>Enables or disables a fine-grained access control policy</td>
</tr>
<tr>
<td>REFRESH_GROUPED_POLICY Procedure</td>
<td>Reparses the SQL statements associated with a refreshed policy</td>
</tr>
<tr>
<td>REFRESH_POLICY Procedure</td>
<td>Causes all the cached statements associated with the policy to be reparsed</td>
</tr>
</tbody>
</table>

### 138.6.1 ADD_GROUPED_POLICY Procedure

This procedure adds a policy associated with a policy group.

#### Syntax

```sql
DBMS_RLS.ADD_GROUPED_POLICY(
    object_schema    IN  VARCHAR2        DEFAULT NULL,
    object_name      IN  VARCHAR2,
    policy_group     IN  VARCHAR2        DEFAULT 'SYS_DEFAULT',
    policy_name      IN  VARCHAR2,
    function_schema  IN  VARCHAR2        DEFAULT NULL,
    policy_function  IN  VARCHAR2,
    statement_types  IN  VARCHAR2        DEFAULT NULL,
    update_check     IN  BOOLEAN         DEFAULT FALSE,
    enable           IN  BOOLEAN         DEFAULT TRUE,
    static_policy    IN  BOOLEAN         DEFAULT FALSE,
    policy_type      IN  BINARY_INTEGER  DEFAULT NULL,
    long_predicate   IN  BOOLEAN         DEFAULT FALSE,
    sec_relevant_cols IN  VARCHAR2,
    sec_relevant_cols_opt IN  BINARY_INTEGER  DEFAULT NULL,
    namespace        IN  VARCHAR2        DEFAULT NULL,
    attribute        IN  VARCHAR2        DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is added</td>
</tr>
<tr>
<td>policy_group</td>
<td>Name of the policy group to which the policy belongs</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of the policy; must be unique for the same table or view</td>
</tr>
</tbody>
</table>
Table 138-3  (Cont.) ADD_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_schema</td>
<td>Schema owning the policy function. If no function_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>policy_function</td>
<td>Name of the function that generates a predicate for the policy. If the function is defined within a package, the name of the package must be present.</td>
</tr>
<tr>
<td>statement_types</td>
<td>Statement types to which the policy applies. It can be any combination of INDEX, SELECT, UPDATE, or DELETE. The default is to apply to all of these types except INSERT and INDEX.</td>
</tr>
<tr>
<td>update_check</td>
<td>For INSERT and UPDATE statements only, setting update_check to TRUE causes the server to check the policy against the value after INSERT or UPDATE. The check applies only to the security relevant columns that are included in the policy definition. In other words, the INSERT or UPDATE operation will fail only if the security relevant column that is defined in the policy is added or updated in the INSERT or UPDATE statement.</td>
</tr>
<tr>
<td>enable</td>
<td>Indicates if the policy is enabled when it is added. The default is TRUE.</td>
</tr>
<tr>
<td>static_policy</td>
<td>Default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privilege user who has the EXEMPT ACCESS POLICY privilege.</td>
</tr>
<tr>
<td>policy_type</td>
<td>Default is NULL, which means policy_type is decided by the value of static_policy. The available policy types are listed in Table 138-5. Specifying any of these policy types overrides the value of static_policy.</td>
</tr>
<tr>
<td>long_predicate</td>
<td>Default is FALSE, which means the policy function can return a predicate with a length of up to 4000 bytes. TRUE means the predicate text string length can be up to 32K bytes. Policies existing prior to the availability of this parameter retain a 32K limit.</td>
</tr>
<tr>
<td>sec_relevant_cols</td>
<td>Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.</td>
</tr>
<tr>
<td>namespace</td>
<td>Name which determines the application context namespace</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute which determines the application context attribute name</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure adds a policy to the specified table, view, or synonym and associates the policy with the specified policy group.
- The policy group must have been created by using the CREATE_POLICY_GROUP Procedure.
- The policy name must be unique within a policy group for a specific object.
• Policies from the default policy group, SYS_DEFAULT, are always executed regardless of the active policy group; however, fine-grained access control policies do not apply to users with EXEMPT_ACCESS_POLICY system privilege.

138.6.2 ADD_POLICY Procedure

This procedure adds a fine-grained access control policy to a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also:
Operational Notes

A COMMIT is also performed at the end of the operation.

Syntax

```sql
DBMS_RLS.ADD_POLICY (
    object_schema            IN  VARCHAR2       DEFAULT NULL,
    object_name              IN  VARCHAR2,
    policy_name              IN  VARCHAR2,
    function_schema          IN  VARCHAR2       DEFAULT NULL,
    policy_function          IN  VARCHAR2,
    statement_types          IN  VARCHAR2       DEFAULT NULL,
    update_check             IN  BOOLEAN        DEFAULT FALSE,
    enable                   IN  BOOLEAN        DEFAULT TRUE,
    static_policy            IN  BOOLEAN        DEFAULT FALSE,
    policy_type              IN  BINARY_INTEGER DEFAULT NULL,
    long_predicate           IN  BOOLEAN        DEFAULT FALSE,
    sec_relevant_cols        IN  VARCHAR2       DEFAULT NULL,
    sec_relevant_cols_opt    IN  BINARY_INTEGER DEFAULT NULL,
    namespace                IN  VARCHAR2       DEFAULT NULL,
    attribute                IN  VARCHAR2       DEFAULT NULL);
```

Parameters

Table 138-4  ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table, view, or synonym to which the policy is added.</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be added. It must be unique for the same table or view. Do not enter special characters such as spaces or commas. If you want to use special characters for the policy name, then enclose the name in quotation marks.</td>
</tr>
<tr>
<td>function_schema</td>
<td>Schema owning the policy function. If no function_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>policy_function</td>
<td>Name of a function which generates a predicate for the policy. If the function is defined within a package, then the name of the package must be present.</td>
</tr>
<tr>
<td>statement_types</td>
<td>Statement types to which the policy applies. It can be any combination of INDEX, SELECT, UPDATE, or DELETE. The default is to apply to all of these types except INSERT and INDEX.</td>
</tr>
<tr>
<td>update_check</td>
<td>Optional argument for the INSERT or UPDATE statement type. The default is FALSE. If you plan to use the INSERT statement type, then you must set update_check to TRUE. Otherwise, an ORA-28104 input value for string is not valid error is generated. The check applies only to the security relevant columns that are included in the policy definition. In other words, the INSERT or UPDATE operation will fail only if the security relevant column that is defined in the policy is added or updated in the INSERT or UPDATE statement.</td>
</tr>
<tr>
<td>enable</td>
<td>Indicates if the policy is enabled when it is added. The default is TRUE.</td>
</tr>
<tr>
<td>static_policy</td>
<td>The default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privileged user who has the EXEMPT ACCESS POLICY privilege.</td>
</tr>
<tr>
<td>policy_type</td>
<td>Default is NULL, which means policy_type is decided by the value of static_policy. The available policy types are listed in Table 138-5. Specifying any of these policy types overrides the value of static_policy.</td>
</tr>
<tr>
<td>long_predicate</td>
<td>Default is FALSE, which means the policy function can return a predicate with a length of up to 4000 bytes. TRUE means the predicate text string length can be up to 32K bytes. Policies existing prior to the availability of this parameter retain a 32K limit.</td>
</tr>
<tr>
<td>sec_relevant_cols</td>
<td>Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.</td>
</tr>
<tr>
<td>sec_relevant_cols_opt</td>
<td>Use with sec_relevant_cols to display all rows for column-level VPD filtered queries (SELECT only), but where sensitive columns appear as NULL. Default is set to NULL, which allows the filtering defined with sec_relevant_cols to take effect. Set to dbms_rls.ALL_ROWS to display all rows, but with sensitive column values, which are filtered by sec_relevant_cols, displayed as NULL. See Usage Notes for restrictions and additional information about this option.</td>
</tr>
<tr>
<td>namespace</td>
<td>Name which determines the application context namespace</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute which determines the application context attribute name</td>
</tr>
</tbody>
</table>
### Table 138-5  DBMS_RLS.ADD_POLICY Policy Types

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC</td>
<td>Predicate is assumed to be the same regardless of the runtime environment. Static policy functions are executed once and then cached in SGA. Statements accessing the same object do not reexecute the policy function. However, each execution of the same cursor could produce a different row set even for the same predicate because the predicate may filter the data differently based on attributes such as SYS_CONTEXT or SYSDATE. Applies to only one object.</td>
</tr>
<tr>
<td>SHARED_STATIC</td>
<td>Same as STATIC except that the server first looks for a cached predicate generated by the same policy function of the same policy type. Shared across multiple objects.</td>
</tr>
<tr>
<td>CONTEXT_SENSITIVE</td>
<td>Server re-evaluates the policy function at statement execution time if it detects context changes since the last use of the cursor. For session pooling where multiple clients share a database session, the middle tier must reset context during client switches. Note that the server does not cache the value returned by the function for this policy type; it always executes the policy function on statement parsing. Applies to only one object.</td>
</tr>
<tr>
<td>SHARED_CONTEXT_SENSITIVE</td>
<td>Same as CONTEXT_SENSITIVE except that the server first looks for a cached predicate generated by the same policy function of the same policy type within the same database session. If the predicate is found in the session memory, the policy function is not reexecuted and the cached value is valid until session private application context changes occur. Shared across multiple objects.</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>The default policy type. Server assumes the predicate may be affected by any system or session environment at any time, and so always re-executes the policy function upon each statement parsing and execution. Applies to only one object.</td>
</tr>
</tbody>
</table>

**Usage Notes**

- **SYS** is free of any security policy.
- The policy functions are called by the server. Following is the interface for the function:
  ```sql
  FUNCTION policy_function (object_schema IN VARCHAR2, object_name VARCHAR2) RETURN VARCHAR2
  --- object_schema is the schema owning the table or view.
  --- object_name is the name of table, view, or synonym to which the policy applies.
  ```
- The policy functions must have the purity level of **WNDS** (write no database state).

**See Also:**

The *Oracle Database Development Guide* has more details about the **RESTRICT_REFERENCES** pragma.

- Predicates generated from different VPD policies for the same object have the combined effect of a conjunction (`AND`) of all the predicates.
• The security check and object lookup are performed against the owner of the policy function for objects in the subqueries of the dynamic predicates.

• If the function returns a zero length predicate, then it is interpreted as no restriction being applied to the current user for the policy.

• When a table alias is required (for example, parent object is a type table) in the predicate, the name of the table or view itself must be used as the name of the alias. The server constructs the transient view as something like

  "select c1, c2, ... from tab tab where <predicate>"

• Validity of the function is checked at runtime for ease of installation and other dependency issues during import and export.

• Column-level VPD column masking behavior (specified with sec_relevant_cols_opt => dbms_rls.ALL_ROWS) is fundamentally different from all other VPD policies, which return only a subset of rows. Instead the column masking behavior returns all rows specified by the user's query, but the sensitive column values display as NULL. The restrictions for this option are as follows:
  – Only applies to SELECT statements
  – Unlike regular VPD predicates, the masking condition that is generated by the policy function must be a simple boolean expression.
  – If your application performs calculations, or does not expect NULL values, then you should use the default behavior of column-level VPD, which is specified with the sec_relevant_cols parameter.
  – If you use UPDATE AS SELECT with this option, then only the values in the columns you are allowed to see will be updated.
  – This option may prevent some rows from displaying. For example:

    SELECT * FROM employees
    WHERE salary = 10

    This query may not return rows if the salary column returns a NULL value because the column masking option has been set.

• When you add a VPD policy to a synonym, it causes all the dependent objects of the synonym, including policy functions that reference the synonym, to be marked INVALID.

• You cannot associate a global application context with a context sensitive policy or a context shared sensitive policy.

• The maximum number of policies that can be created for a single object is 255.

Examples

As the first of two examples, the following creates a policy that applies to the hr.employee table. This is a column-level VPD policy that will be enforced only if a SELECT or an INDEX statement refers to the salary, birthdate, or SSN columns of the table explicitly, or implicitly through a view. It is also a CONTEXT_SENSITIVE policy, so the server will invoke the policy function hr.hrfun at parse time. The namespace and attribute application context parameters restrict the policy evaluation only when the application context values change. During execution, it will only invoke the function if there has been any session private context change since the last use of the statement cursor. The predicate generated by the policy function must not exceed 4000 bytes, the default length limit, since the long_predicate parameter is omitted from the call.
BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema => 'hr',
    object_name => 'employee',
    policy_name => 'hr_policy',
    function_schema => 'hr',
    policy_function => 'hrfun',
    statement_types => 'select,index',
    policy_type => DBMS_RLS.CONTEXT_SENSITIVE,
    sec_relevant_cols => 'salary,birthdate,ssn',
    namespace => 'empno_ctx',
    attribute => 'emp_id');
END;

As the second example, the following command creates another policy that applies to
the same object for hosting, so users can access only data based on their subscriber
ID. Since it is defined as a SHARED_STATIC policy type, the server will first try to find the
predicate in the SGA cache. The server will only invoke the policy function, subfun, if
that search fails.

BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema => 'hr',
    object_name => 'employee',
    policy_name => 'hosting_policy',
    function_schema => 'hr',
    policy_function => 'subfun',
    policy_type => dbms_rls.SHARED_STATIC);  
END;

138.6.3 ADD_POLICY_CONTEXT Procedure

This procedure adds the context for the active application.

Syntax

DBMS_RLS.ADD_POLICY_CONTEXT (  
  object_schema IN VARCHAR2 NULL,
  object_name IN VARCHAR2,
  namespace IN VARCHAR2,
  attribute IN VARCHAR2);

Parameters

Table 138-6  ADD_POLICY_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is added.</td>
</tr>
<tr>
<td>namespace</td>
<td>Name which determines the application context namespace</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute which determines the application context attribute name</td>
</tr>
</tbody>
</table>

Usage Notes

Note the following:
• This procedure indicates the application context that drives the enforcement of policies; this is the context that determines which application is running.

• The driving context can be session or global.

• At execution time, the server retrieves the name of the active policy group from the value of this context.

• There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.

• Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.

• If the driving context is NULL, policies from all policy groups are used.

• If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the SYS_DEFAULT policy group.

• To add a policy to table HR.EMPLOYEES in group access_control_group, the following command is issued:

```sql
DBMS_RLS.ADD_GROUPED_POLICY('hr','employees','access_control_group','policy1','SYS','HR.ACCESS');
```

### 138.6.4 ALTER_POLICY Procedure

This procedure associates an application context attribute with VPD policies.

#### Syntax

```sql
DBMS_RLS.ALTER_POLICY (object_schema IN VARCHAR2 DEFAULT NULL, object_name IN VARCHAR2, policy_name IN VARCHAR2, alter_option IN NUMBER, namespace IN VARCHAR2 DEFAULT NULL, attribute IN VARCHAR2 DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is added</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of the policy, unique for the same table or view</td>
</tr>
<tr>
<td>alter_option</td>
<td>Used to determine whether the application context is being added or removed from an Oracle Virtual Private Database policy</td>
</tr>
<tr>
<td>namespace</td>
<td>Name which determines the application context namespace</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute which determines the application context attribute name</td>
</tr>
</tbody>
</table>

#### Usage Notes

Note the following:
This procedure associates an application context namespace and application context attribute to context sensitive and shared context sensitive policies only. Specifying application context namespace and application context attribute for DYNAMIC, STATIC or SHARED_STATIC policies will result in an error. If namespace is specified, attribute should also be specified for the procedure call.

You cannot associate a global application context with a context sensitive policy or a context shared sensitive policy.

Invocations of ALTER_POLICY which modify a shared context sensitive VPD policy have an effect on all shared context sensitive VPD policies that have the same VPD policy function.

The driving context can be session or global.

At execution time, the server retrieves the name of the active policy group from the value of this context.

There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.

Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.

If the driving context is NULL, policies from all policy groups are used.

If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the SYS_DEFAULT policy group.

To add a policy to table hr.employees in group access_control_group, the following command is issued:

DBMS_RLS.ADD_GROUPED_POLICY(
    'hr','employees','access_control_group','policy1','SYS', 'HR.ACCESS');

138.6.5 ALTER_GROUPED_POLICY Procedure

This procedure adds application context related changes.

Syntax

DBMS_RLS.ALTER_GROUPED_POLICY (
    object_schema    IN VARCHAR2 DEFAULT NULL,
    object_name      IN VARCHAR2,
    policy_group     IN VARCHAR2 DEFAULT SYS_DEFAULT,
    policy_name      IN VARCHAR2,
    alter_option     IN NUMBER,
    namespace        IN VARCHAR2 DEFAULT NULL,
    attribute        IN VARCHAR2 DEFAULT NULL);

Parameters

Table 138-8 ALTER_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
</tbody>
</table>
Table 138-8  (Cont.) ALTER_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is added</td>
</tr>
<tr>
<td>policy_group</td>
<td>Name of the policy group to which this policy belongs; must be unique for the same table or view</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of the policy, unique for the same table or view</td>
</tr>
<tr>
<td>alter_option</td>
<td>Used to determine whether the application context is being added or removed from the Oracle Virtual Private Database policy</td>
</tr>
<tr>
<td>namespace</td>
<td>Name that determines the application context namespace</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute determines the application context attribute name</td>
</tr>
</tbody>
</table>

Usage Notes

Note the following:

- This procedure will associate an application context namespace and application context attribute to context sensitive and shared context sensitive policies only. Specifying application context namespace and application context attribute for DYNAMIC, STATIC or SHARED_STATIC policies will result in an error. If namespace is specified, attribute should also be specified for the procedure call.
- You cannot associate a global application context with a context sensitive policy or a context shared sensitive policy.
- Invocations of ALTER_GROUPED_POLICY which modify a shared context sensitive VPD policy have an effect on all shared context sensitive VPD policies that have the same VPD policy function.
- The driving context can be session or global.
- At execution time, the server retrieves the name of the active policy group from the value of this context.
- There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.
- Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.
- If the driving context is NULL, policies from all policy groups are used.
- If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the SYS_DEFAULT policy group.
- To add a policy to table hr.employees in group access_control_group, the following command is issued:

```sql
DBMS_RLS.ADD_GROUPED_POLICY(
  'hr','employees','access_control_group','policy1','SYS', 'HR.ACCESS');
```
138.6.6 CREATE_POLICY_GROUP Procedure

This procedure creates a policy group.

Syntax

```sql
DBMS_RLS.CREATE_POLICY_GROUP (
    object_schema   IN VARCHAR2 NULL,
    object_name     IN VARCHAR2,
    policy_group    IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is added</td>
</tr>
<tr>
<td>policy_group</td>
<td>Name of the policy group that the policy belongs to</td>
</tr>
</tbody>
</table>

Usage Notes

The group must be unique for each table or view.

138.6.7 DELETE_POLICY_GROUP Procedure

This procedure deletes a policy group.

Syntax

```sql
DBMS_RLS.DELETE_POLICY_GROUP (
    object_schema   IN VARCHAR2 NULL,
    object_name     IN VARCHAR2,
    policy_group    IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is added</td>
</tr>
<tr>
<td>policy_group</td>
<td>Name of the policy group that the policy belongs to</td>
</tr>
</tbody>
</table>

Usage Notes

Note the following:
• This procedure deletes a policy group for the specified table, view, or synonym.
• No policy can be in the policy group.

138.6.8 DISABLE_GROUPED_POLICY Procedure

This procedure disables a row-level group security policy.

Syntax

```sql
DBMS_RLS.DISABLE_GROUPED_POLICY (
  object_schema   IN VARCHAR2 NULL,
  object_name     IN VARCHAR2,
  group_name      IN VARCHAR2,
  policy_name     IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym with which the policy is associated</td>
</tr>
<tr>
<td>group_name</td>
<td>Name of the group of the policy</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of the policy to be enabled or disabled</td>
</tr>
</tbody>
</table>

Usage Notes

• The procedure causes the current transaction, if any, to commit before the operation is carried out.
• A commit is performed at the end of the operation.
• A policy is disabled when this procedure is executed or when the `ENABLE_GROUPED_POLICY` procedure is executed with "enable" set to FALSE.

138.6.9 DROP_GROUPED_POLICY Procedure

This procedure drops a policy associated with a policy group.

Syntax

```sql
DBMS_RLS.DROP_GROUPED_POLICY (
  object_schema   IN VARCHAR2 NULL,
  object_name     IN VARCHAR2,
  policy_group    IN VARCHAR2 'SYS_DEFAULT',
  policy_name     IN VARCHAR2);
```
Parameters

Table 138-12  DROP_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is dropped</td>
</tr>
<tr>
<td>policy_group</td>
<td>Name of the policy group to which the policy belongs</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of the policy</td>
</tr>
</tbody>
</table>

138.6.10 DROP_POLICY Procedure

This procedure drops a fine-grained access control policy from a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also:
Operational Notes

A COMMIT is also performed at the end of the operation.

Syntax

```sql
DBMS_RLS.DROP_POLICY (  
  object_schema  IN VARCHAR2 NULL,  
  object_name   IN VARCHAR2,  
  policy_name   IN VARCHAR2);  
```

Parameters

Table 138-13  DROP_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view or synonym. If no object_schema is specified, or NULL is provided, then the current user's schema is assumed.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym for which the policy is dropped</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be dropped from table, view, or synonym</td>
</tr>
</tbody>
</table>
Usage Notes

• When you drop a VPD policy from a synonym, it causes all the dependent objects of the synonym, including policy functions that reference the synonym, to be marked INVALID.

138.6.11 DROP_POLICY_CONTEXT Procedure

This procedure drops a driving context from the object so that it will have one less driving context.

Syntax

```sql
DBMS_RLS.DROP_POLICY_CONTEXT (  
    object_schema   IN VARCHAR2 NULL,  
    object_name     IN VARCHAR2,  
    namespace       IN VARCHAR2,  
    attribute       IN VARCHAR2);
```

Parameters

**Table 138-14  DROP_POLICY_CONTEXT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym to which the policy is dropped</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace of the driving context</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute of the driving context</td>
</tr>
</tbody>
</table>

138.6.12 ENABLE_GROUPED_POLICY Procedure

This procedure enables or disables a row-level group security policy.

Syntax

```sql
DBMS_RLS.ENABLE_GROUPED_POLICY (  
    object_schema   IN VARCHAR2 NULL,  
    object_name     IN VARCHAR2,  
    group_name      IN VARCHAR2,  
    policy_name     IN VARCHAR2,  
    enable          IN BOOLEAN TRUE);
```

Parameters

**Table 138-15  ENABLE_GROUPED_POLICY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym with which the policy is associated</td>
</tr>
<tr>
<td>group_name</td>
<td>Name of the group of the policy</td>
</tr>
</tbody>
</table>
### Table 138-15  (Cont.) ENABLE_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_name</td>
<td>Name of the policy to be enabled or disabled</td>
</tr>
<tr>
<td>enable</td>
<td>TRUE enables the policy, FALSE disables the policy</td>
</tr>
</tbody>
</table>

**Usage Notes**

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is enabled when it is created.

### 138.6.13 ENABLE_POLICY Procedure

This procedure enables or disables a fine-grained access control policy. A policy is enabled when it is created.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

A **COMMIT** is also performed at the end of the operation.

#### Syntax

```sql
DBMS_RLS.ENABLE_POLICY (  
  object_schema IN VARCHAR2 NULL,  
  object_name   IN VARCHAR2,  
  policy_name   IN VARCHAR2,  
  enable        IN BOOLEAN TRUE);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table, view, or synonym with which the policy is associated</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be enabled or disabled</td>
</tr>
<tr>
<td>enable</td>
<td>TRUE to enable the policy, FALSE to disable the policy</td>
</tr>
</tbody>
</table>
138.6.14 REFRESH_GROUPED_POLICY Procedure

This procedure reparses the SQL statements associated with a refreshed policy.

Syntax

DBMS_RLS.REFRESH_GROUPED_POLICY (  
    object_schema   IN VARCHAR2 NULL,  
    object_name     IN VARCHAR2 NULL,  
    group_name      IN VARCHAR2 NULL,  
    policy_name     IN VARCHAR2 NULL);  

Parameters

Table 138-17  REFRESH_GROUPED_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the table, view, or synonym with which the policy is associated</td>
</tr>
<tr>
<td>group_name</td>
<td>Name of the group of the policy</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of the policy</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to the policy has immediate effect after the procedure is executed.
- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- The procedure returns an error if it tries to refresh a disabled policy.
- The procedure removes the cached results of context and shared sensitive VPD policies.

138.6.15 REFRESH_POLICY Procedure

This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to this policy will have immediate effect after the procedure is executed.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

🔗 See Also:

Operational Notes
A **COMMIT** is also performed at the end of the operation.

**Syntax**

```sql
DBMS_RLS.REFRESH_POLICY ( 
    object_schema  IN VARCHAR2 NULL,  
    object_name    IN VARCHAR2 NULL,  
    policy_name    IN VARCHAR2 NULL);
```

**Parameters**

**Table 138-18  REFRESH_POLICY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of table, view, or synonym with which the policy is associated</td>
</tr>
<tr>
<td>policy_name</td>
<td>Name of policy to be refreshed</td>
</tr>
</tbody>
</table>

**Usage Notes**

- The procedure returns an error if it tries to refresh a disabled policy.
- The procedure removes the cached results of context and shared sensitive VPD policies.
The DBMS_ROLLING PL/SQL package is used to implement the Rolling Upgrade Using Active Data Guard feature, which streamlines the process of upgrading Oracle Database software in a Data Guard configuration in a rolling fashion. The Rolling Upgrade Using Active Data Guard feature requires a license for the Oracle Active Data Guard option, and can be used for database version upgrades starting with the first patchset of Oracle Database 12c.

Additionally, you can use this feature immediately for other database maintenance tasks. The database where maintenance is performed must be operating at a minimum of Oracle Database 12c Release 1 (12.1). Such maintenance tasks include:

- Adding partitioning to non-partitioned tables
- Changing BasicFiles LOBs to SecureFiles LOBs
- Changing XMLElement stored as CLOB to XMLElement stored as binary XML
- Altering tables to be OLTP-compressed

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_ROLLING Subprograms

See Also:

- Oracle Data Guard Concepts and Administration for information about using DBMS_ROLLING to perform a rolling upgrade

139.1 DBMS_ROLLING Overview

The DBMS_ROLLING PL/SQL package provides procedures that you can use to perform any change throughout a Data Guard configuration in a rolling fashion, including a rolling upgrade of the Oracle Database software. Although the focus of this document is rolling upgrade operations, the content is applicable to the deployment of any rolling changes.

All the procedures are executed at the current primary database, which eliminates the potential confusion of moving between remote databases to perform various operations related to the rolling upgrade. If necessary, all the procedures can be called again to resume the rolling upgrade after an error or interruption. (The upgrade script must still be run at the standby.)

The package also provides a procedure that allows you to return a Data Guard configuration back to its original, pre-upgrade state in the event users wish to abandon the rolling upgrade.
The actual execution of a rolling upgrade has been reduced to three steps (excluding the upgrade of the Oracle Database software itself and the on-disk setup of the new Oracle Database software). The number of steps remains the same regardless of the size of the Data Guard configuration.

Conceptually, for the purposes of the DBMS_ROLLING package, you divide your Data Guard configuration into two groups: the leading group and the trailing group. The databases in the leading group undergo the upgrade operation (or any other change that you are deploying) first. The databases in the trailing group undergo the upgrade of the Oracle Database software (or any other change that you are deploying) only after the switchover operation. This insulates them from the upgrade and gives you time to evaluate the effect of the change in the leading group databases.

Each group has a master database: the future primary database as specified in the DBMS_ROLLING.INIT_PLAN procedure is the master of the leading group, called Leading Group Master (LGM), while the original primary database is the master of the trailing group called Trailing Group Master (TGM). You can configure databases to protect the LGM and the TGM. Standbys designated to protect the LGM are referred to as Leading Group Standbys (LGS). Standbys designated to protect the TGM are referred to as Trailing Group Standbys (TGS). These terms are used throughout this documentation.

139.2 DBMS_ROLLING Security Model

The DBMS_ROLLING package is available to users who have been granted the DBA role.

139.3 Summary of DBMS_ROLLING Subprograms

This table lists and briefly describes the DBMS_ROLLING package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT_PLAN Procedure</td>
<td>Initializes a rolling operation plan with system-generated default values.</td>
</tr>
<tr>
<td>DESTROY_PLAN Procedure</td>
<td>Destroys any existing rolling operation plan, its parameters, and all resources associated with the rolling operation.</td>
</tr>
<tr>
<td>BUILD_PLAN Procedure</td>
<td>Validates plan parameters and creates or modifies a rolling operation plan.</td>
</tr>
<tr>
<td>SET_PARAMETER Procedure</td>
<td>Modifies a rolling operation parameter.</td>
</tr>
<tr>
<td>START_PLAN Procedure</td>
<td>Starts the rolling operation.</td>
</tr>
<tr>
<td>SWITCHOVER Procedure</td>
<td>Performs a switchover between the current primary database and the transient logical standby database.</td>
</tr>
<tr>
<td>FINISH_PLAN Procedure</td>
<td>Finalizes the rolling operation.</td>
</tr>
<tr>
<td>ROLLBACK_PLAN Procedure</td>
<td>Completely rolls back the rolling operation.</td>
</tr>
</tbody>
</table>
139.3.1 INIT_PLAN Procedure

This procedure initializes a rolling operation plan with system-generated default values.

Syntax

DBMS_ROLLING.INIT_PLAN (  
    future_primary   IN VARCHAR2);  

Parameters

Table 139-2  INIT_PLAN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>future_primary</td>
<td>DB_UNIQUE_NAME of the future primary (also known as the Leading Group Master (LGM))</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-45400: operation not permitted on current database
- ORA-45401: upgrade plan is already active
- ORA-45402: LOG_ARCHIVE_CONFIG must contain the DG_CONFIG attribute
- ORA-45403: database %s must be specified in DG_CONFIG
- ORA-45411: operation requires additional arguments
- ORA-65040: operation not allowed from within a pluggable database

Usage Notes

- A plan must be prepared before any parameters can be customized.

139.3.2 DESTROY_PLAN Procedure

This procedure destroys any existing upgrade plan, its parameters, and all resources associated with a rolling operation.

Syntax

DBMS_ROLLING.DESTROY_PLAN ();

Parameters

This procedure has no parameters.

Exceptions

- ORA-45422: operation requires existing plan
- ORA-65040: operation not allowed from within a pluggable database
Usage Notes

• When a rolling operation is complete, this procedure can be called to completely purge all states related to a rolling operation.

139.3.3 BUILD_PLAN Procedure

This procedure validates plan parameters and creates or modifies a rolling operation plan.

A successfully constructed plan is required in order to perform a rolling operation. This procedure must return successfully before the START_PLAN procedure can be called to start the rolling operation. Parameter changes made after a plan has been created may require calling the BUILD_PLAN procedure to modify the existing plan. The DBA_ROLLING_EVENTS view will indicate if any invocation of the SET_PARAMETER procedure requires a plan rebuild. Failure to rebuild the plan will result in an ORA-45416 error when attempting to resume the rolling operation.

Syntax

DBMS_ROLLING.BUILD_PLAN ();

Parameters

This procedure has no parameters.

Exceptions

• ORA-45400: operation not permitted on current database
• ORA-45403: database %s must be specified in the DG_CONFIG
• ORA-45414: could not connect to a remote database
• ORA-45419: DB_UNIQUE_NAME parameter must be specified
• ORA-45433: failover was detected on an unsupported database
• ORA-45434: multiple failovers of the same type detected
• ORA-65040: operation not allowed from within a pluggable database

Usage Notes

• This procedure connects to databases specified as plan parameters. These instances must be mounted or open, and must be reachable via the network.

139.3.4 SET_PARAMETER Procedure

This procedure modifies a rolling operation parameter.

Syntax

DBMS_ROLLING.SET_PARAMETER (    scope IN VARCHAR2 DEFAULT NULL,    name IN NUMBER,    value IN VARCHAR2);
Parameters

Table 139-3  SET_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope</td>
<td>Parameter scope. It can either be NULL for global parameters, or the DB_UNIQUE_NAME of a specific database for local parameters.</td>
</tr>
<tr>
<td>name</td>
<td>The DBMS_ROLLING constant for a given parameter.</td>
</tr>
<tr>
<td>value</td>
<td>New value for the parameter or NULL to revert to a default value.</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-45400: operation not permitted on current database
- ORA-45408: parameter name is unknown
- ORA-45409: parameter value is invalid or out of bounds
- ORA-45410: parameter may not be modified
- ORA-45411: operation requires additional arguments
- ORA-45412: parameter scope argument is unknown
- ORA-45413: parameter has no default value
- ORA-45414: could not connect to a remote database
- ORA-65040: operation not allowed from within a pluggable database

Usage Notes

- Changes to a parameter value may require a call to the DBMS_ROLLING.BUILD_PLAN procedure to modify the existing plan. Users should check the DBA_ROLLING_EVENTS view after setting a parameter to determine if a rebuild is necessary.
- Table 139-4 lists all the available parameters and their descriptions. The parameter names and values described in the table are all of type VARCHAR2.
- The MINVAL and MAXVAL columns in the DBA_ROLLING_PARAMETERS view identify the valid range of values for a parameter. The view does not contain any parameters until the DBMS_ROLLING.INIT_PLAN procedure has been successfully invoked.

Table 139-4  Valid Values for DBMS_ROLLING.SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Global?</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE_SESSIONS_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to enforce ACTIVE_SESSIONS_WAIT before halting the rolling upgrade. This parameter is only valid if ACTIVE_SESSIONS_WAIT is set to 1.</td>
<td>3600</td>
</tr>
<tr>
<td>ACTIVE_SESSIONS_WAIT</td>
<td>Yes</td>
<td>Whether the switchover operation will wait for active sessions to finish. If set to 1, the SWITTOVER procedure waits for active sessions to compete. If set to 0, the SWITTOVER procedure kills active sessions to expedite the switchover.</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 139-4 (Cont.) Valid Values for DBMS_ROLLING.SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Global?</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKUP_CONTROLFILE</td>
<td>Yes</td>
<td>File name of the backup control file that is created during a rolling upgrade.</td>
<td>rolling_change_backup.f</td>
</tr>
<tr>
<td>DGBROKER</td>
<td>Yes</td>
<td>Use Data Guard broker for managing apply, recovery, and log archive destinations.</td>
<td>1 if broker is enabled, 0 otherwise.</td>
</tr>
<tr>
<td>DICTIONARY_LOAD_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to enforce DICTIONARY_LOAD_WAIT before halting the rolling upgrade. This parameter is only valid if DICTIONARY_LOAD_WAIT is set to 1.</td>
<td>3600</td>
</tr>
<tr>
<td>DICTIONARY_LOAD_WAIT</td>
<td>Yes</td>
<td>Whether the instantiation of the transient logical standby will include a wait for the complete loading of the data dictionary snapshot in redo. If set to 1, then the START_PLAN procedure will not return until the dictionary has been completely loaded. If set to 0, then the START_PLAN procedure will only verify that the loading of the dictionary has started.</td>
<td>0</td>
</tr>
<tr>
<td>DICTIONARY_PLS_WAIT_TIMEOUT</td>
<td>Yes</td>
<td>The time in seconds to wait in between attempts to quiesce PL/SQL activity in order to write the data dictionary to redo.</td>
<td>300</td>
</tr>
<tr>
<td>DICTIONARY_PLS_WAIT_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to attempt to quiesce PL/SQL activity in order to write the data dictionary to redo.</td>
<td>3600</td>
</tr>
<tr>
<td>EVENT_RECORDS</td>
<td>Yes</td>
<td>The maximum number of records to permit in DBA_ROLLING_EVENTS</td>
<td>10000</td>
</tr>
<tr>
<td>FAILOVER</td>
<td>Yes</td>
<td>Automatically attempt to adjust the upgrade plan as a result of a failover event. This parameter resets its value to 0 upon completion of a subsequent call to BUILD_PLAN.</td>
<td>0</td>
</tr>
<tr>
<td>GRP_PREFIX</td>
<td></td>
<td>Execution of procedures in DBMS_ROLLING results in a number of Guaranteed Restore Points (GRP) taken in various databases participating in the Data Guard configuration. All such GRPs have the same prefix in their names. You can use this parameter to override the default prefix.</td>
<td>DBMSRU</td>
</tr>
<tr>
<td>IGNORE_BUILD_WARNINGS</td>
<td>Yes</td>
<td>Ignore warnings which would otherwise raise exceptions during execution of the BUILD_PLAN procedure.</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_LAST_ERROR</td>
<td>Yes</td>
<td>Ignore last encountered error upon startup of next rolling operation. This parameter resets its value to 0 upon invocation of a procedure call which resumes the rolling upgrade.</td>
<td>0</td>
</tr>
<tr>
<td>LAD_ENABLED_TIMEOUT</td>
<td>Yes</td>
<td>The maximum time in seconds to wait for a recently enabled log archive destination to reach a VALID state.</td>
<td>600</td>
</tr>
<tr>
<td>LOG_LEVEL</td>
<td>Yes</td>
<td>Logging level for the DBS_ROLLING PL/SQL package. A value of INFO results in the logging of errors and relevant non-fatal warnings. A value of FULL results in the logging of all events.</td>
<td>INFO</td>
</tr>
</tbody>
</table>
Table 139-4  (Cont.) Valid Values for DBMS_ROLLING.SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Global?</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMBER</td>
<td>No</td>
<td>The upgrade group in which the specified database is a member.</td>
<td>LEADING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A value of LEADING indicates that the standby is a member of the leading</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>upgrade group. As such, it is a standby of the Leading Group Master (LGM).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The LGM is the database which is converted into the transient logical</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>standby, and which becomes the new primary after the switchover.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A value of TRAILING indicates that the standby is a member of the trailing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>upgrade group. As such, it is a standby of the Trailing Group Master (TGM).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The TGM is the original primary database.</td>
<td></td>
</tr>
<tr>
<td>READY_LGM_LAG_TIME</td>
<td>Yes</td>
<td>The apply lag time in seconds associated with the READY_LGM_LAG_WAIT</td>
<td>600</td>
</tr>
<tr>
<td>REALY_LGM_LAG_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to enforce READY_LGM_LAG_WAIT before</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>halting the rolling upgrade. This parameter is only valid if READY_LGM_LAG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WAIT is set to 1.</td>
<td></td>
</tr>
<tr>
<td>READY_LGM_LAG_WAIT</td>
<td>Yes</td>
<td>Whether the START_PLAN procedure will wait for the apply lag on the leading</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>group master to fall below READY_LGM_LAG_TIME seconds before returning control</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>back to the user. If set to 1, the wait is performed. If set to 0, the wait</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>is not performed.</td>
<td></td>
</tr>
<tr>
<td>SWITCH_LGM_LAG_TIME</td>
<td>Yes</td>
<td>The apply lag time in seconds associated with the SWITCH_LGM_LAG_WAIT</td>
<td>600</td>
</tr>
<tr>
<td>SWITCH_LGM_LAG_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to enforce SWITCH_LGM_LAG_WAIT before</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>halting the rolling upgrade. This parameter is only valid if SWITCH_LGM_LAG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WAIT is set to 1.</td>
<td></td>
</tr>
<tr>
<td>SWITCH_LGM_LAG_WAIT</td>
<td>Yes</td>
<td>Whether the SWITCHOVER procedure will wait for the apply lag on the leading</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>group master to fall below SWITCH_LGM_LAG_TIME seconds before initiating the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>switchover. If set to 1, the wait is performed. If set to 0, the wait is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>not performed.</td>
<td></td>
</tr>
<tr>
<td>SWITCH_LGS_LAG_TIME</td>
<td>Yes</td>
<td>The apply lag time in seconds associated with the SWITCH_LGS_LAG_WAIT</td>
<td>60</td>
</tr>
<tr>
<td>SWITCH_LGS_LAG_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to enforce SWITCH_LGS_LAG_WAIT before</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>halting the rolling upgrade. This parameter is only valid if SWITCH_LGS_LAG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WAIT is set to 1.</td>
<td></td>
</tr>
<tr>
<td>SWITCH_LGS_LAG_WAIT</td>
<td>Yes</td>
<td>Whether the SWITCHOVER procedure will wait for the apply lag on the leading</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>group standbys to fall below SWITCH_LGS_LAG_TIME seconds before initiating the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>switchover. If set to 1, the wait is performed. If set to 0, the wait is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>not performed.</td>
<td></td>
</tr>
<tr>
<td>UPDATED_LGS_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to enforce UPDATED_LGS_WAIT before</td>
<td>10800</td>
</tr>
<tr>
<td></td>
<td></td>
<td>halting the rolling upgrade. This parameter is only valid if UPDATED_LGS_WAIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>is set to 1.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 139-4 (Cont.) Valid Values for DBMS_ROLLING.SET_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Global?</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATED_LGS_WAIT</td>
<td>Yes</td>
<td>Whether the SWITCHOVER procedure will wait for the leading group standbys to complete recovery of all upgrade redo before initiating the switchover. If set to 1, the wait is performed. If set to 0, the wait is not performed.</td>
<td>1</td>
</tr>
<tr>
<td>UPDATED_TGS_TIMEOUT</td>
<td>Yes</td>
<td>The maximum amount of time in seconds to enforce UPDATED_TGS_WAIT before halting the rolling upgrade. This parameter is only valid if UPDATED_TGS_WAIT is set to 1.</td>
<td>10800</td>
</tr>
<tr>
<td>UPDATED_TGS_WAIT</td>
<td>Yes</td>
<td>Whether the FINISH_PLAN procedure will wait for the trailing group standbys to complete recovery of all upgrade redo before returning control to the user. If set to 1, the wait is performed. If set to 0, the wait is not performed.</td>
<td>1</td>
</tr>
</tbody>
</table>

### 139.3.5 START_PLAN Procedure

This procedure starts the rolling operation. This procedure must be executed on the primary database to formally start the rolling operation.

When the START_PLAN procedure is complete, the LGM (identified with the future_primary parameter in the INIT_PLAN procedure) will be converted into a fully configured transient logical standby database.

**Syntax**

```sql
DBMS_ROLLING.START_PLAN ();
```

**Parameters**

This procedure has no parameters.

**Exceptions**

- ORA-45400: operation not permitted on current database
- ORA-45414: could not connect to a remote database
- ORA-45415: instruction execution failure
- ORA-45416: operation cannot start until plan rebuild
- ORA-45417: operation not permitted since current phase was not %s
- ORA-45422: operation requires existing plan
- ORA-45426: managed recovery process was not running
- ORA-45427: logical standby Redo Apply process was not running
- ORA-45428: database was not in expected database role
- ORA-45435: managed recovery process was running
- ORA-45436: logical standby Redo Apply process was running
- ORA-45438: database is not in mounted mode
• ORA-45439: database is not in open read/write mode
• ORA-45486: database update progress is inconsistent
• ORA-65040: operation not allowed from within a pluggable database

Usage Notes
• A rolling operation plan must have previously been generated through the `BUILD_PLAN` procedure.

139.3.6 SWITCHOVER Procedure

This procedure performs a switchover between the current primary database (also known as the TGM) and the transient logical standby database (also known as the LGM).

At the successful completion of the procedure, the LGM assumes the primary role for the Data Guard configuration.

Syntax

```sql
DBMS_ROLLING.SWITCHOVER ();
```

Parameters

This procedure has no parameters.

Exceptions

• ORA-45400: operation not permitted on current database
• ORA-45414: could not connect to a remote database
• ORA-45415: instruction execution failure
• ORA-45416: operation cannot start until plan rebuild
• ORA-45417: operation not permitted since current phase was not %s
• ORA-45422: operation requires existing plan
• ORA-45426: managed recovery process was not running
• ORA-45427: logical standby Redo Apply process was not running
• ORA-45428: database was not in expected database role
• ORA-45435: managed recovery process was running
• ORA-45436: logical standby Redo Apply process was running
• ORA-45438: database is not in mounted mode
• ORA-45439: database is not in open read/write mode
• ORA-45486: database update progress is inconsistent
• ORA-65040: operation not allowed from within a pluggable database

Usage Notes

• This procedure can only be called after you have manually upgraded the transient logical standby and opened it on the higher Oracle Database version.
139.3.7 FINISH_PLAN Procedure

This procedure finalizes the rolling operation.

It configures the former primary (also known as the TGM) as a physical standby, and configures remaining physical standbys to recover the upgrade redo from the future primary.

Syntax

DBMS_ROLLING.FINISH_PLAN ();

Parameters

This procedure has no parameters.

Exceptions

- ORA-45400: operation not permitted on current database
- ORA-45414: could not connect to a remote database
- ORA-45415: instruction execution failure
- ORA-45416: operation cannot start until plan rebuild
- ORA-45417: operation not permitted since current phase was not %s
- ORA-45422: operation requires existing plan
- ORA-45426: managed recovery process was not running
- ORA-45427: logical standby Redo Apply process was not running
- ORA-45428: database was not in expected database role
- ORA-45435: managed recovery process was running
- ORA-45436: logical standby Redo Apply process was running
- ORA-45438: database is not in mounted mode
- ORA-45439: database is not in open read/write mode
- ORA-45486: database update progress is inconsistent
- ORA-65040: operation not allowed from within a pluggable database

Usage Notes

- This procedure can only be called after you have remounted the former primary and remaining physical standbys on the higher Oracle Database version.

139.3.8 ROLLBACK_PLAN Procedure

This procedure rolls back the configuration-wide rolling operation.

Once completed, all of the databases in the leading group become physical standbys of the original primary database. This procedure can only be called if the configuration has not yet gone through a switchover operation since the START_PLAN procedure was invoked.
Syntax

DBMS_ROLLING.ROLLBACK_PLAN;

Parameters

This procedure has no parameters.

Exceptions

• ORA-45400: operation not permitted on current database
• ORA-45414: could not connect to a remote database
• ORA-45415: instruction execution failure
• ORA-45441: no databases eligible for rollback
• ORA-45442: rollback is not permitted after a role change
• ORA-65040: operation not allowed from within a pluggable database

Usage Notes

• You must manually restart media recovery on the lower Oracle Database version if the upgrade of the transient logical standby has already been performed.
140

DBMS_ROWID

The DBMS_ROWID package lets you create ROWIDs and obtain information about ROWIDs from PL/SQL programs and SQL statements. You can find the data block number, the object number, and other ROWID components without writing code to interpret the base-64 character external ROWID. DBMS_ROWID is intended for upgrading from Oracle database version 7 to Oracle database version 8.X.

Note:

DBMS_ROWID is not to be used with universal ROWIDs (UROWIDs).

This chapter contains the following topics:

• Security Model
• Types
• Exceptions
• Operational Notes
• Examples
• Summary of DBMS_ROWID Subprograms

140.1 DBMS_ROWID Security Model

This package runs with the privileges of calling user, rather than the package owner SYS.

140.2 DBMS_ROWID Types

There are four DBMS_ROWID types.

These are:

• Extension and restriction types
• Verification types
• Object types
• Conversion types

Extension and Restriction Type

The types are as follows:

• RESTRICTED—restricted ROWID
• EXTENDED—extended ROWID
For example:

```plaintext
rowid_type_restricted constant integer := 0;
rowid_type_extended constant integer := 1;
```

**Note:**
Extended ROWIDs are only used in Oracle database version 8.X and higher.

### Verification Types

**Table 140-1 Verification Types**

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALID</td>
<td>Valid ROWID</td>
</tr>
<tr>
<td>INVALID</td>
<td>Invalid ROWID</td>
</tr>
</tbody>
</table>

For example:

```plaintext
rowid_is_valid constant integer := 0;
rowid_is_invalid constant integer := 1;
```

### Object Types

**Table 140-2 Object Types**

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDEFINED</td>
<td>Object Number not defined (for restricted ROWIDs)</td>
</tr>
</tbody>
</table>

For example:

```plaintext
rowid_object_undefined constant integer := 0;
```

### Conversion Types

**Table 140-3 Conversion Types**

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNAL</td>
<td>Convert to/from column of ROWID type</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>Convert to/from string format</td>
</tr>
</tbody>
</table>

For example:

```plaintext
rowid_convert_internal constant integer := 0;
rowid_convert_external constant integer := 1;
```

### 140.3 DBMS_ROWID Exceptions

This table describes the Exceptions raised by DBMS_ROWID subprograms.
### Table 140-4 Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWID_INVALID</td>
<td>Invalid rowid format</td>
</tr>
<tr>
<td>ROWID_BAD_BLOCK</td>
<td>Block is beyond end of file</td>
</tr>
</tbody>
</table>

For example:

ROWID_INVALID exception;  
pragma exception_init(ROWID_INVALID, -1410);

ROWID_BAD_BLOCK exception;  
pragma exception_init(ROWID_BAD_BLOCK, -28516);

### 140.4 DBMS_ROWID Operational Notes

These operation notes apply to DBMS_ROWID.

- Some of the functions in this package take a single parameter, such as a ROWID. This can be a character or a PL/SQL ROWID, either restricted or extended, as required.
- You can call the DBMS_ROWID functions and procedures from PL/SQL code, and you can also use the functions in SQL statements.

**Note:**

ROWID_INFO is a procedure. It can only be used in PL/SQL code.

- You can use functions from the DBMS_ROWID package just like built-in SQL functions; in other words, you can use them wherever you can use an expression. In this example, the ROWID_BLOCK_NUMBER function is used to return just the block number of a single row in the EMP table:

  ```sql
  SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) 
  FROM emp 
  WHERE ename = 'KING';
  ```

- If Oracle returns the error "ORA:452, 0, 'Subprogram '%s' violates its associated pragma' for pragma restrict_references, it could mean the violation is due to:
  - A problem with the current procedure or function
  - Calling a procedure or function without a pragma or due to calling one with a less restrictive pragma
  - Calling a package procedure or function that touches the initialization code in a package or that sets the default values
140.5 DBMS_ROWID Examples

This example returns the ROWID for a row in the EMP table, extracts the data object number from the ROWID, using the ROWID_OBJECT function in the DBMS_ROWID package, then displays the object number:

```sql
DECLARE
  object_no   INTEGER;
  row_id      ROWID;
  ...
BEGIN
  SELECT ROWID INTO row_id FROM emp
    WHERE empno = 7499;
  object_no := DBMS_ROWID.ROWID_OBJECT(row_id);
  DBMS_OUTPUT.PUT_LINE('The obj. # is '|| object_no);
  ...
```

140.6 Summary of DBMS_ROWID Subprograms

This table lists the DBMS_ROWID subprograms and briefly describes them.

**Table 140-5  DBMS_ROWID Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWID_BLOCK_NUMBER Function</td>
<td>Returns the block number of a ROWID</td>
</tr>
<tr>
<td>ROWID_CREATE Function</td>
<td>Creates a ROWID, for testing only</td>
</tr>
<tr>
<td>ROWID_INFO Procedure</td>
<td>Returns the type and components of a ROWID</td>
</tr>
<tr>
<td>ROWID_OBJECT Function</td>
<td>Returns the object number of the extended ROWID</td>
</tr>
<tr>
<td>ROWID_RELATIVE_FNO Function</td>
<td>Returns the file number of a ROWID</td>
</tr>
<tr>
<td>ROWID_ROW_NUMBER Function</td>
<td>Returns the row number</td>
</tr>
<tr>
<td>ROWID_TO_ABSOLUTE_FNO Function</td>
<td>Returns the absolute file number associated with the ROWID for a row in a specific table</td>
</tr>
<tr>
<td>ROWID_TO_EXTENDED Function</td>
<td>Converts a ROWID from restricted format to extended</td>
</tr>
<tr>
<td>ROWID_TO_RESTRICTED Function</td>
<td>Converts an extended ROWID to restricted format</td>
</tr>
<tr>
<td>ROWID_TYPE Function</td>
<td>Returns the ROWID type: 0 is restricted, 1 is extended</td>
</tr>
<tr>
<td>ROWID_VERIFY Function</td>
<td>Checks if a ROWID can be correctly extended by the ROWID_TO_EXTENDED function</td>
</tr>
</tbody>
</table>
140.6.1 ROWID_BLOCK_NUMBER Function

This function returns the database block number for the input ROWID.

Syntax

```sql
DBMS_ROWID.ROWID_BLOCK_NUMBER (    row_id      IN   ROWID,
    ts_type_in  IN   VARCHAR2 DEFAULT 'SMALLFILE')
RETURN NUMBER;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_block_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

**Table 140-6  ROWID_BLOCK_NUMBER Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted</td>
</tr>
<tr>
<td>ts_type_in</td>
<td>The type of the tablespace (bigfile/smallfile) to which the row belongs</td>
</tr>
</tbody>
</table>

Examples

The example SQL statement selects the block number from a ROWID and inserts it into another table:

```sql
INSERT INTO T2 (SELECT dbms_rowid.rowid_block_number(ROWID, 'BIGFILE')
FROM some_table
WHERE key_value = 42);
```

140.6.2 ROWID_CREATE Function

This function lets you create a ROWID, given the component parts as parameters.

This is useful for testing ROWID operations, because only the Oracle Server can create a valid ROWID that points to data in a database.

Syntax

```sql
DBMS_ROWID.ROWID_CREATE (    rowid_type    IN NUMBER,
    object_number IN NUMBER,
    relative_fno   IN NUMBER,
    block_number   IN NUMBER,
    row_number     IN NUMBER)
RETURN ROWID;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_create,WNDS,RNDS,WNPS,RNPS);
```
### Parameters

#### Table 140-7  ROWID_CREATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowid_type</td>
<td>Type (restricted or extended)</td>
</tr>
<tr>
<td></td>
<td>Set the rowid_type parameter to 0 for a restricted ROWID.  Set it to 1 to</td>
</tr>
<tr>
<td></td>
<td>create an extended ROWID.</td>
</tr>
<tr>
<td></td>
<td>If you specify rowid_type as 0, then the required object_number parameter</td>
</tr>
<tr>
<td></td>
<td>is ignored, and ROWID_CREATE returns a restricted ROWID.</td>
</tr>
<tr>
<td>object_number</td>
<td>Data object number (rowid_object_undefined for restricted)</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Relative file number</td>
</tr>
<tr>
<td>block_number</td>
<td>Block number in this file</td>
</tr>
<tr>
<td>row_number</td>
<td>Returns row number in this block</td>
</tr>
</tbody>
</table>

#### Examples

Create a dummy extended ROWID:

```sql
my_rowid := DBMS_ROWID.ROWID_CREATE(1, 9999, 12, 1000, 13);
```

Find out what the rowid_object function returns:

```sql
obj_number := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

The variable `obj_number` now contains 9999.

### 140.6.3 ROWID_INFO Procedure

This procedure returns information about a ROWID, including its type (restricted or extended), and the components of the ROWID.

This is a procedure, and it cannot be used in a SQL statement.

#### Syntax

```sql
DBMS_ROWID.ROWID_INFO (  
    rowid_in         IN   ROWID,  
    rowid_type       OUT  NUMBER,  
    object_number    OUT  NUMBER,  
    relative_fno     OUT  NUMBER,  
    block_number     OUT  NUMBER,  
    row_number       OUT  NUMBER);
```

#### Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_info,WNDS,RNDS,WNPS,RNPS);
```
### Parameters

#### Table 140-8  ROWID_INFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowid_in</td>
<td>ROWID to be interpreted. This determines if the ROWID is a restricted (0) or extended (1) ROWID.</td>
</tr>
<tr>
<td>rowid_type</td>
<td>Returns type (restricted/extended)</td>
</tr>
<tr>
<td>object_number</td>
<td>Returns data object number (rowid_object_undefined for restricted)</td>
</tr>
<tr>
<td>relative_fno</td>
<td>Returns relative file number</td>
</tr>
<tr>
<td>block_number</td>
<td>Returns block number in this file</td>
</tr>
<tr>
<td>row_number</td>
<td>Returns row number in this block</td>
</tr>
</tbody>
</table>

#### See Also:

"ROWID_TYPE Function"

### Examples

This example reads back the values for the ROWID that you created in the ROWID_CREATE:

```sql
DBMS_ROWID.ROWID_INFO (my_rowid, rid_type, obj_num, file_num, block_num, row_num, 'BIGFILE');
```

### 140.6.4 ROWID_OBJECT Function

This function returns the data object number for an extended ROWID.

The function returns zero if the input ROWID is a restricted ROWID.

#### Syntax

```sql
DBMS_ROWID.ROWID_OBJECT (rowid_id IN ROWID) RETURN NUMBER;
```

#### Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_object, WNDS, RNDS, WNPS, RNPS);
```

#### Parameters

#### Table 140-9  ROWID_OBJECT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted</td>
</tr>
</tbody>
</table>
The ROWID_OBJECT_UNDEFINED constant is returned for restricted ROWIDs.

Examples

SELECT dbms_rowid.rowid_object(ROWID)
FROM emp
WHERE empno = 7499;

140.6.5 ROWID_RELATIVE_FNO Function

This function returns the relative file number of the ROWID specified as the IN parameter. (The file number is relative to the tablespace.)

Syntax

DBMS_ROWID.ROWID_RELATIVE_FNO (
    rowid_id      IN   ROWID,
    ts_type_in    IN   VARCHAR2 DEFAULT 'SMALLFILE')
RETURN NUMBER;

Pragmas

pragma RESTRICT_REFERENCES(rowid_relative_fno,WNDS,RNDS,WNPS,RNPS);

Parameters

Table 140-10  ROWID_RELATIVE_FNO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted</td>
</tr>
<tr>
<td>ts_type_in</td>
<td>Type of the tablespace (bigfile/smallfile) to which the row belongs</td>
</tr>
</tbody>
</table>

Examples

The example PL/SQL code fragment returns the relative file number:

DECLARE
    file_number INTEGER;
    rowid_val    ROWID;
BEGIN
    SELECT ROWID INTO rowid_val
    FROM dept
    WHERE loc = 'Boston';
    file_number :=
        dbms_rowid.rowid_relative_fno(rowid_val, 'SMALLFILE');
    ...

140.6.6 ROWID_ROW_NUMBER Function

This function extracts the row number from the ROWID IN parameter.

Syntax

```sql
DBMS_ROWID.ROWID_ROW_NUMBER
    (row_id IN ROWID)
RETURN NUMBER;
```

Pragmas

```sql
PRAGMA RESTRICT_REFERENCES(rowid_row_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 140-11 ROWID_ROW_NUMBER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted.</td>
</tr>
</tbody>
</table>

Examples

Select a row number:

```sql
SELECT dbms_rowid.rowid_row_number(ROWID)
FROM emp
WHERE ename = 'ALLEN';
```

140.6.7 ROWID_TO_ABSOLUTE_FNO Function

This function extracts the absolute file number from a ROWID, where the file number is absolute for a row in a given schema and table.

The schema name and the name of the schema object (such as a table name) are provided as IN parameters for this function.

Syntax

```sql
DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO
    (row_id IN ROWID,
    schema_name IN VARCHAR2,
    object_name IN VARCHAR2)
RETURN NUMBER;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(rowid_to_absolute_fno,WNDS,WNPS,RNPS);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of the schema which contains the table</td>
</tr>
<tr>
<td>object_name</td>
<td>Table name</td>
</tr>
</tbody>
</table>

Examples

```
DECLARE
    abs_fno   INTEGER;
    rowid_val CHAR(18);
    object_name VARCHAR2(20) := 'EMP';
BEGIN
    SELECT ROWID INTO rowid_val
    FROM emp
    WHERE empno = 9999;
    abs_fno := dbms_rowid.rowid_to_absolute_fno(
        rowid_val, 'SCOTT', object_name);
```

**Note:**
For partitioned objects, the name must be a table name, not a partition or a sub/partition name.

140.6.8 ROWID_TO_EXTENDED Function

This function translates a restricted ROWID that addresses a row in a schema and table that you specify to the extended ROWID format.

Later, it may be removed from this package into a different place.

**Syntax**

```
DBMS_ROWID.ROWID_TO_EXTENDED (  
    old_rowid   IN ROWID,  
    schema_name IN VARCHAR2,  
    object_name IN VARCHAR2,  
    conversion_type IN INTEGER)
RETURN ROWID;
```

**Pragmas**

```
pragma RESTRICT_REFERENCES(rowid_to_extended,WNDS,WNPS,RNPS);
```
Parameters

Table 140-13  ROWID_TO_EXTENDED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_rowid</td>
<td>ROWID to be converted</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of the schema which contains the table (optional)</td>
</tr>
<tr>
<td>object_name</td>
<td>Table name (optional).</td>
</tr>
<tr>
<td>conversion_type</td>
<td>The following constants are defined:</td>
</tr>
<tr>
<td></td>
<td>ROWID_CONVERT_INTERNAL (:=0)</td>
</tr>
<tr>
<td></td>
<td>ROWID_CONVERT_EXTERNAL (:=1)</td>
</tr>
</tbody>
</table>

Return Values

ROWID_TO_EXTENDED returns the ROWID in the extended character format. If the input ROWID is NULL, then the function returns NULL. If a zero-valued ROWID is supplied (00000000.0000.0000), then a zero-valued restricted ROWID is returned.

Examples

Assume that there is a table called RIDS in the schema SCOTT, and that the table contains a column ROWID_COL that holds ROWIDs (restricted), and a column TABLE_COL that point to other tables in the SCOTT schema. You can convert the ROWIDs to extended format with the statement:

```sql
UPDATE SCOTT.RIDS
SET rowid_col =
  dbms_rowid.rowid_to_extended (
    rowid_col, 'SCOTT', TABLE_COL, 0);
```

Usage Notes

- If the schema and object names are provided as IN parameters, then this function verifies SELECT authority on the table named, and converts the restricted ROWID provided to an extended ROWID, using the data object number of the table. That ROWID_TO_EXTENDED returns a value, however, does not guarantee that the converted ROWID actually references a valid row in the table, either at the time that the function is called, or when the extended ROWID is actually used.

- If the schema and object name are not provided (are passed as NULL), then this function attempts to fetch the page specified by the restricted ROWID provided. It treats the file number stored in this ROWID as the absolute file number. This can cause problems if the file has been dropped, and its number has been reused prior to the migration. If the fetched page belongs to a valid table, then the data object number of this table is used in converting to an extended ROWID value. This is very inefficient, and Oracle recommends doing this only as a last resort, when the target table is not known. The user must still know the correct table name at the time of using the converted value.

- If an extended ROWID value is supplied, the data object number in the input extended ROWID is verified against the data object number computed from the table name parameter. If the two numbers do not match, the INVALID_ROWID exception is raised. If they do match, the input ROWID is returned.
• **ROWID_TO_EXTENDED** cannot be used with partition tables.

See Also:

The **ROWID_VERIFY Function** has a method to determine if a given **ROWID** can be converted to the extended format.

### 140.6.9 ROWID_TO_RESTRICTED Function

This function converts an extended **ROWID** into restricted **ROWID** format.

**Syntax**

```sql
DBMS_ROWID.ROWID_TO_RESTRICTED (  
    old_rowid       IN ROWID,  
    conversion_type IN INTEGER)  
RETURN ROWID;
```

**Pragmas**

```sql
pragma RESTRICT_REFERENCES(rowid_to_restricted,WNDS,RNDS,WNPS,RNPS);
```

**Parameters**

Table 140-14  **ROWID_TO_RESTRICTED Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_rowid</td>
<td><strong>ROWID</strong> to be converted</td>
</tr>
<tr>
<td>conversion_type</td>
<td>The following constants are defined:</td>
</tr>
<tr>
<td></td>
<td>ROWID_CONVERT_INTERNAL (=0)</td>
</tr>
<tr>
<td></td>
<td>ROWID_CONVERT_EXTERNAL (=1)</td>
</tr>
</tbody>
</table>

### 140.6.10 ROWID_TYPE Function

This function returns 0 if the **ROWID** is a restricted **ROWID**, and 1 if it is extended.

**Syntax**

```sql
DBMS_ROWID.ROWID_TYPE (  
    rowid_id IN ROWID)  
RETURN NUMBER;
```

**Pragmas**

```sql
pragma RESTRICT_REFERENCES(rowid_type,WNDS,RNDS,WNPS,RNPS);
```
Parameters

Table 140-15   ROWID_TYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>ROWID to be interpreted</td>
</tr>
</tbody>
</table>

Examples

IF DBMS_ROWID.ROWID_TYPE(my_rowid) = 1 THEN
    my_obj_num := DBMS_ROWID.ROWID_OBJECT(my_rowid);

140.6.11 ROWID_VERIFY Function

This function verifies the ROWID.

It returns 0 if the input restricted ROWID can be converted to extended format, given the input schema name and table name, and it returns 1 if the conversion is not possible.

Note:

You can use this function in a WHERE clause of a SQL statement, as shown in the example.

Syntax

DBMS_ROWID.ROWID_VERIFY (  
    rowid_in        IN ROWID,  
    schema_name     IN VARCHAR2,  
    object_name     IN VARCHAR2,  
    conversion_type IN INTEGER  
RETURN NUMBER;  

Pragmas

pragma RESTRICT_REFERENCES(rowid_verify,WNDS,WNPS,RNPS);

Parameters

Table 140-16   ROWID_VERIFY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowid_in</td>
<td>ROWID to be verified</td>
</tr>
<tr>
<td>schema_name</td>
<td>Name of the schema which contains the table</td>
</tr>
<tr>
<td>object_name</td>
<td>Table name</td>
</tr>
<tr>
<td>conversion_type</td>
<td>The following constants are defined:</td>
</tr>
<tr>
<td></td>
<td>ROWID_CONVERT_INTERNAL (=0)</td>
</tr>
<tr>
<td></td>
<td>ROWID_CONVERT_EXTERNAL (=1)</td>
</tr>
</tbody>
</table>
Examples

Considering the schema in the example for the `ROWID_TO_EXTENDED` function, you can use the following statement to find bad `ROWIDS` prior to conversion. This enables you to fix them beforehand.

```sql
SELECT ROWID, rowid_col
FROM SCOTT.RIDS
WHERE dbms_rowid.rowid_verify(rowid_col, NULL, NULL, 0) =1;
```

See Also:

- `UTL_RAW`, `UTL_REF`
The `DBMS_RULE` package contains subprograms that enable the evaluation of a rule set for a specified event.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Summary of DBMS_RULE Subprograms**

### 141.1 DBMS_RULE Overview

This package contains subprograms that enable the evaluation of a rule set for a specified event.

**See Also:**

- **Rule TYPES** for more information about the types used with the `DBMS_RULE` package
- **DBMS_RULE_ADM** and *Oracle Streams Concepts and Administration* for more information about this package and rules

### 141.2 DBMS_RULE Security Model

`PUBLIC` is granted `EXECUTE` privilege on this package.

**See Also:**

*Oracle Database Security Guide* for more information about user group `PUBLIC`

### 141.3 Summary of DBMS_RULE Subprograms

This table lists the `DBMS_RULE` subprograms and briefly describes them.

**Table 141-1 DBMS_RULE Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CLOSE_ITERATOR Procedure</code></td>
<td>Closes an open iterator</td>
</tr>
</tbody>
</table>
Table 141-1  (Cont.) DBMS_RULE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVALUATE Procedure</td>
<td>Evaluates the rules in the specified rule set that use the evaluation context specified</td>
</tr>
<tr>
<td>EVALUATE_EXPRESSION Procedure</td>
<td>Evaluates an expression under the logged in user in a session</td>
</tr>
<tr>
<td>EVALUATE_EXPRESSION_ITERATOR Procedure</td>
<td>Finds the relevant datapoints and pass re$value_list into evaluation interface</td>
</tr>
<tr>
<td>EVALUATE_RULE Procedure</td>
<td>Evaluates the condition defined in the Rule</td>
</tr>
<tr>
<td>EVALUATE_RULE_ITERATOR Procedure</td>
<td>Finds the relevant datapoints and pass re$value_list into evaluation interface</td>
</tr>
<tr>
<td>GET_NEXT_HIT Function</td>
<td>Returns the next rule that evaluated to TRUE from a true rules iterator, or returns the next rule that evaluated to MAYBE from a maybe rules iterator; returns NULL if there are no more rules that evaluated to TRUE or MAYBE.</td>
</tr>
<tr>
<td>IS_FAST Procedure</td>
<td>Returns TRUE if the expression can be evaluated fast. An expression can be evaluated fast if the engine does not need to run any internal SQL and does not need to go to PL/SQL layer in case there are any PL/SQL functions referred.</td>
</tr>
<tr>
<td>GET_NEXT_RESULT Function</td>
<td>Iterates over result from the expression given result_val_iterator.</td>
</tr>
</tbody>
</table>

141.3.1 CLOSE_ITERATOR Procedure

This procedure closes an open iterator.

Syntax

```
DBMS_RULE.CLOSE_ITERATOR(
    iterator  IN  BINARY_INTEGER);
```

Parameter

Table 141-2  CLOSE_ITERATOR Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterator</td>
<td>Iterator to be closed</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure requires an open iterator that was returned by an earlier call to DBMS_RULE.EVALUATE in the same session. The user who runs this procedure does not require any privileges on the rule set being evaluated.

Closing an iterator frees resources, such as memory, associated with the iterator. Therefore, Oracle recommends that you close an iterator when it is no longer needed.
141.3.2 EVALUATE Procedure

This procedure evaluates the rules in the specified rule set that use the evaluation context specified for a specified event.

This procedure is overloaded. The true_rules and maybe_rules parameters are mutually exclusive with the true_rules_iterator and maybe_rules_iterator parameters. In addition, the procedure with the true_rules and maybe_rules parameters includes the stop_on_first_hit parameter, but the other procedure does not.

Syntax

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```

```
DBMS_RULE.EVALUATE(
    rule_set_name          IN     VARCHAR2,
    evaluation_context     IN     VARCHAR2,
    event_context          IN     SYS.RE$NV_LIST               DEFAULT NULL,
    table_values           IN     SYS.RE$TABLE_VALUE_LIST      DEFAULT NULL,
    column_values          IN     SYS.RE$COLUMN_VALUE_LIST DEF
```
### Parameters

**Table 141-3  EVALUATE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_set_name</td>
<td>Name of the rule set in the form [schema_name.]rule_set_name. For example, to evaluate all of the rules in a rule set named hr_rules in the hr schema, enter hr.hr_rules for this parameter. If the schema is not specified, then the schema of the current user is used.</td>
</tr>
<tr>
<td>evaluation_context</td>
<td>An evaluation context name in the form [schema_name.]evaluation_context_name. If the schema is not specified, then the name of the current user is used. Only rules that use the specified evaluation context are evaluated.</td>
</tr>
<tr>
<td>event_context</td>
<td>A list of name-value pairs that identify events that cause evaluation.</td>
</tr>
<tr>
<td>table_values</td>
<td>Contains the data for table rows using the table aliases specified when the evaluation context was created. Each table alias in the list must be unique.</td>
</tr>
<tr>
<td>column_values</td>
<td>Contains the partial data for table rows. It must not contain column values for tables, whose values are already specified in table_values.</td>
</tr>
<tr>
<td>variable_values</td>
<td>A list containing the data for variables. The only way for an explicit variable value to be known is to specify its value in this list.itous in this list.</td>
</tr>
<tr>
<td></td>
<td>If an implicit variable value is not specified in the list, then the function used to obtain the value of the implicit variable is invoked. If an implicit variable value is specified in the list, then this value is used and the function is not invoked.</td>
</tr>
<tr>
<td>attribute_values</td>
<td>Contains the partial data for variables. It must not contain attribute values for variables whose values are already specified in variable_values.</td>
</tr>
<tr>
<td>stop_on_first_hit</td>
<td>If TRUE, then the rules engine stops evaluation as soon as it finds a TRUE rule.</td>
</tr>
<tr>
<td></td>
<td>If TRUE and there are no TRUE rules, then the rules engine stops evaluation as soon as it finds a rule that may evaluate to TRUE given more data.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the rules engine continues to evaluate rules even after it finds a TRUE rule.</td>
</tr>
<tr>
<td>simple_rules_only</td>
<td>If TRUE, then only those rules that are simple enough to be evaluated fast (without issuing SQL) are considered for evaluation.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then evaluates all rules.</td>
</tr>
</tbody>
</table>
Table 141-3  (Cont.) EVALUATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true_rules</td>
<td>Receives the output of the EVALUATE procedure into a varray of RE$RULE_HIT_LIST type.</td>
</tr>
<tr>
<td></td>
<td>If no rules evaluate to TRUE, then true_rules is empty.</td>
</tr>
<tr>
<td></td>
<td>If at least one rule evaluates to TRUE and stop_on_first_hit is TRUE, then true_rules contains one rule that evaluates to TRUE.</td>
</tr>
<tr>
<td></td>
<td>If stop_on_first_hit is FALSE, then true_rules contains all rules that evaluate to TRUE.</td>
</tr>
<tr>
<td>maybe_rules</td>
<td>If all rules can be evaluated completely, without requiring any additional data, then maybe_rules is empty.</td>
</tr>
<tr>
<td></td>
<td>If stop_on_first_hit is TRUE, then if there is at least one rule that may evaluate to TRUE given more data, and no rules evaluate to TRUE, then maybe_rules contains one rule that may evaluate to TRUE.</td>
</tr>
<tr>
<td></td>
<td>If stop_on_first_hit is FALSE, then maybe_rules contains all rules that may evaluate to TRUE given more data.</td>
</tr>
<tr>
<td>true_rules_iterator</td>
<td>Contains the iterator for accessing rules that are TRUE</td>
</tr>
<tr>
<td>maybe_rules_iterator</td>
<td>Contains the iterator for accessing rules that may be TRUE given additional data or the ability to issue SQL</td>
</tr>
<tr>
<td>skip_rules</td>
<td>List of rules to skip within this evaluation.</td>
</tr>
<tr>
<td>dop</td>
<td>Degree of parallelism</td>
</tr>
<tr>
<td>result_cache</td>
<td>If TRUE, Result Cache will be created. If evaluate procedure is called with either true_rules_iterator or maybe_rules_iterator, then result_cache is not enabled.</td>
</tr>
</tbody>
</table>

Usage Notes

**Note:**

Rules in the rule set that use an evaluation context different from the one specified are not considered for evaluation.

The rules in the rule set are evaluated using the data specified for table_values, column_values, variable_values, and attribute_values. These values must refer to tables and variables in the specified evaluation context. Otherwise, an error is raised.

The caller may specify, using stop_on_first_hit, if evaluation must stop as soon as the first TRUE rule or the first MAYBE rule (if there are no TRUE rules) is found.

The caller may also specify, using simple_rules_only, if only rules that are simple enough to be evaluated fast (which means without SQL) should be considered for evaluation. This makes evaluation faster, but causes rules that cannot be evaluated without SQL to be returned as MAYBE rules.
Partial evaluation is supported. The `EVALUATE` procedure can be called with data for only some of the tables, columns, variables, or attributes. In such a case, rules that cannot be evaluated because of a lack of data are returned as `MAYBE` rules, unless they can be determined to be `TRUE` or `FALSE` based on the values of one or more simple expressions within the rule. For example, given a value of 1 for attribute "a.b" of variable "x", a rule with the following rule condition can be returned as `TRUE`, without a value for table "tab":

```
(:x.a.b = 1) or (tab.c > 10)
```

The results of an evaluation are the following:

- **TRUE** rules, which is the list of rules that evaluate to `TRUE` based on the given data. These rules are returned either in the `OUT` parameter `true_rules`, which returns all of the rules that evaluate to `TRUE`, or in the `OUT` parameter `true_rules_iterator`, which returns each rule that evaluates to `TRUE` one at a time.

- **MAYBE** rules, which is the list of rules that could not be evaluated for one of the following reasons:
  - The rule refers to data that was unavailable. For example, a variable attribute "x.a.b" is specified, but no value is specified for the variable "x", the attribute "a", or the attribute "a.b".
  - The rule is not simple enough to be evaluated fast (without SQL) and `simple_rules_only` is specified as `TRUE`, or partial data is available.

Maybe rules are returned either in the `OUT` parameter `maybe_rules`, which returns all of the rules that evaluate to `MAYBE`, or in the `OUT` parameter `maybe_rules_iterator`, which returns each rule that evaluates to `MAYBE` one at a time.

The caller may specify whether the procedure returns all of the rules that evaluate to `TRUE` and `MAYBE` for the event or an iterator for rules that evaluate to `TRUE` and `MAYBE`. A true rules iterator enables the client to fetch each rule that evaluates to `TRUE` one at a time, and a maybe rules iterator enables the client to fetch each rule that evaluates to `MAYBE` one at a time.

If you use an iterator, then you use the `GET_NEXT_HIT` function in the `DBMS_RULE` package to retrieve the next rule that evaluates to `TRUE` or `MAYBE` from an iterator. Oracle recommends that you close an iterator if it is no longer needed to free resources, such as memory, used by the iterator. An iterator can be closed in the following ways:

- The `CLOSE_ITERATOR` procedure in the `DBMS_RULE` package is run with the iterator specified.
- The iterator returns `NULL` because no more rules evaluate to `TRUE` or `MAYBE`.
- The session in which the iterator is running ends.

To run the `DBMS_RULE.EVALUATE` procedure, a user must meet at least one of the following requirements:

- Have `EXECUTE_ON_RULE_SET` privilege on the rule set
- Have `EXECUTE_ANY_RULE_SET` system privilege
- Be the rule set owner
Note:
The rules engine does not invoke any actions. An action context can be returned with each returned rule, but the client of the rules engine must invoke any necessary actions.

See Also:
- Rule TYPES for more information about the types used with the DBMS_RULE package
- GET_NEXT_HIT Function
- CLOSE_ITERATOR Procedure

141.3.3 EVALUATE_EXPRESSION Procedure

This procedure allows user to evaluate an expression under the logged in user in a session. Any re-execute of the same expression with same table alias and variable type will result in reusing the same compiled context. With fixed compile cache size, its possible of aging....

Syntax

DBMS_RULE.EVALUATE_EXPRESSION(
  rule_expression         IN         VARCHAR2,
  table_aliases           IN         SYS.RE$TABLE_ALIAS_LIST:= NULL,
  variable_types          IN         SYS.RE$VARIABLE_TYPE_LIST:= NULL,
  table_values            IN         SYS.RE$TABLE_VALUE_LIST:= NULL,
  column_values           IN         SYS.RE$COLUMN_VALUE_LIST:=NULL,
  variable_values         IN         SYS.RE$ATTRIBUTE_VALUE_LIST:=NULL,
  cache                   IN         BOOLEAN DEFAULT FALSE,
  result_val              OUT        BOOLEAN);

Parameters

Table 141-4  EVALUATE_EXPRESSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_expression</td>
<td>Contains an expression string.</td>
</tr>
<tr>
<td>table_alias</td>
<td>Contains alias of tables referred in the expression string.</td>
</tr>
<tr>
<td>variable_types</td>
<td>Contains type definitions of variables used in expression.</td>
</tr>
<tr>
<td>table_values</td>
<td>Contains ROWID of table row for expression evaluation.</td>
</tr>
<tr>
<td>column_values</td>
<td>Contains values of columns referred in the expression.</td>
</tr>
<tr>
<td>variable_values</td>
<td>Contains values of variables referred in the expression.</td>
</tr>
<tr>
<td>attribute_values</td>
<td>Contains values of attributes referred in the expression.</td>
</tr>
<tr>
<td>cache</td>
<td>If TRUE, Result Cache will be created.</td>
</tr>
</tbody>
</table>
141.3.4 EVALUATE_EXPRESSION_ITERATOR Procedure

This is an user visible interface. Because PL/SQL based callbacks can be expensive, we provide an array based approach. The client program is assumed to find the relevant datapoints and pass \texttt{value_list} into evaluation interface. The expression evaluation engine is expected to walk through this list and evaluate expression for each datapoint (\texttt{value_list}) element.

Syntax

```sql
DBMS_RULE.EVALUATE_EXPRESSION_ITERATOR(
  rule_expression        IN    varchar2,
  table_aliases          IN    sys.реtable_alias_list:= NULL,
  variable_types         IN    sys.реvariable_type_list:= NULL,
  values                 IN    sys.реvalue_list,
  cache                  IN    boolean DEFAULT FALSE,
  result_val_iter_id     OUT   BINARY_INTEGER)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_expression</td>
<td>Contains an expression string.</td>
</tr>
<tr>
<td>table_alias</td>
<td>Alias of tables referred in the above expression string.</td>
</tr>
<tr>
<td>variable_types</td>
<td>Type definitions of variables used in expression.</td>
</tr>
<tr>
<td>values</td>
<td>List of datapoint values for evaluation.</td>
</tr>
<tr>
<td>cache</td>
<td>If TRUE, Result Cache will be created.</td>
</tr>
<tr>
<td>result_val_iter_id</td>
<td>Contains iterator for result of array of values sent using value.</td>
</tr>
</tbody>
</table>

141.3.5 EVALUATE_RULE Procedure

The Rule Evaluation API expects that \texttt{CREATE_RULE} procedure has been called with an legitimate \texttt{EVALUATION_CONTEXT} prior. This API will evaluate the condition defined in the Rule.

Syntax

```sql
DBMS_RULE.EVALUATE_RULE(
  rule_name            IN   VARCHAR2,
  event_context        IN   SYS.РесNV_LIST       DEFAULT NULL,
  table_values         IN   SYS.РесTABLE_VALUE_LIST       DEFAULT NULL,
  column_values        IN   SYS.РесCOLUMN_VALUE_LIST      DEFAULT NULL,
  variable_values      IN   SYS.РесVARIABLE_VALUE_LIST     DEFAULT NULL,
  attribute_values     IN   SYS.РесATTRIBUTE_VALUE_LIST    DEFAULT NULL,
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td></td>
</tr>
<tr>
<td>event_context</td>
<td></td>
</tr>
<tr>
<td>table_values</td>
<td></td>
</tr>
<tr>
<td>column_values</td>
<td></td>
</tr>
<tr>
<td>variable_values</td>
<td></td>
</tr>
<tr>
<td>attribute_values</td>
<td></td>
</tr>
</tbody>
</table>
Parameters

Table 141-6  EVALUATE_RULE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>Name of the rule previously create using CREATE_RULE procedure.</td>
</tr>
<tr>
<td>event_context</td>
<td>A list of name-value pairs that identify events that cause evaluation.</td>
</tr>
<tr>
<td>table_values</td>
<td>ROWID of table row for expression evaluation.</td>
</tr>
<tr>
<td>column_values</td>
<td>Values of columns referred in the expression</td>
</tr>
<tr>
<td>variable_values</td>
<td>Values of variables referred in expression</td>
</tr>
<tr>
<td>attribute_values</td>
<td>Values of attributes referred in expression</td>
</tr>
<tr>
<td>cache</td>
<td>If TRUE, Result Cache will be created.</td>
</tr>
<tr>
<td>result_val</td>
<td>Result of the evaluation</td>
</tr>
</tbody>
</table>

141.3.6 EVALUATE_RULE_ITERATOR Procedure

This is an iterative interface. The client program is assumed to find the relevant datapoints and pass \$value_list into evaluation interface.

Evaluation engine is expected to walk through this list and evaluate expression for each datapoint \(\$value_list\) element. User can use DBMS_RULE.GET_NEXT_RESULT procedure to iterate through the result list.

Syntax

```
DBMS_RULE.EVALUATE_RULE_ITERATOR(
    rule_name            IN       VARCHAR2,
    event_context        IN       SYS.RE\$NV_LIST DEFAULT NULL,
    values               IN       SYS.RE\$VALUE_LIST,
    cache                IN       BOOLEAN DEFAULT FALSE,
    result_val_iter_id   OUT      BINARY_INTEGER);
```

Parameters

Table 141-7  EVALUATE_RULE_ITERATOR Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>Name of the rule previously create using CREATE_RULE procedure.</td>
</tr>
<tr>
<td>event_context</td>
<td>A list of name-value pairs that identify events that cause evaluation.</td>
</tr>
<tr>
<td>values</td>
<td>List of datapoint values for evaluation.</td>
</tr>
<tr>
<td>cache</td>
<td>If TRUE, Result Cache will be created.</td>
</tr>
<tr>
<td>result_val_iter_id</td>
<td>Contains iterator for result of array of values sent using values</td>
</tr>
</tbody>
</table>
141.3.7 GET_NEXT_HIT Function

This function returns the next rule that evaluated to `TRUE` from a true rules iterator, or returns the next rule that evaluated to `MAYBE` from a maybe rules iterator. The function returns `NULL` if there are no more rules that evaluated to `TRUE` or `MAYBE`.

Syntax

```sql
DBMS_RULE.GET_NEXT_HIT(
    iterator  IN  BINARY_INTEGER)
RETURN SYS.RE$RULE_HIT;
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterator</td>
<td>The iterator from which the rule that evaluated to <code>TRUE</code> or <code>MAYBE</code> is retrieved</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure requires an open iterator that was returned by an earlier call to `DBMS_RULE.EVALUATE` in the same session. The user who runs this procedure does not require any privileges on the rule set being evaluated.

When an iterator returns `NULL`, it is closed automatically. If an open iterator is no longer needed, then use the `CLOSE_ITERATOR` procedure in the `DBMS_RULE` package to close it.

Note:

This function raises an error if the rule set being evaluated was modified after the call to the `DBMS_RULE.EVALUATE` procedure that returned the iterator. Modifications to a rule set include added rules to the rule set, changing existing rules in the rule set, dropping rules from the rule set, and dropping the rule set.

See Also:

- Rule TYPES for more information about the types used with the `DBMS_RULE` package
- EVALUATE Procedure
- CLOSE_ITERATOR Procedure
141.3.8 GET_NEXT_RESULT Function

This function iterates over result from the expression given in RESULT_VAL_ITERATOR. It returns the expression at iterator evaluated to TRUE or FALSE.

Syntax

```sql
DBMS_RULE.GET_NEXT_RESULT (result_val_iterator_id IN BINARY_INTEGER) RETURN BOOLEAN;
```

Parameter

Table 141-9  GET_NEXT_RESULT Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result_val_iterator_id</td>
<td>Iterator returned from EVALUATE_EXPRESSION_ITERATOR</td>
</tr>
</tbody>
</table>

141.3.9 IS_FAST Procedure

Given an expression, of either rule or Independent Expression, this procedure will return TRUE if the expression can be evaluated as fast. An expression can be evaluated as fast if the engine does not need to run any internal SQL and does not need to go to PL/SQL layer in case there are any PL/SQL functions referred.

Syntax

```sql
DBMS_RULE.IS_FAST (expression IN VARCHAR2,
                   table_aliases IN SYS.RESTABLE_ALIAS_LIST:= NULL,
                   variable_types IN SYS.RESVARIABLE_TYPE_LIST:= NULL,
                   result_val OUT BOOLEAN);
```

Parameter

Table 141-10  IS_FAST Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Expression to check</td>
</tr>
<tr>
<td>table_aliases</td>
<td>Alias of tables referred in the above expression string</td>
</tr>
<tr>
<td>variable_types</td>
<td>Type definitions of variables used in expression</td>
</tr>
<tr>
<td>result_val</td>
<td>If the expression can be evaluated as fast</td>
</tr>
</tbody>
</table>
142

DBMS_RULE_ADM

The DBMS_RULE_ADM package provides the subprograms for creating and managing rules, rule sets, and rule evaluation contexts.

This chapter contains the following topics:

• Overview
• Security Model
• Summary of DBMS_RULE_ADM Subprograms

142.1 DBMS_RULE_ADM Overview

This package provides the subprograms for creating and managing rules, rule sets, and rule evaluation contexts.

See Also:
• Rule TYPEs for more information about the types used with the DBMS_RULE_ADM package
• DBMS_RULE and Oracle Streams Concepts and Administration for more information about this package and rules

142.2 DBMS_RULE_ADM Security Model

PUBLIC is granted EXECUTE privilege on this package.

See Also:
Oracle Database Security Guide for more information about user group PUBLIC

142.3 Summary of DBMS_RULE_ADM Subprograms

This table lists the DBMS_RULE_ADM subprograms and briefly describes them.

Table 142-1   DBMS_RULE_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_RULE Procedure</td>
<td>Adds the specified rule to the specified rule set</td>
</tr>
</tbody>
</table>
Table 142-1  (Cont.) DBMS_RULE_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_EVALUATION_CONTEXT Procedure</td>
<td>Alters a rule evaluation context</td>
</tr>
<tr>
<td>ALTER_RULE Procedure</td>
<td>Changes one or more aspects of the specified rule</td>
</tr>
<tr>
<td>CREATE_EVALUATION_CONTEXT Procedure</td>
<td>Creates a rule evaluation context</td>
</tr>
<tr>
<td>CREATE_RULE Procedure</td>
<td>Creates a rule with the specified name</td>
</tr>
<tr>
<td>CREATE_RULE_SET Procedure</td>
<td>Creates a rule set with the specified name</td>
</tr>
<tr>
<td>DROP_EVALUATION_CONTEXT Procedure</td>
<td>Drops the rule evaluation context with the specified name</td>
</tr>
<tr>
<td>DROP_RULE Procedure</td>
<td>Drops the rule with the specified name</td>
</tr>
<tr>
<td>DROP_RULE_SET Procedure</td>
<td>Drops the rule set with the specified name</td>
</tr>
<tr>
<td>GRANT_OBJECT_PRIVILEGE Procedure</td>
<td>Grants the specified object privilege on the specified object to the specified user or role</td>
</tr>
<tr>
<td>GRANT_SYSTEM_PRIVILEGE Procedure</td>
<td>Grants the specified system privilege to the specified user or role</td>
</tr>
<tr>
<td>REMOVE_RULE Procedure</td>
<td>Removes the specified rule from the specified rule set</td>
</tr>
<tr>
<td>REVOKE_OBJECT_PRIVILEGE Procedure</td>
<td>Revokes the specified object privilege on the specified object from the specified user or role</td>
</tr>
<tr>
<td>REVOKE_SYSTEM_PRIVILEGE Procedure</td>
<td>Revokes the specified system privilege from the specified user or role</td>
</tr>
</tbody>
</table>

Note: All subprograms commit unless specified otherwise.

142.3.1 ADD_RULE Procedure

This procedure adds the specified rule to the specified rule set.

Syntax

```
DBMS_RULE_ADM.ADD_RULE(
    rule_name IN VARCHAR2,
    rule_set_name IN VARCHAR2,
    evaluation_context IN VARCHAR2 DEFAULT NULL,
    rule_comment IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 142-2  ADD_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule you are adding to the rule set, specified as [schema_name.]rule_name. For example, to add a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>rule_set_name</td>
<td>The name of the rule set to which you are adding the rule, specified as [schema_name.]rule_set_name. For example, to add the rule to a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>evaluation_context</td>
<td>An evaluation context name in the form [schema_name.]evaluation_context_name. If the schema is not specified, then the current user is the default. Only specify an evaluation context if the rule itself does not have an evaluation context and you do not want to use the rule set's evaluation context for the rule.</td>
</tr>
<tr>
<td>rule_comment</td>
<td>Optional description, which can contain the reason for adding the rule to the rule set</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE_SET privilege on the rule set
- Have ALTER_ANY_RULE_SET system privilege
- Be the owner of the rule set

Also, the rule set owner must meet at least one of the following requirements:

- Have EXECUTE_ON_RULE privilege on the rule
- Have EXECUTE_ANY_RULE system privilege
- Be the rule owner

If the rule has no evaluation context and no evaluation context is specified when you run this procedure, then the rule uses the evaluation context associated with the rule set. In such a case, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
- Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.
142.3.2 ALTER_EVALUATION_CONTEXT Procedure

This procedure alters a rule evaluation context. A rule evaluation context defines external data that can be referenced in rule conditions. The external data can either exist as variables or as table data.

Syntax

```
DBMS_RULE_ADM.ALTER_EVALUATION_CONTEXT(
    evaluation_context_name      IN  VARCHAR2,
    table_aliases                IN  SYS.RE$TABLE_ALIAS_LIST    DEFAULT NULL,
    remove_table_aliases         IN  BOOLEAN                    DEFAULT FALSE,
    variable_types               IN  SYS.RE$VARIABLE_TYPE_LIST  DEFAULT NULL,
    remove_variable_types        IN  BOOLEAN                    DEFAULT FALSE,
    evaluation_function          IN  VARCHAR2                   DEFAULT NULL,
    remove_evaluation_function   IN  BOOLEAN                    DEFAULT FALSE,
    evaluation_context_comment   IN  VARCHAR2                   DEFAULT NULL,
    remove_eval_context_comment  IN  BOOLEAN                    DEFAULT FALSE);
```

Parameters

Table 142-3  ALTER_EVALUATION_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation_context_name</td>
<td>The name of the evaluation context you are altering, specified as [schema_name.]evaluation_context_name. For example, to alter an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_aliases</td>
<td>If NULL and remove_table_aliases is FALSE, then the procedure retains the existing table aliases. If NULL and remove_table_aliases is TRUE, then the procedure removes the existing table aliases. If non-NULL, then the procedure replaces the existing table aliases for the evaluation context with the specified table aliases. Table aliases specify the tables in an evaluation context. The table aliases can be used to reference tables in rule conditions.</td>
</tr>
<tr>
<td>remove_table_aliases</td>
<td>If TRUE and table_aliases is NULL, then the procedure removes the existing table aliases for the evaluation context. If TRUE and table_aliases is non-NULL, then the procedure raises an error. If FALSE, then the procedure does not remove table aliases.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>variable_types</td>
<td>If NULL and remove_variable_types is FALSE, then the procedure retains the variable types. If NULL and remove_variable_types is TRUE, then the procedure removes the existing variable types.</td>
</tr>
<tr>
<td></td>
<td>If non-NULL, then the procedure replaces the existing variable types for the evaluation context with the specified variable types.</td>
</tr>
<tr>
<td>remove_variable_types</td>
<td>If TRUE and variable_types is NULL, then the procedure removes the existing variable types for the evaluation context. If TRUE and variable_types is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not remove the variable types.</td>
</tr>
<tr>
<td>evaluation_function</td>
<td>If NULL and remove_evaluation_function is FALSE, then the procedure retains the existing evaluation function. If NULL and remove_evaluation_function is TRUE, then the procedure removes the existing evaluation function.</td>
</tr>
<tr>
<td></td>
<td>If non-NULL, then the procedure replaces the existing evaluation function for the evaluation context with the specified evaluation function.</td>
</tr>
<tr>
<td></td>
<td>An evaluation function is an optional function that will be called to evaluate rules that use the evaluation context. It must have the same form as the DBMS_RULE.EVALUATE procedure. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>See CREATE_EVALUATION_CONTEXT Procedure for more information about evaluation functions.</td>
</tr>
<tr>
<td>remove_evaluation_function</td>
<td>If TRUE and evaluation_function is NULL, then the procedure removes the existing evaluation function for the evaluation context. If TRUE and evaluation_function is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not remove the evaluation function.</td>
</tr>
<tr>
<td>evaluation_context_comment</td>
<td>If NULL and remove_eval_context_comment is FALSE, then the procedure retains the existing evaluation context comment. If NULL and remove_evaluation_function is TRUE, then the procedure removes the existing evaluation context comment.</td>
</tr>
<tr>
<td></td>
<td>If non-NULL, then the procedure replaces the existing comment for the evaluation context with the specified comment.</td>
</tr>
<tr>
<td></td>
<td>An evaluation context comment is an optional description of the rule evaluation context.</td>
</tr>
</tbody>
</table>
Table 142-3  (Cont.) ALTER_EVALUATION_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove_eval_context_comment</td>
<td>If TRUE and evaluation_context_comment is NULL, then the procedure removes the existing comment for the evaluation context. If TRUE and evaluation_context_comment is non-NULL, then the procedure raises an error. If FALSE, then the procedure does not remove the evaluation context comment.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context being altered
- Have ALL_ON_EVALUATION_CONTEXT or ALTER_ON_EVALUATION_CONTEXT object privilege on an evaluation context owned by another user
- Have ALTER_ANY_EVALUATION_CONTEXT system privilege

See Also:

Rule TYPEs for more information about the types used with the DBMS_RULE_ADM package

142.3.3 ALTER_RULE Procedure

This procedure changes one or more aspects of the specified rule.

Syntax

```sql
DBMS_RULE_ADM.ALTER_RULE ( 
  rule_name                  IN  VARCHAR2,
  condition                  IN  VARCHAR2        DEFAULT NULL,
  evaluation_context         IN  VARCHAR2        DEFAULT NULL,
  remove_evaluation_context  IN  BOOLEAN         DEFAULT FALSE,
  action_context             IN  SYS.RE$NV_LIST  DEFAULT NULL,
  remove_action_context      IN  BOOLEAN         DEFAULT FALSE,
  rule_comment               IN  VARCHAR2        DEFAULT NULL,
  remove_rule_comment        IN  BOOLEAN         DEFAULT FALSE);
```
Parameters

Table 142-4  ALTER_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule you are altering, specified as [schema_name.]rule_name. For example, to alter a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>condition</td>
<td>The condition to be associated with the rule. If non-NULL, then the procedure replaces the existing condition of the rule with the specified condition.</td>
</tr>
<tr>
<td>evaluation_context</td>
<td>An evaluation context name in the form [schema_name.]evaluation_context_name. If the schema is not specified, then the current user is the default. If non-NULL, then the procedure replaces the existing evaluation context of the rule with the specified evaluation context.</td>
</tr>
<tr>
<td>remove_evaluation_context</td>
<td>If TRUE, then the procedure sets the evaluation context for the rule to NULL, which effectively removes the evaluation context from the rule.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure retains any evaluation context for the specified rule.</td>
</tr>
<tr>
<td></td>
<td>If the evaluation_context parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
<tr>
<td>action_context</td>
<td>If non-NULL, then the procedure changes the action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.</td>
</tr>
<tr>
<td>remove_action_context</td>
<td>If TRUE, then the procedure sets the action context for the rule to NULL, which effectively removes the action context from the rule.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure retains any action context for the specified rule.</td>
</tr>
<tr>
<td></td>
<td>If the action_context parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
<tr>
<td>rule_comment</td>
<td>If non-NULL, then the existing comment of the rule is replaced by the specified comment.</td>
</tr>
<tr>
<td>remove_rule_comment</td>
<td>If TRUE, then the procedure sets the comment for the rule to NULL, which effectively removes the comment from the rule.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure retains any comment for the specified rule.</td>
</tr>
<tr>
<td></td>
<td>If the rule_comment parameter is non-NULL, then this parameter should be set to FALSE.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER_ON_RULE privilege on the rule
• Have ALTER_ANY_RULE system privilege
• Be the owner of the rule being altered

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

• Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
• Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
• Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

See Also:
Rule TYPES for more information about the types used with the DBMS_RULE_ADM package

142.3.4 CREATE_EVALUATION_CONTEXT Procedure

This procedure creates a rule evaluation context. A rule evaluation context defines external data that can be referenced in rule conditions. The external data can either exist as variables or as table data.

Syntax

```
DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name    IN  VARCHAR2,
    table_aliases              IN  SYS.RE$TABLE_ALIAS_LIST  DEFAULT NULL,
    variable_types             IN  SYS.RE$VARIABLE_TYPE_LIST DEFAULT NULL,
    evaluation_function        IN  VARCHAR2                   DEFAULT NULL,
    evaluation_context_comment IN  VARCHAR2                   DEFAULT NULL);
```

Parameters

Table 142-5  CREATE_EVALUATION_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation_context_name</td>
<td>The name of the evaluation context you are creating, specified as [schema_name.]evaluation_context_name. For example, to create an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_aliases</td>
<td>Table aliases that specify the tables in an evaluation context. The table aliases can be used to reference tables in rule conditions.</td>
</tr>
<tr>
<td>variable_types</td>
<td>A list of variables for the evaluation context</td>
</tr>
</tbody>
</table>


Table 142-5  (Cont.) CREATE_EVALUATION_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation_function</td>
<td>An optional function that will be called to evaluate rules using the evaluation context. It must have the same form as the DBMS_RULE.EVALUATE procedure. If the schema is not specified, then the current user is the default. See &quot;Usage Notes&quot; for more information about the evaluation function.</td>
</tr>
<tr>
<td>evaluation_context_comment</td>
<td>An optional description of the rule evaluation context.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context being created and have CREATE_EVALUATION_CONTEXT_OBJ system privilege
- Have CREATE_ANY_EVALUATION_CONTEXT system privilege

See Also:

Rule TYPEs for more information about the types used with the DBMS_RULE_ADM package

The evaluation function must have the following signature:

```sql
FUNCTION evaluation_function_name(  
    rule_set_name       IN   VARCHAR2,  
    evaluation_context  IN   VARCHAR2,  
    event_context       IN   SYS.RE$NV_LIST DEFAULT NULL,  
    table_values        IN   SYS.RE$TABLE_VALUE_LIST DEFAULT NULL,  
    column_values       IN   SYS.RE$COLUMN_VALUE_LIST DEFAULT NULL,  
    variable_values     IN   SYS.RE$VARIABLE_VALUE_LIST DEFAULT NULL,  
    attribute_values    IN   SYS.RE$ATTRIBUTE_VALUE_LIST DEFAULT NULL,  
    stop_on_first_hit   IN   BOOLEAN                      DEFAULT FALSE,  
    simple_rules_only   IN   BOOLEAN                      DEFAULT FALSE,  
    true_rules          OUT  SYS.RE$RULE_HIT_LIST,  
    maybe_rules         OUT  SYS.RE$RULE_HIT_LIST);  
RETURN BINARY_INTEGER;  
```

Note:

Each parameter is required and must have the specified datatype. However, you can change the names of the parameters.

The return value of the function must be one of the following:

- DBMS_RULE_ADM.EVALUATION_SUCCESS: The user specified evaluation function completed the rule set evaluation successfully. The rules engine returns the results of the evaluation
obtained by the evaluation function to the rules engine client using the
DBMS_RULE.EVALUATE procedure.

- **DBMS_RULE_ADM.EVALUATION_CONTINUE**: The rules engine evaluates the rule set as
  if there were no evaluation function. The evaluation function is not used, and any
  results returned by the evaluation function are ignored.

- **DBMS_RULE_ADM.EVALUATION_FAILURE**: The user specified evaluation function
  failed. Rule set evaluation stops, and an error is raised.

### 142.3.5 CREATE_RULE Procedure

This procedure creates a rule.

**Syntax**

```sql
DBMS_RULE_ADM.CREATE_RULE(
    rule_name           IN  VARCHAR2,
    condition           IN  VARCHAR2,
    evaluation_context  IN  VARCHAR2        DEFAULT NULL,
    action_context      IN  SYS.RE$NV_LIST  DEFAULT NULL,
    rule_comment        IN  VARCHAR2        DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule you are creating, specified as [schema_name.]rule_name. For example, to create a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>condition</td>
<td>The condition to be associated with the rule. A condition evaluates to TRUE or FALSE and can be any condition allowed in the WHERE clause of a SELECT statement. For example, the following is a valid rule condition: department_id = 30 Ensure that the proper case is used for text in rule conditions. <strong>Note</strong>: Do not include the word &quot;WHERE&quot; in the condition.</td>
</tr>
<tr>
<td>evaluation_context</td>
<td>An optional evaluation context name in the form [schema_name.]evaluation_context_name, which is associated with the rule. If the schema is not specified, then the current user is the default. If evaluation_context is not specified, then the rule inherits the evaluation context from its rule set.</td>
</tr>
<tr>
<td>action_context</td>
<td>The action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.</td>
</tr>
<tr>
<td>rule_comment</td>
<td>An optional description of the rule</td>
</tr>
</tbody>
</table>
Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

• Be the owner of the rule being created and have the CREATE_RULE_OBJ system privilege
• Have CREATE_ANY_RULE system privilege

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

• Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
• Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS.
• Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

See Also:

Rule TYPES for more information about the types used with the DBMS_RULE_ADM package

142.3.6 CREATE_RULE_SET Procedure

This procedure creates a rule set.

Syntax

```
DBMS_RULE_ADM.CREATE_RULE_SET(
   rule_set_name       IN  VARCHAR2,
   evaluation_context  IN  VARCHAR2  DEFAULT NULL,
   rule_set_comment    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_set_name</td>
<td>The name of the rule set you are creating, specified as [schema_name.]rule_set_name. For example, to create a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>evaluation_context</td>
<td>An optional evaluation context name in the form [schema_name.]evaluation_context_name, which applies to all rules in the rule set that are not associated with an evaluation context explicitly. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>rule_set_comment</td>
<td>An optional description of the rule set</td>
</tr>
</tbody>
</table>
Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule set being created and have CREATE_RULE_SET_OBJ system privilege
- Have CREATE_ANY_RULE_SET system privilege

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have EXECUTE_ON_EVALUATION_CONTEXT privilege on the evaluation context
- Have EXECUTE_ANY_EVALUATION_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
- Be the evaluation context owner

142.3.7 DROP_EVALUATIONCONTEXT Procedure

This procedure drops a rule evaluation context.

Syntax

```sql
DBMS_RULE_ADM.DROP_EVALUATIONCONTEXT(
    evaluation_context_name  IN  VARCHAR2,
    force                    IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation_context_name</td>
<td>The name of the evaluation context you are dropping, specified as [schema_name.evaluation_context_name]. For example, to drop an evaluation context named dept_eval_context in the hr schema, enter hr.dept_eval_context for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, then the procedure removes the rule evaluation context from all rules and rule sets that use it. If FALSE and no rules or rule sets use the rule evaluation context, then the procedure drops the rule evaluation context. If FALSE and one or more rules or rule sets use the rule evaluation context, then the procedure raises an exception. Caution: Setting force to TRUE can result in rules and rule sets that do not have an evaluation context. If neither a rule nor the rule set it is in has an evaluation context, and no evaluation context was specified for the rule by the ADD_RULE procedure, then the rule cannot be evaluated.</td>
</tr>
</tbody>
</table>
Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context
- Have `DROP_ANY_EVALUATION_CONTEXT` system privilege

142.3.8 DROP_RULE Procedure

This procedure drops a rule.

Syntax

```sql
DBMS_RULE_ADM.DROP_RULE(
    rule_name  IN  VARCHAR2,
    force      IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 142-9  DROP_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule you are dropping, specified as [schema_name.]rule_name. For example, to drop a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, then the procedure removes the rule from all rule sets that contain it. If FALSE and no rule sets contain the rule, then the procedure drops the rule. If FALSE and one or more rule sets contain the rule, then the procedure raises an exception.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule
- Have `DROP_ANY_RULE` system privilege

Note:

- To remove a rule from a rule set without dropping the rule from the database, use the `REMOVE_RULE` procedure.
- The rule evaluation context associated with the rule, if any, is not dropped when you run this procedure.
142.3.9 DROP_RULE_SET Procedure

This procedure drops a rule set.

Syntax

DBMS_RULE_ADM.DROP_RULE_SET(
    rule_set_name  IN  VARCHAR2,
    delete_rules   IN  BOOLEAN   DEFAULT FALSE);

Parameters

Table 142-10  DROP_RULE_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_set_name</td>
<td>The name of the rule set you are dropping, specified as [schema_name.]rule_set_name. For example, to drop a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>delete_rules</td>
<td>If TRUE, then the procedure drops any rules that are in the rule set. If any of the rules in the rule set are also in another rule set, then these rules are not dropped. If FALSE, then the procedure does not drop the rules in the rule set.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have DROP_ANY_RULE_SET system privilege
- Be the owner of the rule set

Note:

The rule evaluation context associated with the rule set, if any, is not dropped when you run this procedure.

142.3.10 GRANT_OBJECT_PRIVILEGE Procedure

This procedure grants the specified object privilege on the specified object to the specified user or role. If a user owns the object, then the user automatically is granted all privileges on the object, with grant option.

Syntax

DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege     IN  BINARY_INTEGER,
    object_name   IN  VARCHAR2,
    grantee       IN  VARCHAR2,
    grant_option  IN  BOOLEAN   DEFAULT FALSE);
Parameters

Table 142-11  GRANT_OBJECT_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The name of the object privilege to grant to the grantee on the object. See &quot;Usage Notes&quot; for the available object privileges.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object for which you are granting the privilege to the grantee, specified as [schema_name.object_name]. For example, to grant the privilege on a rule set named apply_rules in the hr schema, enter hr.apply_rules for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.</td>
</tr>
<tr>
<td>grantee</td>
<td>The name of the user or role for which the privilege is granted. The specified user cannot be the owner of the object.</td>
</tr>
<tr>
<td>grant_option</td>
<td>If TRUE, then the specified user or users granted the specified privilege can grant this privilege to others. If FALSE, then the specified user or users granted the specified privilege cannot grant this privilege to others.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the object on which the privilege is granted
- Have the same privilege as the privilege being granted with the grant option

In addition, if the object is a rule set, then the user must have EXECUTE privilege on all the rules in the rule set with grant option or must own the rules in the rule set.

Table 142-12 lists the object privileges.

Table 142-12  Object Privileges for Evaluation Contexts, Rules, and Rule Sets

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS.DBMS_RULE_ADM.ALL_ON_EVALUATION_CONTEXT</td>
<td>Alter and execute a particular evaluation context in another user’s schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.ALL_ON_RULE</td>
<td>Alter and execute a particular rule in another user’s schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET</td>
<td>Alter and execute a particular rule set in another user’s schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.ALTER_ON_EVALUATION_CONTEXT</td>
<td>Alter a particular evaluation context in another user’s schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.ALTER_ON_RULE</td>
<td>Alter a particular rule in another user’s schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.ALTER_ON_RULE_SET</td>
<td>Alter a particular rule set in another user’s schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.EXECUTE_ON_EVALUATION_CONTEXT</td>
<td>Execute a particular evaluation context in another user’s schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE</td>
<td>Execute a particular rule in another user’s schema</td>
</tr>
</tbody>
</table>
Table 142-12  (Cont.) Object Privileges for Evaluation Contexts, Rules, and Rule Sets

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET</td>
<td>Execute a particular rule set in another user's</td>
</tr>
<tr>
<td></td>
<td>schema</td>
</tr>
</tbody>
</table>

Examples

For example, to grant the HR user the privilege to alter a rule named hr_dml in the strmadmin schema, enter the following:

BEGIN
    DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
        privilege => SYS.DBMS_RULE_ADM.ALTER_ON_RULE,
        object_name => 'strmadmin.hr_dml',
        grantee    => 'hr',
        grant_option => FALSE);
END;
/

142.3.11 GRANT_SYSTEM_PRIVILEGE Procedure

This procedure grant the specified system privilege to the specified user or role.

Syntax

DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege IN  BINARY_INTEGER,
    grantee   IN  VARCHAR2,
    grant_option IN  BOOLEAN DEFAULT FALSE);

Parameters

Table 142-13  GRANT_SYSTEM_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The name of the system privilege to grant to the grantee.</td>
</tr>
<tr>
<td>grantee</td>
<td>The name of the user or role for which the privilege is granted</td>
</tr>
<tr>
<td>grant_option</td>
<td>If TRUE, then the specified user or users granted the specified privilege</td>
</tr>
<tr>
<td></td>
<td>can grant the system privilege to others.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the specified user or users granted the specified privilege</td>
</tr>
<tr>
<td></td>
<td>cannot grant the system privilege to others.</td>
</tr>
</tbody>
</table>

Usage Notes

Table 142-14 lists the system privileges.

Table 142-14  System Privileges for Evaluation Contexts, Rules, and Rule Sets

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS.DBMS_RULE_ADM.ALTER_ANY_EVALUATION_CONTEXT</td>
<td>Alter any evaluation context owned by any user</td>
</tr>
</tbody>
</table>
### Table 142-14  (Cont.) System Privileges for Evaluation Contexts, Rules, and Rule Sets

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS.DBMS_RULE_ADM.ALTER_ANY_RULE</td>
<td>Alter any rule owned by any user</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.ALTER_ANY_RULE_SET</td>
<td>Alter any rule set owned by any user</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.CREATE_ANY_EVALUATION_CONTEXT</td>
<td>Create a new evaluation context in any schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ</td>
<td>Create a new evaluation context in the grantee's schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.CREATE_ANY_RULE</td>
<td>Create a new rule in any schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.CREATE_RULE_OBJ</td>
<td>Create a new rule in the grantee's schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.CREATE_RULE_SET</td>
<td>Create a new rule set in any schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.CREATE_RULE SET OBJ</td>
<td>Create a new rule set in the grantee's schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.DROP_ANY_EVALUATION_CONTEXT</td>
<td>Drop any evaluation context in any schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.DROP_ANY_RULE</td>
<td>Drop any rule in any schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.DROP_ANY_RULE_SET</td>
<td>Drop any rule set in any schema</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.EXECUTE_ANY_EVALUATION_CONTEXT</td>
<td>Execute any evaluation context owned by any user</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE</td>
<td>Execute any rule owned by any user</td>
</tr>
<tr>
<td>SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE_SET</td>
<td>Execute any rule set owned by any user</td>
</tr>
</tbody>
</table>

For example, to grant the `strmadmin` user the privilege to create a rule set in any schema, enter the following:

```sql
BEGIN
    DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
        privilege => SYS.DBMS_RULE_ADM.CREATE_ANY_RULE_SET,
        grantee   => 'strmadmin',
        grant_option => FALSE);
END;
/
```

**Note:**

When you grant a privilege on "ANY" object (for example, `ALTER ANY RULE`), and the initialization parameter `O7_DICTIONARY_ACCESSIBILITY` is set to `FALSE`, you give the user access to that type of object in all schemas except the `SYS` schema. By default, the initialization parameter `O7_DICTIONARY_ACCESSIBILITY` is set to `FALSE`.

If you want to grant access to an object in the `SYS` schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter to `TRUE`. Then privileges granted on "ANY" object allows access to any schema, including `SYS`.
142.3.12 REMOVE_RULE Procedure

This procedure removes the specified rule from the specified rule set.

Syntax

```sql
DBMS_RULE_ADM.REMOVE_RULE(
    rule_name                IN  VARCHAR2,
    rule_set_name            IN  VARCHAR2,
    evaluation_context       IN  VARCHAR2  DEFAULT NULL,
    all_evaluation_contexts  IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

**Table 142-15  REMOVE_RULE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule you are removing from the rule set, specified as <code>[schema_name.]rule_name</code>. For example, to remove a rule named <code>all_a</code> in the <code>hr</code> schema, enter <code>hr.all_a</code> for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>rule_set_name</td>
<td>The name of the rule set from which you are removing the rule, specified as <code>[schema_name.]rule_set_name</code>. For example, to remove the rule from a rule set named <code>apply_rules</code> in the <code>hr</code> schema, enter <code>hr.apply_rules</code> for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>evaluation_context</td>
<td>The name of the evaluation context associated with the rule you are removing, specified as <code>[schema_name.]evaluation_context</code>. For example, to specify an evaluation context named <code>dept_eval_context</code> in the <code>hr</code> schema, enter <code>hr.dept_eval_context</code> for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>all_evaluation_contexts</td>
<td>If <code>TRUE</code>, then the procedure removes the rule from the rule set with all of its associated evaluation contexts. If <code>FALSE</code>, then the procedure only removes the rule with the specified evaluation context. This parameter is relevant only if the same rule is added more than once to the rule set with different evaluation contexts.</td>
</tr>
</tbody>
</table>
Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have `ALTER_ON_RULE_SET` privilege on the rule set
- Have `ALTER_ANY_RULE_SET` system privilege
- Be the owner of the rule set

Note:

This procedure does not drop a rule from the database. To drop a rule from the database, use the `DROP_RULE` procedure.

142.3.13 REVOKE_OBJECT_PRIVILEGE Procedure

This procedure revokes the specified object privilege on the specified object from the specified user or role.

Syntax

```
DBMS_RULE_ADM.REVOKE_OBJECT_PRIVILEGE(
    privilege    IN  BINARY_INTEGER,
    object_name  IN  VARCHAR2,
    revokee      IN  VARCHAR2);
```

Parameters

Table 142-16  REVOKE_OBJECT_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The name of the object privilege on the object to revoke from the revokee. See <code>GRANT_OBJECT_PRIVILEGE Procedure</code> for a list of the object privileges.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object for which you are revoking the privilege from the revokee, specified as <code>[schema_name.]object_name</code>. For example, to revoke an object privilege on a rule set named <code>apply_rules</code> in the <code>hr</code> schema, enter <code>hr.apply_rules</code> for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.</td>
</tr>
<tr>
<td>revokee</td>
<td>The name of the user or role from which the privilege is revoked. The user who owns the object cannot be specified.</td>
</tr>
</tbody>
</table>
142.3.14 REVOKE_SYSTEM_PRIVILEGE Procedure

This procedure revokes the specified system privilege from the specified user or role.

Syntax

```sql
DBMS_RULE_ADM.REVOKE_SYSTEM_PRIVILEGE(
    privilege  IN  BINARY_INTEGER,
    revokee    IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>The name of the system privilege to revoke from the revokee. See <a href="#">GRANT_SYSTEM_PRIVILEGE Procedure</a> for a list of the system privileges.</td>
</tr>
<tr>
<td>revokee</td>
<td>The name of the user or role from which the privilege is revoked</td>
</tr>
</tbody>
</table>
The DBMS_SCHEDULER package provides a collection of scheduling functions and procedures that can be called from any PL/SQL program.

This chapter contains the following topics:

• Deprecated Subprograms
• Security Model
• Rules and Limits
• Operational Notes
• Data Structures
• Summary of DBMS_SCHEDULER Subprograms

See Also:
Oracle Database Administrator's Guide for more information regarding how to use DBMS_SCHEDULER

143.1 DBMS_SCHEDULER Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 12c Release 1 (12.1):

• CREATE_CREDENTIAL Procedure
• DROP_CREDENTIAL Procedure

143.2 DBMS_SCHEDULER Security Model

The DBMS_SCHEDULER package ignores privileges granted on scheduler objects, such as jobs or chains, through roles. Object privileges must be granted directly to the user.

143.3 DBMS_SCHEDULER Rules and Limits

These rules apply when using the DBMS_SCHEDULER package.

• Only SYS can perform actions on objects in the SYS schema.
• Several of the procedures accept comma-delimited lists of object names. If you provide a list of names, then the Scheduler stops executing the list at the first object that returns an
error. Therefore, the Scheduler does not perform the tasks needed for the remaining objects on the list.

For example, consider the statement `DBMS_SCHEDULER.STOP_JOB ('job1, job2, job3, sys.jobclass1, sys.jobclass2, sys.jobclass3');`

If `job3` cannot be stopped, then the jobs that follow it, `jobclass1, jobclass2, and jobclass3` cannot be stopped. The jobs that preceded `job3, job1 and job2, are stopped.`

- Performing an action on an object that does not exist returns a PL/SQL exception stating that the object does not exist.

### 143.4 DBMS_SCHEDULER Operational Notes

The Scheduler uses a rich **calendaring syntax** to enable you to define repeating schedules, such as "every Tuesday and Friday at 4:00 p.m." or "the second Wednesday of every month." This calendaring syntax is used in calendaring expressions in the `repeat_interval` argument of a number of package subprograms. Evaluating a calendaring expression results in a set of discrete timestamps.

See *Oracle Database Administrator's Guide* for examples of the calendaring syntax.

**Calendaring Syntax**

This section starts with the calendaring syntax. It is followed by descriptions of various parts of the syntax.

In the calendaring syntax, * means 0 or more.

```
repeat_interval = regular_schedule | combined_schedule

regular_schedule = frequency_clause
[";" interval_clause] [";" bymonth_clause] [";" byweekno_clause]
[";" byyearday_clause] [";" bydate_clause] [";" bymonthday_clause]
[";" byday_clause] [";" byhour_clause] [";" byminute_clause]
[";" bysecond_clause] [";" bysetpos_clause] [";" include_clause]
[";" exclude_clause] [";" intersect_clause][";" periods_clause]
[";" byperiod_clause]
```

```
frequency_clause = "FREQ" "=" ( predefined_frequency | user_defined_frequency )
predefined_frequency = "YEARLY" | "MONTHLY" | "WEEKLY" | "DAILY" |
"HOURLY" | "MINUTELY" | "SECONDLY"
user_defined_frequency = named_schedule

interval_clause = "INTERVAL" "=" intervalnum
intervalnum = 1 through 99

bymonth_clause = "BMONTH" "=" monthlist
monthlist = month ( "," month)*
month = numeric_month | char_month
numeric_month = 1 | 2 | 3 ... 12
char_month = "JAN" | "FEB" | "MAR" | "APR" | "MAY" | "JUN" |
"JUL" | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"
byweekno_clause = "BYWEEKNO" "=" weeknumber_list
weeknumber_list = weeknumber ( "," weeknumber)*
weeknumber = [minus] weekno
weekno = 1 through 53
byyearday_clause = "BYYEARDAY" "=" yearday_list
yearday_list = yearday ( "," yearday)*
```

---

**Note:** The above code snippet is a representation of the calendaring syntax as described in the document. It is not a fragment of code to be executed, as it is part of the description of the syntax itself.
yearday = [minus] yeardaynum
yeardaynum = 1 through 366
bydate_clause = "BYDATE" "=" date_list
date_list = date ( "," date)*
date = [YYYY]MMD [ offset | span ]
bymonthday_clause = "BYMONTHDAY" "=" monthday_list
monthday_list = monthday ( "," monthday)*
monthday = [minus] monthdaynum
monthdaynum = 1 through 31
byday_clause = "BYDAY" "=" byday_list
byday_list = byday ( "," byday)*
byday = [weekdaynum] day
weekdaynum = [minus] daynum
daynum = 1 through 53 /* if frequency is yearly */
daynum = 1 through 5 /* if frequency is monthly */
day = "MON" | "TUE" | "WED" | "THU" | "FRI" | "SAT" | "SUN"
BYTIME clause:  BYTIME=[hour_minute_second_list|minute_second_list]
hour_minute_second_list: hh24mmss, .., hh24mmss
minute_second_list: mmss, .. mmss
byhour_clause = "BYHOUR" "=" hour_list
hour_list = hour ( "," hour)*
hour = 0 through 23
byminute_clause = "BYMINUTE" "=" minute_list
minute_list = minute ( "," minute)*
minute = 0 through 59
bysecond_clause = "BYSECOND" "=" second_list
second_list = second ( "," second)*
second = 0 through 59
bysetpos_clause = "BYSETPOS" "=" setpos_list
setpos_list = setpos ( "," setpos)*
setpos = [minus] setpos_num
setpos_num = 1 through 9999
include_clause = "INCLUDE" "=" schedule_list
exclude_clause = "EXCLUDE" "=" schedule_list
intersect_clause = "INTERSECT" "=" schedule_list
schedule_list = schedule_clause ( "," schedule_clause)*
schedule_clause = named_schedule [ offset ]
named_schedule = [schema "."] schedule
periods_clause = "PERIODS" "=" periodnum
byperiod_clause = "BYPERIOD" "=" period_list
period_list = periodnum ( "," periodnum)*
periodnum = 1 through 100
offset = ("+" | "-" | ["OFFSET:"] duration_val
span = ("+" | "-" | "+" | "-" | "SPAN:" duration_val
duration_val = dur-weeks | dur_days
dur_weeks = numofweeks "W"
dur_days = numofdays "D"
numofweeks = 1 through 53
numofdays = 1 through 376
minus = "-"
combined_schedule = schedule_list
### Table 143-1  Values for repeat_interval

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREQ</td>
<td>This specifies the type of recurrence. It must be specified. The possible predefined frequency values are YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, and SECONDLY. Alternatively, specifies an existing schedule to use as a user-defined frequency.</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>This specifies a positive integer representing how often the recurrence repeats. The default is 1, which means every second for secondly, every day for daily, and so on. The maximum value is 99.</td>
</tr>
<tr>
<td>BYMONTH</td>
<td>This specifies which month or months you want the job to execute in. You can use numbers such as 1 for January and 3 for March, as well as three-letter abbreviations such as FEB for February and JUL for July.</td>
</tr>
<tr>
<td>BYWEEKNO</td>
<td>This specifies the week of the year as a number. It follows ISO-8601, which defines the week as starting with Monday and ending with Sunday; and the first week of a year as the first week, which is mostly within the Gregorian year. The first week is equivalent to the following two variants: the week that contains the first Thursday of the Gregorian year; and the week containing January 4th. The ISO-8601 week numbers are integers from 1 to 52 or 53; parts of week 1 may be in the previous calendar year; parts of week 52 may be in the following calendar year; and if a year has a week 53, parts of it must be in the following calendar year. As an example, in the year 1998, the ISO week 1 began on Monday December 29th, 1997; and the last ISO week (week 53) ended on Sunday January 3rd, 1999. So December 29th, 1997, is in the ISO week 1998-01, and January 1st, 1999, is in the ISO week 1998-53. byweekno is only valid for YEARLY. Examples of invalid specifications are &quot;FREQ=YEARLY; BYWEEKNO=1; BYMONTH=12&quot; and &quot;FREQ=YEARLY;BYWEEKNO=53;BYMONTH=1&quot;.</td>
</tr>
<tr>
<td>BYYEARDAY</td>
<td>This specifies the day of the year as a number. Valid values are 1 to 366. An example is 69, which is March 10 (31 for January, 28 for February, and 10 for March). 69 evaluates to March 10 for non-leap years and March 9 in leap years. -2 will always evaluate to December 30th independent of whether it is a leap year.</td>
</tr>
<tr>
<td>BYDATE</td>
<td>This specifies a list of dates, where each date is of the form [YYYY]MMDD. A list of consecutive dates can be generated by using the SPAN modifier, and a date can be adjusted with the OFFSET modifier. An example of a simple BYDATE clause follows: BYDATE=0115,0315,0615,0915,1215,20060115 The following SPAN example is equivalent to BYDATE=0110,0111,0112,0113,0114, which is a span of 5 days starting at 1/10: BYDATE=0110+SPAN:5D The plus sign in front of the SPAN keyword indicates a span starting at the supplied date. The minus sign indicates a span ending at the supplied date, and the &quot;^&quot; sign indicates a span of n days or weeks centered around the supplied date. If n is an even number, it is adjusted up to the next odd number. Offsets adjust the supplied date by adding or subtracting n days or weeks. BYDATE=0205-OFFSET:2W is equivalent to BYDATE=0205-14D (the OFFSET: keyword is optional), which is also equivalent to BYDATE=0122.</td>
</tr>
</tbody>
</table>
### Table 143-1  (Cont.) Values for repeat_interval

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BYMONTHDAY</code></td>
<td>This specifies the day of the month as a number. Valid values are 1 to 31. An example is 10, which means the 10th day of the selected month. You can use the minus sign (-) to count backward from the last day, so, for example, <code>BYMONTHDAY=-1</code> means the last day of the month and <code>BYMONTHDAY=-2</code> means the next to last day of the month.</td>
</tr>
<tr>
<td><code>BYDAY</code></td>
<td>This specifies the day of the week from Monday to Sunday in the form MON, TUE, and so on. Using numbers, you can specify the 26th Friday of the year, if using a YEARLY frequency, or the 4th THU of the month, using a MONTHLY frequency. Using the minus sign, you can say the second to last Friday of the month. For example, <code>-1 FRI</code> is the last Friday of the month.</td>
</tr>
<tr>
<td><code>BYHOUR</code></td>
<td>This specifies the hour on which the job is to run. Valid values are 0 to 23. As an example, 10 means 10 a.m.</td>
</tr>
<tr>
<td><code>BYMINUTE</code></td>
<td>This specifies the minute on which the job is to run. Valid values are 0 to 59. As an example, 45 means 45 minutes past the chosen hour.</td>
</tr>
<tr>
<td><code>BYSECOND</code></td>
<td>This specifies the second on which the job is to run. Valid values are 0 to 59. As an example, 30 means 30 seconds past the chosen minute.</td>
</tr>
<tr>
<td><code>BYSETPOS</code></td>
<td>This selects one or more items, by position, in the list of timestamps that result after the whole calendaring expression is evaluated. It is useful for requirements such as running a job on the last workday of the month. Rather than attempting to express this with the other BY clauses, you can code the calendaring expression to evaluate to a list of every workday of the month, and then add the <code>BYSETPOS</code> clause to select only the last item of that list. Assuming that workdays are Monday through Friday, the syntax would then be: <code>FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; BYSETPOS=-1</code> Valid values are 1 through 9999. A negative number selects an item from the end of the list (-1 is the last item, -2 is the next to last item, and so on) and a positive number selects from the front of the list. The <code>BYSETPOS</code> clause is always evaluated last. <code>BYSETPOS</code> is only supported with the MONTHLY and YEARLY frequencies. The <code>BYSETPOS</code> clause is applied to the list of timestamps once per frequency period. For example, when the frequency is defined as MONTHLY, the Scheduler determines all valid timestamps for the month, orders that list, and then applies the <code>BYSETPOS</code> clause. The Scheduler then moves on to the next month and repeats the procedure. Assuming a start date of Jun 10, 2004, the example evaluates to: Jun 30, Jul 30, Aug 31, Sep 30, Oct 29, and so on.</td>
</tr>
<tr>
<td><code>INCLUDE</code></td>
<td>This includes one or more named schedules in the calendaring expression. That is, the set of timestamps defined by each included named schedule is added to the results of the calendaring expression. If an identical timestamp is contributed by both an included schedule and the calendaring expression, it is included in the resulting set of timestamps only once. The named schedules must have been defined with the CREATE_SCHEDULE procedure. This clause only works on a full day and therefore cannot be used with <code>BYHOUR</code>, <code>BYMIN</code>, and <code>BYSECOND</code>.</td>
</tr>
</tbody>
</table>
### Table 143-1  (Cont.) Values for repeat_interval

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **EXCLUDE** | This excludes one or more named schedules from the calendaring expression. That is, the set of timestamps defined by each excluded named schedule is removed from the results of the calendaring expression. The named schedules must have been defined with the `CREATE_SCHEDULE` procedure.  
This clause only works on a full day and therefore cannot be used with `BYHOUR`, `BYMIN`, and `BYSECOND`. |
| **INTERSECT** | This specifies an intersection between the calendaring expression results and the set of timestamps defined by one or more named schedules. Only the timestamps that appear both in the calendaring expression and in one of the named schedules are included in the resulting set of timestamps.  
For example, assume that the named schedule `last_sat` indicates the last Saturday in every month, and that for the year 2005, the only months where the last day of the month is also a Saturday are April and December. Assume also that the named schedule `end_qtr` indicates the last day of each quarter in 2005:  
These calendaring expressions result in the dates that follow:  
FREQ=MONTHLY; BYMONTHDAY=-1; INTERSECT=last_sat,end_qtr  
In this example, the terms FREQ=MONTHLY; BYMONTHDAY=-1 indicate the last day of each month.  
This clause only works on a full day and therefore cannot be used with `BYHOUR`, `BYMIN`, and `BYSECOND`. |
| **PERIODS** | This identifies the number of periods that together form one cycle of a user-defined frequency. It is used in the repeat_interval expression of the schedule that defines the user-defined frequency. It is mandatory when the repeat_interval expression in the main schedule contains a `BYPERIOD` clause. The following example defines the quarters of a fiscal year.  
FREQ=YEARLY; BYDATE=0301,0601,0901,1201; PERIODS=4 |
| **BYPERIOD** | This selects periods from a user-defined frequency. For example, if a main schedule names a user-defined frequency schedule that defines the fiscal quarters shown in the previous example, the clause `BYPERIOD=2,4` in the main schedule selects the 2nd and 4th fiscal quarters. |

**Combining Schedules**

There are two ways to combine schedules:

- Using a combined schedule expression, which is a list of individual schedules  
  For example, to create a schedule for all company holidays, you provide a list of individual schedules, where each schedule in the list defines a single holiday. The Scheduler evaluates each individual schedule, and then returns a union of the timestamps returned by each individual schedule.
• Embedding other schedules into the main schedule using include, exclude, and intersect clauses

With this method, the embedded schedules inherit certain attributes from the main schedule.

- Timestamps generated by the INCLUDE clause that fall into periods that are skipped by the main schedule are ignored. This is the case when the main schedule skips periods due to the INTERVAL clause, the BYPERIOD clause, or the BYMONTH clause for freq=monthly.

- Days that are added by the INCLUDE clause follow the hourly/minutely/secondly execution pattern of the main schedule.

- When the INCLUDE clause is present, no date-specific defaults are retrieved from the start date (but time-specific defaults can be). (See "Start Dates and Repeat Intervals", later in this section.) For example, a repeat_interval of FREQ=MONTHLY;INCLUDE=HOLIDAY executes only on holidays and not on the month/day defaults retrieved from the start date.

The following is an example:

BEGIN
  dbms_scheduler.create_schedule('embed_sched', repeat_interval => 'FREQ=YEARLY;BYDATE=0130,0220,0725');
  dbms_scheduler.create_schedule('main_sched', repeat_interval => 'FREQ=MONTHLY;INTERVAL=2;BYMONTHDAY=15;BYHOUR=9,17;INCLUDE=embed_sched');
END;
/

In this example, the dates 1/30, 2/20, and 7/25 are added to the main schedule. However, the Scheduler does not include dates that fall in months that are skipped by the INTERVAL clause. If the start date of the main schedule is 1/1/2005, then 2/20 is not added. On the dates that are added, the embedded schedule follows the execution pattern of the main schedule: jobs are executed at 9:00 a.m. and 5:00 p.m. on 1/30 and 7/25. If the embedded schedule does not itself have a start date, it inherits the start date from the main schedule.

User-Defined Frequencies

Instead of using predefined frequencies like DAILY, WEEKLY, MONTHLY, and so on, you can create your own frequencies by creating a schedule that returns the start date of each period. For example, the following repeat_interval expression is used in a schedule named fiscal_year that defines the start of each quarter in a fiscal year:

FREQ=YEARLY;BYDATE=0301,0601,0901,1201;PERIODS=4

To return the last Wednesday of every quarter, you create a schedule (the "main schedule") that uses the fiscal_year schedule as a user-defined frequency:

FREQ=fiscal_year;BYDAY=-1WED

Periods in a user-defined frequency do not have to be equal in length. In the main schedule, the BYSETPOS clause and numbered weekdays are recalculated based on the size of each period. To select dates in specific periods, you must use the BYPERIOD clause in the main schedule. To enable this, the schedule that is used as the user-defined frequency must include a PERIODS clause, and it must set its start date appropriately. The first date returned by this schedule is used as the starting point of period 1.
As another example, assuming work days are Monday through Friday, to get the last work day of the 2nd and 4th quarters of the fiscal year, the `repeat_interval` clause in the main schedule is the following:

```
FREQ=fiscal_year;BYDAY=MON,TUE,WED,THU,FRI;BYPERIOD=2,4;BYSETPOS=-1
```

**Start Dates and Repeat Intervals**

The Scheduler retrieves the date and time from the job or schedule start date and incorporates them as defaults into the `repeat_interval`. For example, if the specified frequency is yearly and there is no `BYMONTH` or `BYMONTHDAY` clause in the repeat interval, then the month and day that the job runs on are retrieved from the start date. Similarly, if frequency is monthly but there is no `BYMONTHDAY` clause in the repeat interval, then the day of the month that the job runs on is retrieved from the start date. If present, `BYHOUR`, `BYMINUTE`, and `BYSECOND` defaults are also retrieved from the start date, and used if those clauses are not specified. Note that if the `INCLUDE`, `EXCLUDE`, or `INTERSECT` clauses are present, no date-related defaults are retrieved from the start date, but time-related defaults are. The following are some examples:

```
start_date:      4/15/05 9:00:00
repeat_interval: freq=yearly
```

is expanded internally to:

```
freq=yearly;bymonth=4;bymonthday=15;byhour=9;byminute=0;bysecond=0
```

The preceding schedule executes on 04/15/05 9:00:00, 04/15/06 9:00:00, 04/15/07 9:00:00, and so on.

For the next example, assume that schedule `S1` has a `repeat_interval` of `FREQ=YEARLY;BYDATE=0701`.

```
start_date:      01/20/05 9:00:00
repeat_interval: freq=yearly;include=S1
```

is expanded internally to:

```
freq=yearly;byhour=9;byminute=0;bysecond=0;include=S1
```

Because an `INCLUDE` clause is present, date-related information is not retrieved from the start date. However, time-specific information is, so the preceding schedule executes on 07/01/05 9:00:00, 07/01/06 9:00:00, 07/01/08 9:00:00, and so on.

**General Rules**

When using a calendaring expression, consider the following rules:

- For a regular schedule (as opposed to a combined schedule), the calendar string must start with the frequency clause. All other clauses are optional and can be put in any order.
- All clauses are separated by a semicolon, and each clause can be present at most once, with the exception of the `include`, `exclude`, and `intersect` clauses.
- Spaces are allowed between syntax elements and the strings are case-insensitive.
- The list of values for a specific `BY` clause do not need to be ordered.
When not enough `BY` clauses are present to determine what the next date is, this information is retrieved from the start date. For example, "FREQ=YEARLY" with a start date of 02/15/2003 becomes "FREQ=YEARLY;BYMONTH=FEB;BYMONTHTDAY=15", which means every year on the 15th of February.

"FREQ=YEARLY;BYMONTH=JAN,JUL" with start date 01/21/2003 becomes "FREQ=YEARLY;BYMONTH=JAN,JUL;BYMONTHTDAY=21", which means every year on January 21 and July 21.

The `byweekno` clause is only allowed if the frequency is `YEARLY`. It cannot be used with other frequencies. When it is present, it will return all days in that week number. If you want to limit it to specific days within the week, you have to add a `BYDAY` clause. For example, "FREQ=YEARLY;BYWEEKNO=2" with a start date of 01/01/2003 will return:

01/06/2003, 01/07/2003, 01/08/2003, 01/09/2003, 01/10/2003, 01/11/2003, 01/12/2003, 01/05/2004, 01/06/2004, 01/07/2004, .... and so on.

Note that when the `byweekno` clause is used, it is possible that the dates returned are from a year other than the current year. For example, if returning dates for the year 2004 and the calendar string is "FREQ=YEARLY;BYWEEKNO=1,53" for the specified week numbers in 2004, it will return the dates:

12/29/03, 12/30/03, 12/31/03, 01/01/04, 01/02/04, 01/03/04, 01/04/04, 12/27/04, 12/28/04, 12/29/04, 12/30/04, 12/31/04, 01/01/05, 01/02/05

For those `BY` clauses that do not have a consistent range of values, you can count backward by putting a "-" in front of the numeric value. For example, specifying `BYMONTHTDAY=31` will not give you the last day of every month, because not every month has 31 days. Instead, `BYMONTHTDAY=-1` will give you the last day of the month.

This is not supported for `BY` clauses that are fixed in size. In other words, `BYMONTHT`, `BYHOUR`, `BYMINUTE`, and `BYSECOND` are not supported.

The basic values for the `BYDAY` clause are the days of the week. When the frequency is `YEARLY`, or `MONTHLY`, you are allowed to specify a positive or negative number in front of each day of the week. In the case of `YEARLY`, `BYDAY=40MON`, indicates the 40th Monday of the year. In the case of `MONTHLY`, `BYDAY=-2SAT`, indicates the second to last Saturday of the month.

Note that positive or negative numbers in front of the weekdays are not supported for other frequencies and that in the case of yearly, the number ranges from -53 ... -1, 1 ... 53, whereas for the monthly frequency it is limited to -5 ... -1, 1 ... 5.

If no number is present in front of the weekday it specifies, every occurrence of that weekday in the specified frequency.

The first day of the week is Monday.

Repeating jobs with frequencies smaller than daily follow their frequencies exactly across daylight savings adjustments. For example, suppose that a job is scheduled to repeat every 3 hours, the clock is moved forward from 1:00 a.m. to 2:00 a.m., and the last time the job ran was midnight. Its next scheduled time will be 4:00 a.m. Thus, the 3 hour period between subsequent job runs is retained. The same applies when the clock is moved back. This behavior is not the case for repeating jobs that have frequencies of daily or larger. For example, if a repeating job is supposed to be executed on a daily basis at midnight, it will continue to run at midnight if the clock is moved forward or backward. When the execution time of such a daily (or larger frequency) job happens to fall inside a window where the clock is moved forward, the job executes at the end of the window.
• The calendaring syntax does not allow you to specify a time zone. Instead the Scheduler retrieves the time zone from the `start_date` argument. If jobs must follow daylight savings adjustments, then you must specify a region name for the time zone of the `start_date`. For example specifying the `start_date` time zone as `'US/Eastern'` in New York ensures that daylight saving adjustments are automatically applied. If instead, the time zone of the `start_date` is set to an absolute offset, such as `-5:00`, then daylight savings adjustments are not followed and your job execution is off by an hour for half the year.

• When `start_date` is `NULL`, the Scheduler determines the time zone for the repeat interval as follows:

1. It checks whether or not the session time zone is a region name. The session time zone can be set by either:
   - Issuing an `ALTER SESSION` statement, for example:
     ```sql
     SQL> ALTER SESSION SET time_zone = 'Asia/Shanghai';
     ```
   - Setting the `ORA_SDTZ` environment variable.

2. If the session time zone is an absolute offset instead of a region name, the Scheduler uses the value of the `DEFAULT_TIMEZONE` Scheduler attribute. For more information, see the `SET_SCHEDULER_ATTRIBUTE` Procedure.

3. If the `DEFAULT_TIMEZONE` attribute is `NULL`, the Scheduler uses the time zone of `systimestamp` when the job or window is enabled.

**BYSETPOS Clause Rules**

The following are rules for the `BYSETPOS` clause.

• The `BYSETPOS` clause is the last clause to be evaluated. It is processed after all other `BY` clauses and the `INCLUDE`, `EXCLUDE` and `INTERSECT` clauses have been evaluated.

• The `INTERVAL` clause does not change the size of the period to which the `BYSETPOS` clause is applied. For example, when the frequency is set to monthly and interval is set to 3, the list of timestamps to which `BYSETPOS` is applied is generated from a month, not a quarter. The only impact of the `INTERVAL` clause is to cause months to be skipped. However, you can still select the second to last workday of the quarter like this:

```
FREQ=MONTHLY;INTERVAL=3;BYDAY=MON,TUE,FRI;BYSETPOS=-2
```

provided that you set the start date in the right month. This example returns the next to last workday of a month, and repeats once a quarter.

• To get consistent results, the set to which `BYSETPOS` is applied is determined from the beginning of the frequency period independently of when the evaluation occurs. Whether the Scheduler evaluates

```
FREQ=MONTHLY;BYDAY=MON,TUE,FRI;BYSETPOS=1,3
```

on 01/01/2004 or 01/15/2004, in both cases the expression evaluates to Friday 01/02/2004, and Tuesday 01/06/2004. The only difference is that when the expression is evaluated on 01/15/2004, the Scheduler determines that there are no matches in January because the timestamps found are in the past, and it moves on to the matches in the next month, February.

**BYDATE Clause Rules**
The following are rules for the **BYDATE** clause.

- If dates in the **BYDATE** clause do not have their optional year component, the job runs on those dates every year.
- The job execution times on the included dates are derived from the **BY** clauses in the calendaring expression. For example, if `repeat_interval` is defined as
  ```plaintext
  freq=daily;byhour=8,13,18;byminute=0;bysecond=0;bydate=0502,0922
  ```
  then the execution times on 05/02 and 09/22 are 8:00 a.m., 1:00 p.m., and 6:00 p.m.

**EXCLUDE Clause Rules**

Excluded dates without a time component are 24 hour periods. All timestamps that fall on an excluded date are removed. In the following example, `jan_fifteen` is a named schedule that resolves to the single date of 01/15:

```plaintext
freq=monthly;bymonthday=15,30;byhour=8,13,18;byminute=0;bysecond=0;
exclude=jan_fifteenth
```

In this case, all three instances of the job are removed for 01/15.

**OFFSET Rules**

You can adjust the dates of individual named schedules by adding positive offsets to them. For example, to execute `JOB2` exactly 15 days after every occurrence of `JOB1`, add `OFFSET:15D` to the schedule of `JOB1`, as follows:

```plaintext
BEGIN
  dbms_scheduler.create_schedule('job2_schedule', repeat_interval =>
    'job1_schedule+OFFSET:15D');
END;
/
```

Note that negative offsets to named schedules are not supported.

**Example 143-1    Putting It All Together**

This example demonstrates the use of user-defined frequencies, spans, offsets, and the **BYSETPOS** and **INCLUDE** clauses. (Note that the **OFFSET**: keyword is optional in an offset clause.)

Many companies in the retail industry share the same fiscal year. The fiscal year starts on the Sunday closest to February 1st, and subsequent quarters start exactly 13 weeks later. The fiscal year schedule for the retail industry can be defined as the following:

```plaintext
BEGIN
  dbms_scheduler.create_schedule('year_start', repeat_interval =>
    'FREQ=YEARLY;BYDATE=0201^SPAN:1W;BYDAY=SUN');
  dbms_scheduler.create_schedule('retail_fiscal_year',
    to_timestamp_tz('15-JAN-2005 12:00:00','DD-MON-YYYY HH24:MI:SS'),
    'year_start,year_start+13w,year_start+26w,year_start+39w;periods=4');
END;
/
```

The following schedule can be used to execute a job on the 5th day off in the 2nd and the 4th quarters of the retail industry. This assumes that Saturday and Sunday are off days as well as the days in the existing holiday schedule.

```plaintext
BEGIN
  dbms_scheduler.create_schedule('fifth_day_off', repeat_interval =>
    'FREQ=YEARLY;BYDATE=0201^SPAN:1W;BYDAY=SUN');
  dbms_scheduler.create_schedule('retail_fiscal_year',
    to_timestamp_tz('15-JAN-2005 12:00:00','DD-MON-YYYY HH24:MI:SS'),
    'year_start,year_start+13w,year_start+26w,year_start+39w;periods=4');
END;
/
```
143.5 DBMS_SCHEDULER Data Structures

The DBMS_SCHEDULER package defines OBJECT types and TABLE types.

OBJECT Types

- JOBARG Object Type
- JOB_DEFINITION Object Type
- JOBATTR Object Type
- SCHEDULER$_STEP_TYPE Object Type
- SCHEDULER$_EVENT_INFO Object Type
- SCHEDULER_FILEWATCHER_RESULT Object Type
- SCHEDULER_FILEWATCHER_REQUEST Object Type

TABLE Types

- JOBARG_ARRAY Table Type
- JOB_DEFINITION_ARRAY Table Type
- JOBATTR_ARRAY Table Type
- SCHEDULER$_STEP_TYPE_LIST Table Type

143.5.1 DBMS_SCHEDULER JOBARG Object Type

This type is used by the JOB and JOBATTR object types. It represents a job argument in a batch of job arguments.

Syntax

```plsql
TYPE jobarg IS OBJECT ( 
  arg_position         NUMBER, 
  arg_text_value       VARCHAR2(4000), 
  arg_anydata_value    ANYDATA, 
  arg_operation        VARCHAR2(5));
```

Attributes

**Table 143-2  JOBARG Object Type Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg_position</td>
<td>Position of the argument</td>
</tr>
<tr>
<td>arg_text_value</td>
<td>Value of the argument if the type is VARCHAR2</td>
</tr>
<tr>
<td>arg_anydata_value</td>
<td>Value of the argument if the type is AnyData</td>
</tr>
</tbody>
</table>
Table 143-2  (Cont.) JOBARG Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg_operation</td>
<td>Type of the operation:</td>
</tr>
<tr>
<td></td>
<td>· SET</td>
</tr>
<tr>
<td></td>
<td>· RESET</td>
</tr>
</tbody>
</table>

JOBARG Constructor Function

This constructor function constructs a job argument. It is overloaded to construct job arguments with different types of values.

Syntax

Constructs a job argument with a text value.

```
constructor function jobarg (
    arg_position        IN POSITIVEN,
    arg_value           IN VARCHAR2)
RETURN SELF AS RESULT;
```

Constructs a job argument with an AnyData value.

```
constructor function jobarg (
    arg_position        IN POSITIVEN,
    arg_value           IN ANYDATA)
RETURN SELF AS RESULT;
```

Constructs a job argument with a NULL value.

```
constructor function jobarg (
    arg_position        IN POSITIVEN,
    arg_reset           IN BOOLEAN DEFAULT FALSE)
RETURN SELF AS RESULT;
```

Parameters

Table 143-3  JOBARG Constructor Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg_position</td>
<td>Position of the argument</td>
</tr>
<tr>
<td>arg_value</td>
<td>Value of the argument</td>
</tr>
<tr>
<td>arg_reset</td>
<td>If arg_reset is TRUE, then the argument at that position is reset. Setting arg_reset to FALSE (which is the default) will create an argument with a NULL value.</td>
</tr>
</tbody>
</table>

JOBARG_ARRAY Table Type

Syntax

```
TYPE jobarg_array IS TABLE OF jobarg;
```
143.5.2 JOBARG_ARRAY Table Type

The jobarg_array type is a table of jobarg.

Syntax

TYPE jobarg_array IS TABLE OF jobarg;

143.5.3 DBMS_SCHEDULER JOB_DEFINITION Object Type

This type is used by the CREATE_JOBS procedure and represents a job in a batch of jobs.

Syntax

TYPE job_definition IS OBJECT (  
  job_name                       VARCHAR2(100),  
  job_class                      VARCHAR2(32),  
  job_style                      VARCHAR2(11),  
  program_name                   VARCHAR2(100),  
  job_action                     VARCHAR2(4000),  
  job_type                       VARCHAR2(20),  
  schedule_name                  VARCHAR2(65),  
  repeat_interval                VARCHAR2(4000),  
  schedule_limit                 INTERVAL DAY TO SECOND,  
  start_date                     TIMESTAMP WITH TIME ZONE,  
  end_date                       TIMESTAMP WITH TIME ZONE,  
  event_condition                VARCHAR2(4000),  
  queue_spec                     VARCHAR2(100),  
  number_of_arguments            NUMBER,  
  arguments                      SYS.JOBARG_ARRAY,  
  job_priority                   NUMBER,  
  job_weight                     NUMBER,  
  max_run_duration               INTERVAL DAY TO SECOND,  
  max_runs                       NUMBER,  
  max_failures                   NUMBER,  
  logging_level                  NUMBER,  
  restartable                    VARCHAR2(5),  
  stop_on_window_close           VARCHAR2(5),  
  raise_events                   NUMBER,  
  comments                       VARCHAR2(240),  
  auto_drop                      VARCHAR2(5),  
  enabled                        VARCHAR2(5),  
  follow_default_timezone        VARCHAR2(5),  
  parallel_instances             VARCHAR2(5),  
  aq_job                         VARCHAR2(5),  
  instance_id                    NUMBER,  
  credential_name                VARCHAR2(65),  
  destination                    VARCHAR2(20),  
  database_role                  VARCHAR2(20),  
  allow_runs_in_restricted_mode  VARCHAR2(5),  
  restart_on_recovery            BOOLEAN;  
  restart_on_failure             BOOLEAN;  
)
Object Attributes

Table 143-4 provides brief descriptions of the attributes of the JOB_DEFINITION object type. For more complete information about these attributes, see the "CREATE_JOB Procedure" and the "SET_ATTRIBUTE Procedure".

Table 143-4  JOB_DEFINITION Object Types

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Name of the job</td>
</tr>
<tr>
<td>job_class</td>
<td>Name of the job class</td>
</tr>
<tr>
<td>job_style</td>
<td>Style of the job:</td>
</tr>
<tr>
<td></td>
<td>· REGULAR</td>
</tr>
<tr>
<td></td>
<td>· LIGHTWEIGHT</td>
</tr>
<tr>
<td></td>
<td>· IN_MEMORY_RUNTIME</td>
</tr>
<tr>
<td></td>
<td>· IN_MEMORY_FULL</td>
</tr>
<tr>
<td>program_name</td>
<td>Name of the program that the job runs</td>
</tr>
<tr>
<td>job_action</td>
<td>Inline action of the job. This is either the code for an anonymous PL/SQL block or the name of a stored procedure, external executable, or chain.</td>
</tr>
<tr>
<td>job_type</td>
<td>Job action type ('PLSQL_BLOCK', 'STORED_PROCEDURE', 'EXECUTABLE', 'CHAIN', 'EXTERNAL_SCRIPT', 'SQL_SCRIPT', and 'BACKUP_SCRIPT')</td>
</tr>
<tr>
<td>schedule_name</td>
<td>Name of the schedule that specifies when the job has to execute</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>Inline time-based schedule</td>
</tr>
<tr>
<td>schedule_limit</td>
<td>Maximum delay time between scheduled and actual job start before a job run is canceled</td>
</tr>
<tr>
<td>start_date</td>
<td>Start date and time of the job</td>
</tr>
<tr>
<td>end_date</td>
<td>End date and time of the job</td>
</tr>
<tr>
<td>event_condition</td>
<td>Event condition for event-based jobs</td>
</tr>
<tr>
<td>queue_spec</td>
<td>File watcher name or queue specification for event-based jobs</td>
</tr>
<tr>
<td>number_of_arguments</td>
<td>Number of job arguments</td>
</tr>
<tr>
<td>arguments</td>
<td>Array of job arguments</td>
</tr>
<tr>
<td>job_priority</td>
<td>Job priority</td>
</tr>
<tr>
<td>job_weight</td>
<td>*** Deprecated in Oracle Database 11gr2. Do not change the value of this attribute from the default, which is 1. Weight of the job for parallel execution.</td>
</tr>
<tr>
<td>max_run_duration</td>
<td>Maximum run duration of the job</td>
</tr>
<tr>
<td>max_runs</td>
<td>Maximum number of runs before the job is marked as completed</td>
</tr>
<tr>
<td>max_failures</td>
<td>Maximum number of failures tolerated before the job is marked as broken</td>
</tr>
<tr>
<td>logging_level</td>
<td>Job logging level</td>
</tr>
<tr>
<td>restartable</td>
<td>Indicates whether the job is restartable (TRUE) or not (FALSE)</td>
</tr>
</tbody>
</table>
### Table 143-4  (Cont.) JOB_DEFINITION Object Types

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop_on_window_close</td>
<td>Indicates whether the job is stopped when the window that it runs in ends (TRUE) or not (FALSE). Equivalent to the stop_on_window_close job attribute described in the SET_ATTRIBUTE Procedure.</td>
</tr>
<tr>
<td>raise_events</td>
<td>State changes that raise events</td>
</tr>
<tr>
<td>comments</td>
<td>Comments on the job</td>
</tr>
<tr>
<td>auto_drop</td>
<td>If TRUE (the default), indicates that the job should be dropped once completed</td>
</tr>
<tr>
<td>enabled</td>
<td>Indicates whether the job should be enabled immediately after creating it (TRUE) or not (FALSE)</td>
</tr>
<tr>
<td>follow_default_timezone</td>
<td>If TRUE and if the job start_date is null, then when the default_timezone scheduler attribute is changed, the Scheduler recomputes the next run date and time for this job so that it is in accordance with the new time zone.</td>
</tr>
<tr>
<td>parallel_instances</td>
<td>For event-based jobs only.</td>
</tr>
<tr>
<td></td>
<td>If TRUE, on the arrival of the specified event, the Scheduler creates a new lightweight job to handle that event, so multiple instances of the same event-based job can run in parallel.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then an event is discarded if it is raised while the job that handles it is already running.</td>
</tr>
<tr>
<td>aq_job</td>
<td>For internal use only</td>
</tr>
<tr>
<td>instance_id</td>
<td>The instance ID of the instance that the job must run on</td>
</tr>
<tr>
<td></td>
<td>For in-memory full jobs, the instance_id value determines in which instance to stop the job; if left NULL, the job is stopped in all instances.</td>
</tr>
<tr>
<td>credential_name</td>
<td>The credential to use for a single destination or the default credential for a group of destinations</td>
</tr>
<tr>
<td>destination</td>
<td>The name of a single external destination or database destination, or a group name of type external destination or database destination</td>
</tr>
<tr>
<td>database_role</td>
<td>In an Oracle Data Guard environment, the database role ('PRIMARY' or 'LOGICAL STANDBY') for which the job runs</td>
</tr>
<tr>
<td>allow_runs_in_restricted_mode</td>
<td>If TRUE, the job is permitted to run when the database is in restricted mode, provided that the job owner is permitted to log in during this mode</td>
</tr>
<tr>
<td>restart_on_recovery</td>
<td>If set to TRUE for a job and the job is stopped by a database shutdown, then the job is restarted when the database is recovered.</td>
</tr>
<tr>
<td></td>
<td>If set to FALSE, and the job is stopped by a database shutdown, then the job is marked as stopped when the database is recovered.</td>
</tr>
<tr>
<td>restart_on_failure</td>
<td>If set to TRUE for a job and the job fails due to an application error, then the job is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the job is marked FAILED.</td>
</tr>
<tr>
<td></td>
<td>If set to FALSE (the default), a failed job is immediately marked FAILED.</td>
</tr>
</tbody>
</table>

---

Chapter 143
DBMS_SCHEDULER Data Structures
**JOB_DEFINITION Constructor Function**

This constructor function constructs a job_definition object.

**Syntax**

```
constructor function job_definition (  
  job_name                IN     VARCHAR2,  
  job_style               IN     VARCHAR2 DEFAULT 'REGULAR',  
  program_name            IN     VARCHAR2 DEFAULT NULL,  
  job_action              IN     VARCHAR2 DEFAULT NULL,  
  job_type                IN     VARCHAR2 DEFAULT NULL,  
  schedule_name           IN     VARCHAR2 DEFAULT NULL,  
  repeat_interval         IN     VARCHAR2 DEFAULT NULL,  
  event_condition         IN     VARCHAR2 DEFAULT NULL,  
  queue_spec              IN     VARCHAR2 DEFAULT NULL,  
  start_date              IN     TIMESTAMP WITH TIME ZONE DEFAULT NULL,  
  end_date                IN     TIMESTAMP WITH TIME ZONE DEFAULT NULL,  
  number_of_arguments     IN     NATURAL DEFAULT NULL,  
  arguments               IN     SYS.JOBARG_ARRAY DEFAULT NULL,  
  job_class               IN     VARCHAR2 DEFAULT 'DEFAULT_JOB_CLASS',  
  schedule_limit          IN     INTERVAL DAY TO SECOND DEFAULT NULL,  
  job_priority            IN     NATURAL DEFAULT NULL,  
  job_weight              IN     NATURAL DEFAULT NULL,  
  max_run_duration        IN     INTERVAL DAY TO SECOND DEFAULT NULL,  
  max_runs                IN     NATURAL DEFAULT NULL,  
  max_failures            IN     NATURAL DEFAULT NULL,  
  logging_level           IN     NATURALN DEFAULT 64,  
  restartable             IN     BOOLEAN DEFAULT FALSE,  
  stop_on_window_close    IN     BOOLEAN DEFAULT FALSE,  
  raise_events            IN     NATURAL DEFAULT NULL,  
  comments                IN     VARCHAR2 DEFAULT NULL,  
  auto_drop               IN     BOOLEAN DEFAULT TRUE,  
  enabled                 IN     BOOLEAN DEFAULT FALSE,  
  follow_default_timezone IN     BOOLEAN DEFAULT FALSE,  
  parallel_instances      IN     BOOLEAN DEFAULT FALSE,  
  aq_job                  IN     BOOLEAN DEFAULT FALSE,  
  instance_id             IN     NATURAL DEFAULT NULL,  
  credential_name         IN     VARCHAR2 DEFAULT NULL,  
  destination             IN     VARCHAR2 DEFAULT NULL,  
  database_role           IN     VARCHAR2 DEFAULT NULL,  
  allow_runs_in_restricted_mode IN BOOLEAN DEFAULT FALSE)  
RETURN SELF AS RESULT;
```

**JOB_DEFINITION_ARRAY Table Type**

**Syntax**

```
TYPE job_definition_array IS TABLE OF job_definition;
```

### 143.5.4 JOB_DEFINITION_ARRAY Table Type

The type job_definition_array is a table of job_definition.

**Syntax**

```
TYPE job_definition_array IS TABLE OF job_definition;
```
143.5.5 JOBATTR Object Type

This type is used by the `SET_JOB_ATTRIBUTES` procedure and represents a job attribute in a batch of job attributes.

Syntax

```
TYPE jobattr IS OBJECT (
  job_name             VARCHAR2(100),
  attr_name            VARCHAR2(30),
  char_value           VARCHAR2(4000),
  char_value2          VARCHAR2(4000),
  args_value           JOBARG_ARRAY,
  num_value            NUMBER,
  timestamp_value      TIMESTAMP(6) WITH TIME ZONE,
  interval_value       INTERVAL DAY(2) TO SECOND(6));
```

Attributes

Table 143-5  JOBATTR Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Name of the job</td>
</tr>
<tr>
<td>attr_name</td>
<td>Name of the attribute</td>
</tr>
<tr>
<td>char_value</td>
<td>Value of the argument if the type is VARCHAR2</td>
</tr>
<tr>
<td>char_value2</td>
<td>Second VARCHAR2 attribute value</td>
</tr>
<tr>
<td>args_value</td>
<td>Value of the argument if the type is a JOBARG array</td>
</tr>
<tr>
<td>num_value</td>
<td>Value of the argument if the type is NUMBER</td>
</tr>
<tr>
<td>timestamp_value</td>
<td>Value of the argument if the type is TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>interval_value</td>
<td>Value of the argument if the type is INTERVAL DAY TO SECOND</td>
</tr>
</tbody>
</table>

JOBATTR Constructor Function

This constructor function constructs a job attribute. It is overloaded to create attribute values of the following types: VARCHAR2, NUMBER, TIMESTAMP WITH TIME ZONE, INTERVAL DAY TO SECOND, and an array of JOBARG types.

Syntax

```
constructor function jobattr (  
  job_name            IN VARCHAR2,
  attr_name            IN VARCHAR2,
  attr_value           IN VARCHAR2,
  attr_value2          IN VARCHAR2 DEFAULT NULL)
RETURN SELF AS RESULT;
```

```
constructor function jobattr (  
  job_name            IN VARCHAR2,
  attr_name            IN VARCHAR2,
  attr_value           IN [NUMBER, BOOLEAN,
                         TIMESTAMP WITH TIME ZONE,]  
```
INTERVAL DAY TO SECOND, JOBARG_ARRAY})
RETURN SELF AS RESULT;

constructor function jobattr {
    job_name          IN VARCHAR2,
    attr_name         IN VARCHAR2
RETURN SELF AS RESULT;

Parameters

Table 143-6  JOBATTR Constructor Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Name of the job</td>
</tr>
<tr>
<td>attr_name</td>
<td>Name of the argument</td>
</tr>
<tr>
<td>attr_value</td>
<td>Value of the argument</td>
</tr>
<tr>
<td>attr_value2</td>
<td>Most attributes have only one value associated with them, but some can have two. The attr_value2 argument is for this optional second value.</td>
</tr>
</tbody>
</table>

JOBATTR Table Type

Syntax

TYPE jobattr_array IS TABLE OF jobattr;

143.5.6 JOBATTR_ARRAY Table Type

The type jobattr_array is a table of jobattr.

Syntax

TYPE jobattr_array IS TABLE OF jobattr;

143.5.7 SCHEDULER$_STEP_TYPE Object Type

This type is used by RUN_CHAIN to return a list of chain steps with an initial state.

Syntax

TYPE scheduler$_step_type IS OBJECT (  
    step_name VARCHAR2(32),  
    step_type VARCHAR2(32));

Attributes

Table 143-7  SCHEDULER$_STEP_TYPE Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>step_name</td>
<td>Name of the step</td>
</tr>
<tr>
<td>step_type</td>
<td>State of the step</td>
</tr>
</tbody>
</table>
143.5.8 SCHEDULER$_STEP_TYPE_LIST Table Type

This type is a table of scheduler$_step_type.

Syntax

```
TYPE scheduler$_step_type_list IS TABLE OF scheduler$_step_type;
```

143.5.9 SCHEDULER$_EVENT_INFO Object Type

This the datatype of the Scheduler event queue SYS.SCHEDULER$_EVENT_QUEUE, from which your application consumes job state events raised by the Scheduler.

It is a secure queue owned by SYS.

Syntax

```
TYPE SCHEDULER$_EVENT_INFO IS OBJECT (
    event_type        VARCHAR2(4000),
    object_owner      VARCHAR2(4000),
    object_name       VARCHAR2(4000),
    event_timestamp   TIMESTAMP WITH TIME ZONE,
    error_code        NUMBER,
    error_msg         VARCHAR2(4000),
    event_status      NUMBER,
    log_id            NUMBER,
    run_count         NUMBER,
    failure_count     NUMBER,
    retry_count       NUMBER,
    spare1            NUMBER,
    spare2            NUMBER,
    spare3            VARCHAR2(4000),
    spare4            VARCHAR2(4000),
    spare5            TIMESTAMP WITH TIME ZONE,
    spare6            TIMESTAMP WITH TIME ZONE,
    spare7            RAW(2000),
    spare8            RAW(2000)));
```

Attributes

Table 143-8  SCHEDULER$_EVENT_INFO Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_type</td>
<td>One of &quot;JOB_STARTED&quot;, &quot;JOB_SUCCEEDED&quot;, &quot;JOB_FAILED&quot;, &quot;JOB_BROKEN&quot;, &quot;JOB_COMPLETED&quot;, &quot;JOB_STOPPED&quot;, &quot;JOB_SCH_LIM_REACHED&quot;, &quot;JOB_DISABLED&quot;, &quot;JOB_CHAIN_STALLED&quot;, &quot;JOB_OVER_MAX_DUR&quot;. For descriptions of these event types, see Table 143-84.</td>
</tr>
<tr>
<td>object_owner</td>
<td>Owner of the job that raised the event</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the job that raised the event</td>
</tr>
<tr>
<td>event_timestamp</td>
<td>Time at which the event occurred</td>
</tr>
<tr>
<td>error_code</td>
<td>Applicable only when an error is thrown during job execution. Contains the top-level error code.</td>
</tr>
</tbody>
</table>
Table 143-8 (Cont.) SCHEDULER_EVENT_INFO Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_msg</td>
<td>Applicable only when an error is thrown during job execution. Contains the entire error stack.</td>
</tr>
<tr>
<td>event_status</td>
<td>Adds further qualification to the event type. If event_type is &quot;JOB_STARTED,&quot; status 1 indicates that it is a normal start, and status 2 indicates that it is a retry. If event_type is &quot;JOB_FAILED,&quot; status 4 indicates that it was a failure due to an error that was thrown during job execution, and status 8 indicates that it was an abnormal termination of some kind. If event_type is &quot;JOB_STOPPED,&quot; status 16 indicates that it was a normal stop, and status 32 indicates that it was a stop with the FORCE option set to TRUE.</td>
</tr>
<tr>
<td>log_id</td>
<td>Points to the ID in the scheduler job log from which additional information can be obtained. Note that there need not always be a log entry corresponding to an event. In such cases, log_id is NULL.</td>
</tr>
<tr>
<td>run_count</td>
<td>Run count for the job when the event was raised.</td>
</tr>
<tr>
<td>failure_count</td>
<td>Failure count for the job when the event was raised.</td>
</tr>
<tr>
<td>retry_count</td>
<td>Retry count for the job when the event was raised.</td>
</tr>
<tr>
<td>spare1 - spare8</td>
<td>Not currently in use.</td>
</tr>
</tbody>
</table>

143.5.10 SCHEDULER_FILEWATCHER_RESULT Object Type

This is the datatype of a file arrival event message.

You access the event message as a parameter of an event-based job (or a parameter of a program referenced by an event-based job). The message contains information needed to locate and process a file that arrived on a local or remote system.

Syntax

```
TYPE scheduler_filewatcher_result IS OBJECT (
  destination         VARCHAR2(4000),
  directory_path      VARCHAR2(4000),
  actual_file_name    VARCHAR2(4000),
  file_size           NUMBER,
  file_timestamp      TIMESTAMP WITH TIME ZONE,
  ts_ms_from_epoch    NUMBER,
  matching_requests   SYS.SCHEDULER_FILEWATCHER_REQ_LIST);
```

Attributes

Table 143-9 SCHEDULER_FILEWATCHER_RESULT Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>Destination at which the file was found, expressed as a host name or IP address.</td>
</tr>
<tr>
<td>directory_path</td>
<td>Absolute path of directory in which the file was found.</td>
</tr>
</tbody>
</table>
Table 143-9 (Cont.) SCHEDULER_FILEWATCHER_RESULT Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual_file_name</td>
<td>Actual name of the file that was found. If the file name specified in the file watcher did not contain wildcards, then this is the same as the name specified in the file watcher.</td>
</tr>
<tr>
<td>file_size</td>
<td>Size of the file that was found, in bytes.</td>
</tr>
<tr>
<td>file_timestamp</td>
<td>Timestamp assigned to the file when the file watcher considered the file found, based on the minimum file size and steady state duration attributes.</td>
</tr>
<tr>
<td>ts_ms_from_epoch</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>matching_requests</td>
<td>List of matching requests. This is a TABLE of type objects SCHEDULER_FILEWATCHER_REQUEST. Each matching request corresponds to a file watcher whose destination, directory_path, and file_name attributes matched the arrived file. See &quot;SCHEDULER_FILEWATCHER_REQUEST Object Type&quot;.</td>
</tr>
</tbody>
</table>

143.5.11 SCHEDULER_FILEWATCHER_REQUEST Object Type

This type is returned in the matching_requests attribute of the SCHEDULER_FILEWATCHER_RESULT Object Type. Its attributes are similar to the attributes of a file watcher.

Syntax

```sql
TYPE scheduler_filewatcher_request IS OBJECT (  
    owner                 VARCHAR2(4000),  
    name                  VARCHAR2(4000),  
    requested_path_name   VARCHAR2(4000),  
    requested_file_name   VARCHAR2(4000),  
    credential_owner      VARCHAR2(4000),  
    credential_name       VARCHAR2(4000),  
    min_file_size         NUMBER,  
    steady_state_dur      NUMBER);  
```

Attributes

Table 143-10 SCHEDULER_FILEWATCHER_REQUEST Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Owner of the matched file watcher.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the matched file watcher.</td>
</tr>
<tr>
<td>requested_path_name</td>
<td>Value of the directory_path attribute of the matched file watcher.</td>
</tr>
<tr>
<td>requested_file_name</td>
<td>Value of the file_name attribute of the matched file watcher.</td>
</tr>
<tr>
<td>credential_owner</td>
<td>Owner of the credential referenced by the matched file watcher.</td>
</tr>
<tr>
<td>credential_name</td>
<td>Name of the credential referenced by the matched file watcher.</td>
</tr>
<tr>
<td>min_file_size</td>
<td>Value of the min_file_size attribute of the matched file watcher.</td>
</tr>
</tbody>
</table>
Table 143-10 (Cont.) SCHEDULER_FILEWATCHER_REQUEST Object Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>steady_state_dur</td>
<td>Value of the steady_state_duration attribute of the matched file watcher.</td>
</tr>
</tbody>
</table>

Related Topics

- SCHEDULER_FILEWATCHER_RESULT Object Type
  This is the datatype of a file arrival event message.

143.6 Summary of DBMS_SCHEDULER Subprograms

This table lists the DBMS_SCHEDULER subprograms and briefly describes them.

Table 143-11 DBMS_SCHEDULER Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_EVENT_QUEUE_SUBSCRIBER Procedure</td>
<td>Adds a user as a subscriber to the Scheduler event queue SYS.SCHEDULER$EVENT_QUEUE</td>
</tr>
<tr>
<td>ADD_GROUP_MEMBER Procedure</td>
<td>Adds one or more members to an existing group</td>
</tr>
<tr>
<td>ADD_JOB_EMAIL_NOTIFICATION Procedure</td>
<td>Adds e-mail notifications for a job for a list of recipients and a list of job state events</td>
</tr>
<tr>
<td>ADD_TO_INCOMPATIBILITY Procedure</td>
<td>Adds jobs or programs to an existing incompatibility definition</td>
</tr>
<tr>
<td>ALTER_CHAIN Procedure</td>
<td>Alters specified steps of a chain</td>
</tr>
<tr>
<td>ALTER_RUNNING_CHAIN Procedure</td>
<td>Alters specified steps of a running chain</td>
</tr>
<tr>
<td>CLOSE_WINDOW Procedure</td>
<td>Closes an open window prematurely</td>
</tr>
<tr>
<td>COPY_JOB Procedure</td>
<td>Copies an existing job</td>
</tr>
<tr>
<td>CREATE_CHAIN Procedure</td>
<td>Creates a chain, which is a named series of programs that are linked together for a combined objective</td>
</tr>
<tr>
<td>CREATE_CREDENTIAL Procedure</td>
<td>Creates a credential</td>
</tr>
<tr>
<td>CREATE_DATABASE_DESTINATION Procedure</td>
<td>Creates a database destination for use with remote database jobs</td>
</tr>
<tr>
<td>CREATE_EVENT_SCHEDULE Procedure</td>
<td>Creates an event schedule, which is a schedule that starts a job based on the detection of an event</td>
</tr>
<tr>
<td>CREATE_FILE_WATCHER Procedure</td>
<td>Creates a file watcher, which is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job</td>
</tr>
<tr>
<td>CREATE_GROUP Procedure</td>
<td>Creates a group</td>
</tr>
</tbody>
</table>
### Table 143-11  (Cont.) DBMS_SCHEDULER Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_INCOMPATIBILITY Procedure</td>
<td>Creates an incompatibility definition</td>
</tr>
<tr>
<td>CREATE_JOB Procedure</td>
<td>Creates a single job</td>
</tr>
<tr>
<td>CREATE_JOB_CLASS Procedure</td>
<td>Creates a job class, which provides a way to group jobs for resource allocation and prioritization</td>
</tr>
<tr>
<td>CREATE_JOBS Procedure</td>
<td>Creates multiple jobs</td>
</tr>
<tr>
<td>CREATE_PROGRAM Procedure</td>
<td>Creates a program</td>
</tr>
<tr>
<td>CREATE_RESOURCE Procedure</td>
<td>Specifies resources used by jobs or creates a new resource</td>
</tr>
<tr>
<td>CREATE_SCHEDULE Procedure</td>
<td>Creates a schedule</td>
</tr>
<tr>
<td>CREATE_WINDOW Procedure</td>
<td>Creates a window, which provides a way to automatically activate different resource plans at different times</td>
</tr>
<tr>
<td>DEFINE_ANYDATA_ARGUMENT Procedure</td>
<td>Defines a program argument whose value is of a complex type and must be passed encapsulated in an AnyData object</td>
</tr>
<tr>
<td>DEFINE_CHAIN_EVENT_STEP Procedure</td>
<td>Adds or replaces a chain step and associates it with an event schedule or inline event. See also: DEFINE_CHAIN_STEP.</td>
</tr>
<tr>
<td>DEFINE_CHAIN_RULE Procedure</td>
<td>Adds a rule to an existing chain</td>
</tr>
<tr>
<td>DEFINE_CHAIN_STEP Procedure</td>
<td>Defines a chain step, which can be a program or another (nested) chain. See also: DEFINE_CHAIN_EVENT_STEP.</td>
</tr>
<tr>
<td>DEFINE_METADATA_ARGUMENT Procedure</td>
<td>Defines a special metadata argument for the program. You can retrieve specific metadata through this argument.</td>
</tr>
<tr>
<td>DEFINE_PROGRAM_ARGUMENT Procedure</td>
<td>Defines a program argument whose value can be passed as a string literal to the program</td>
</tr>
<tr>
<td>DISABLE Procedure</td>
<td>Disables a program, job, chain, window, database destination, external destination, file watcher, group, or incompatibility</td>
</tr>
<tr>
<td>DROP_AGENT_DESTINATION Procedure</td>
<td>Drops one or more external destinations. Use only when the preferred method of dropping external destinations—unregistering the Scheduler agent with the database—fails.</td>
</tr>
<tr>
<td>DROP_CHAIN Procedure</td>
<td>Drops an existing chain</td>
</tr>
<tr>
<td>DROP_CHAIN_RULE Procedure</td>
<td>Removes a rule from an existing chain</td>
</tr>
<tr>
<td>DROP_CHAIN_STEP Procedure</td>
<td>Drops a chain step</td>
</tr>
</tbody>
</table>
Table 143-11  (Cont.) DBMS_SCHEDULER Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP_CREDENTIAL Procedure</td>
<td>Drops a credential</td>
</tr>
<tr>
<td>DROP_DATABASE_DESTINATION Procedure</td>
<td>Drops one or more database destinations</td>
</tr>
<tr>
<td>DROP_FILEWATCHER Procedure</td>
<td>Drops one or more file watchers</td>
</tr>
<tr>
<td>DROP_GROUP Procedure</td>
<td>Drops one or more groups</td>
</tr>
<tr>
<td>DROP_INCOMPATIBILITY Procedure</td>
<td>Drops an existing incompatibility definition</td>
</tr>
<tr>
<td>DROP_JOB Procedure</td>
<td>Drops a job or all jobs in a job class</td>
</tr>
<tr>
<td>DROP_JOB_CLASS Procedure</td>
<td>Drops a job class</td>
</tr>
<tr>
<td>DROP_PROGRAM Procedure</td>
<td>Drops a program</td>
</tr>
<tr>
<td>DROP_PROGRAM_ARGUMENT Procedure</td>
<td>Drops a program argument</td>
</tr>
<tr>
<td>DROP_SCHEDULE Procedure</td>
<td>Drops a schedule</td>
</tr>
<tr>
<td>DROP_WINDOW Procedure</td>
<td>Drops a window</td>
</tr>
<tr>
<td>ENABLE Procedure</td>
<td>Enables a program, job, chain, window, database destination, external</td>
</tr>
<tr>
<td></td>
<td>destination, file watcher, or group</td>
</tr>
<tr>
<td>END_DETACHED_JOB_RUN Procedure</td>
<td>Ends a running detached job</td>
</tr>
<tr>
<td>EVALUATECALENDARSTRING Procedure</td>
<td>Evaluates the calendar string and tells you what the next execution date</td>
</tr>
<tr>
<td></td>
<td>of a job or window will be</td>
</tr>
<tr>
<td>EVALUATERUNNINGCHAIN Procedure</td>
<td>Forces reevaluation of the rules of a running chain to trigger any rules</td>
</tr>
<tr>
<td></td>
<td>for conditions that have been satisfied</td>
</tr>
<tr>
<td>GENERATE_JOB_NAME Function</td>
<td>Generates a unique name for a job. This enables you to identify jobs by</td>
</tr>
<tr>
<td></td>
<td>adding a prefix, so, for example, Sally's jobs would be named</td>
</tr>
<tr>
<td></td>
<td>sally1, sally2, and so on</td>
</tr>
<tr>
<td>GETAGENTINFO Function</td>
<td>Returns job information specific to an agent, such as how many are running</td>
</tr>
<tr>
<td></td>
<td>and so on, depending on the attribute selected</td>
</tr>
<tr>
<td>GETAGENTVERSION Function</td>
<td>Returns the version string of a Scheduler agent that is registered with</td>
</tr>
<tr>
<td></td>
<td>the database and is currently running</td>
</tr>
<tr>
<td>GET_ATTRIBUTE Procedure</td>
<td>Retrieves the value of an attribute of an object</td>
</tr>
<tr>
<td>GET_FILE Procedure</td>
<td>Retrieves a file from a host</td>
</tr>
<tr>
<td>GET_SCHEDULER_ATTRIBUTE Procedure</td>
<td>Retrieves the value of a Scheduler attribute</td>
</tr>
<tr>
<td>OPEN_WINDOW Procedure</td>
<td>Opens a window prematurely. The window is opened immediately for the</td>
</tr>
<tr>
<td></td>
<td>duration</td>
</tr>
<tr>
<td>PURGE_LOG Procedure</td>
<td>Purges specific rows from the job and window logs</td>
</tr>
</tbody>
</table>
### Table 143-11 (Cont.) DBMS_SCHEDULER Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT_FILE Procedure</td>
<td>Saves a file to one or more hosts</td>
</tr>
<tr>
<td>REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure</td>
<td>Unsubscribes a user from the Scheduler event queue</td>
</tr>
<tr>
<td>REMOVE_FROM_INCOMPATIBILITY Procedure</td>
<td>Removes jobs or programs from an incompatibility definition</td>
</tr>
<tr>
<td>REMOVE_GROUP_MEMBER Procedure</td>
<td>Removes one or more members from a group</td>
</tr>
<tr>
<td>REMOVE_JOB_EMAIL_NOTIFICATION Procedure</td>
<td>Removes e-mail notifications for a job</td>
</tr>
<tr>
<td>RESET_JOB_ARGUMENT_VALUE Procedure</td>
<td>Resets the current value assigned to an argument defined with the associated program</td>
</tr>
<tr>
<td>RUN_CHAIN Procedure</td>
<td>Immediately runs a chain by creating a run-once job</td>
</tr>
<tr>
<td>RUN_JOB Procedure</td>
<td>Runs a job immediately</td>
</tr>
<tr>
<td>SET_AGENT_REGISTRATION_PASS Procedure</td>
<td>Sets the agent registration password for a database</td>
</tr>
<tr>
<td>SET_ATTRIBUTE Procedure</td>
<td>Changes an attribute of a job, schedule, or other Scheduler object</td>
</tr>
<tr>
<td>SET_ATTRIBUTE_NULL Procedure</td>
<td>Changes an attribute of an object to NULL</td>
</tr>
<tr>
<td>SET_JOB_ANYDATA_VALUE Procedure</td>
<td>Sets the value of a job argument encapsulated in an AnyData object</td>
</tr>
<tr>
<td>SET_JOB_ARGUMENT_VALUE Procedure</td>
<td>Sets the value of a job argument</td>
</tr>
<tr>
<td>SET_JOB_ATTRIBUTES Procedure</td>
<td>Sets the value of a job attribute</td>
</tr>
<tr>
<td>SET_RESOURCE_CONSTRAINT Procedure</td>
<td>Specifies the resources used by jobs</td>
</tr>
<tr>
<td>SET_SCHEDULER_ATTRIBUTE Procedure</td>
<td>Sets the value of a Scheduler attribute</td>
</tr>
<tr>
<td>STOP_JOB Procedure</td>
<td>Stops a currently running job or all jobs in a job class</td>
</tr>
</tbody>
</table>

#### 143.6.1 ADD_EVENT_QUEUE_SUBSCRIBER Procedure

This procedure adds a user as a subscriber to the Scheduler event queue
SYS.SCHEDULER$ EVENT_QUEUE, and grants the user permission to dequeue from this queue using the designated agent.

**Syntax**

```sql
DBMS_SCHEDULER.ADD_EVENT_QUEUE_SUBSCRIBER (
    subscriber_name     IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 143-12  ADD_EVENT_QUEUE_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriber_name</td>
<td>Name of the Oracle Streams Advanced Queuing (AQ) agent to be used to subscribe to the Scheduler event queue. If NULL, an agent is created and assigned the user name of the calling user.</td>
</tr>
</tbody>
</table>

Usage Notes

The subscription is rule-based. The rule permits the user to see only events raised by jobs that the user owns, and filters out all other messages. If an AQ agent with the same name already exists, an error is raised.

143.6.2 ADD_GROUP_MEMBER Procedure

This procedure adds one or more members to an existing group.

Syntax

```sql
DBMS_SCHEDULER.ADD_GROUP_MEMBER (
    group_name              IN VARCHAR2,
    member                  IN VARCHAR2);
```

Parameters

Table 143-13  ADD_GROUP_MEMBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_name</td>
<td>The name of the group.</td>
</tr>
<tr>
<td>member</td>
<td>A comma-separated list of members to add to the group. Members must match the group type. A group of the same type can be a member. The Scheduler immediately expands the included group name into its list of members. An error is returned if any of the members do not exist. A member that is already in the group is skipped, and no error is generated. The keyword LOCAL can be included as a member for database destination or external destination groups. See the &quot;CREATE_GROUP Procedure&quot; for information about this keyword.</td>
</tr>
</tbody>
</table>

Usage Notes

The following users may add members to a group:

- The group owner
- A user that has been granted the ALTER object privilege on the group
- A user with the CREATE ANY JOB system privilege
- You must have the MANAGE SCHEDULER privilege to add a member to a group of type WINDOW.
143.6.3 ADD_JOB_EMAIL_NOTIFICATION Procedure

This procedure adds e-mail notifications for a job. E-mails are then sent to the specified list of recipients whenever any of the specified job state events is raised.

Syntax

```
DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (  
    job_name             IN VARCHAR2,  
    recipients           IN VARCHAR2,  
    sender               IN VARCHAR2 DEFAULT NULL,  
    subject              IN VARCHAR2 DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_SUBJECT,  
    body                 IN VARCHAR2 DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_BODY,  
    events               IN VARCHAR2 DEFAULT 'JOB_FAILED,JOB_BROKEN,JOB_SCH_LIM_REACHED, JOB_CHAIN_STALLED,JOB_OVER_MAX_DUR',  
    filter_condition     IN VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 143-14 ADD_JOB_EMAIL_NOTIFICATION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Name of the job that e-mail notifications are added for. Cannot be NULL.</td>
</tr>
<tr>
<td>recipients</td>
<td>Comma-separated list of e-mail addresses to send notifications to. E-mail notifications for all listed events are sent to all recipients. Cannot be NULL.</td>
</tr>
<tr>
<td>sender</td>
<td>e-mail address to use as the sender address (the From: address) in the e-mail header. If NULL or omitted, the e-mail address specified in the Scheduler attribute email_sender is used. See Oracle Database Administrator's Guide for more information on this Scheduler attribute.</td>
</tr>
<tr>
<td>subject</td>
<td>The subject to use in the e-mail header. Table 143-15 describes the variables that you can include within this parameter. The Scheduler assigns values to these variables before sending the notification. If subject is omitted, the default subject is used. The default subject is the following text, where text enclosed in the '%&lt;' character represents a variable:</td>
</tr>
</tbody>
</table>

'Oracle Scheduler Job Notification -
%job_owner%.%job_name%.%job_subname% %event_type%'
**Table 143-14** (Cont.) ADD_JOB_EMAIL_NOTIFICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>The body of the e-mail message. Table 143-15 describes the variables that you can include within this parameter. The Scheduler assigns values to these variables before sending the notification. If body is omitted, the default body is used. The default body is the following text, where text enclosed in the % character represents a variable:</td>
</tr>
</tbody>
</table>

'Job: %job_owner%.%job_name%.%job_subname%
Event: %event_type%
Date: %event_timestamp%
Log id: %log_id%
Job class: %job_class_name%
Run count: %run_count%
Failure count: %failure_count%
Retry count: %retry_count%
Error code: %error_code%
Error message: %error_message%

| events    | Comma-separate list of job state events to send e-mail notifications for. Cannot be NULL. A notification is sent to all recipients if any of the listed events is raised. Table 143-84 lists the valid events for this parameter. If events is omitted, notifications are sent for the following default events: JOB_FAILED, JOB_BROKEN, JOB_SCH_LIM_REACHED, JOB_CHAIN_STALLED, JOB_OVER_MAX_DUR |

| filter_condition | Used to filter events to send e-mail notifications for. If NULL, all occurrences of the specified events cause e-mail notifications to be sent. filter_condition must be a boolean SQL WHERE clause that may refer to the :event bind variable. This bind variable is automatically bound to an object of type SCHEDULER$_EVENT_INFO that represents the raised event. For example, to send an e-mail notification only when the error number in an event is 600 or 700, use the following filter_condition: :event.error_code=600 or :event.error_code=700 |

See "SCHEDULER$_EVENT_INFO Object Type".

**Table 143-15** lists the variables that you can use in the subject and body arguments.

**Table 143-15** Variables Used in the SUBJECT and BODY Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>%job_owner%</td>
<td>Schema in which job was created</td>
</tr>
<tr>
<td>%job_name%</td>
<td>Name of the job that e-mail notifications are added for</td>
</tr>
<tr>
<td>%job_subname%</td>
<td>Present for event-based jobs with the parallel_instances attribute set and for chain steps</td>
</tr>
<tr>
<td>%event_type%</td>
<td>Valid values are listed in Table 143-84</td>
</tr>
<tr>
<td>%event_timestamp%</td>
<td>Time at which the event occurred</td>
</tr>
</tbody>
</table>
Table 143-15  (Cont.) Variables Used in the SUBJECT and BODY Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>%log_id%</td>
<td>Refers to the LOG_ID column in views *_SCHEDULER_JOB_LOG and *_SCHEDULER_JOB_RUN_DETAILS</td>
</tr>
<tr>
<td>%error_code%</td>
<td>Number of the error code.</td>
</tr>
<tr>
<td>%error_message%</td>
<td>The text of the error message</td>
</tr>
<tr>
<td>%run_count%</td>
<td>Run count for the job when the event was raised</td>
</tr>
<tr>
<td>%failure_count%</td>
<td>Failure count for the job when the event was raised</td>
</tr>
<tr>
<td>%retry_count%</td>
<td>Retry count for the job when the event was raised</td>
</tr>
</tbody>
</table>

Usage Notes

You can call ADD_JOB_EMAIL_NOTIFICATION once for each different set of notifications that you want to configure for a particular job. For example, you may want to send notifications for the JOB_FAILED, JOB_BROKEN, JOB_SCH_LIM_REACHED, and JOB_CHAIN_STALLED events to the principle DBA and all senior DBAs, but send a notification for the JOB_OVER_MAX_DUR event only to the principle DBA.

This procedure succeeds only if the Scheduler attribute email_server is set to a valid SMTP server. See Oracle Database Administrator's Guide for more information.

To call this procedure, you must be the job owner or have the CREATE ANY JOB system privilege or have the ALTER object privilege on the job.

143.6.4 ADD_TO_INCOMPATIBILITY Procedure

This procedure adds jobs or programs to an existing incompatibility definition.

Syntax

```sql
DBMS_SCHEDULER.ADD_TO_INCOMPATIBILITY (  
    incompatibility_name    IN VARCHAR2,  
    object_name             IN VARCHAR2);  
```

Parameters

Table 143-16  ADD_TO_INCOMPATIBILITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incompatibility_name</td>
<td>The name of the incompatibility definition.</td>
</tr>
<tr>
<td>object_name</td>
<td>One or more (comma-separated) programs or jobs</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure does not raise an error if any specified objects already exist in the incompatibility definition.
143.6.5 ALTER_CHAIN Procedure

This procedure alters an attribute of the specified steps of a chain. This affects all future runs of the specified steps, both in the currently running chain job and in future runs of the same chain job or other chain jobs that point to the chain.

Syntax

Alters the value of a boolean attribute of one or more steps:

```
DBMS_SCHEDULER.ALTER_CHAIN (
    chain_name   IN VARCHAR2,
    step_name    IN VARCHAR2,
    attribute    IN VARCHAR2,
    value        IN BOOLEAN);
```

Alters the value of a character attribute of one or more steps:

```
DBMS_SCHEDULER.ALTER_CHAIN (
    chain_name   IN VARCHAR2,
    step_name    IN VARCHAR2,
    attribute    IN VARCHAR2,
    char_value   IN VARCHAR2);
```

Parameters

Table 143-17 ALTER_CHAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain to alter</td>
</tr>
<tr>
<td>step_name</td>
<td>The name of the step or a comma-separated list of steps to alter. This cannot be NULL.</td>
</tr>
</tbody>
</table>
Table 143-17 (Cont.) ALTER_CHAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>The attribute of the steps to change. Must be one of the following:</td>
</tr>
<tr>
<td></td>
<td>· 'PAUSE'</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE for a step, after the step has run, its state changes to PAUSED (and the completed attribute remains FALSE).</td>
</tr>
<tr>
<td></td>
<td>If PAUSE is reset to FALSE for a paused chain step (using ALTER_RUNNING_CHAIN), the state is set to its completion state (SUCCEEDED, FAILED, or STOPPED) and the completed attribute is set to TRUE.</td>
</tr>
<tr>
<td></td>
<td>Setting PAUSE has no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps.</td>
</tr>
<tr>
<td></td>
<td>· 'PAUSED_BEFORE'</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE for a step and if any of the rule conditions that start the step are true, then its state changes to PAUSED and the step does not run.</td>
</tr>
<tr>
<td></td>
<td>If PAUSE_BEFORE is reset to FALSE for a chain step that has paused before starting (using ALTER_RUNNING_CHAIN), then the step starts running if any of the rule conditions that start the step are true.</td>
</tr>
<tr>
<td></td>
<td>Setting PAUSE_BEFORE has no effect on steps that are running or have already run. This allows execution of a chain to be suspended before the execution of certain steps.</td>
</tr>
<tr>
<td></td>
<td>· 'SKIP'</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met, the step immediately changes to state PAUSED.</td>
</tr>
<tr>
<td></td>
<td>· 'RESTART_ON_FAILURE'</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE for a step and the step fails due to an application error, then the step is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the chain step is marked FAILED.</td>
</tr>
<tr>
<td></td>
<td>If set to FALSE (the default), a failed chain step is immediately marked FAILED.</td>
</tr>
<tr>
<td></td>
<td>· 'RESTART_ON_RECOVERY'</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE for a step and the step is stopped by a database shutdown, then the step is restarted when the database is recovered.</td>
</tr>
<tr>
<td></td>
<td>If set to FALSE, and the step is stopped by a database shutdown, then the step is marked as stopped when the database is recovered and the chain continues.</td>
</tr>
<tr>
<td></td>
<td>· 'DESTINATION_NAME'</td>
</tr>
<tr>
<td></td>
<td>The name of an existing database destination or external destination. You can view external destination names in the view ALL_SCHEDULER_EXTERNAL_DESTS, and database destination names in the views * _SCHEDULER_DB_DESTS. You cannot specify a destination group for this attribute. This parameter is NULL by default.</td>
</tr>
<tr>
<td></td>
<td>· 'CREDENTIAL_NAME'</td>
</tr>
</tbody>
</table>
|                 | The credential to use when running this step. NULL by default.
Table 143-17  (Cont.) ALTER_CHAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value to set for the attribute (for a boolean attribute).</td>
</tr>
<tr>
<td>char_value</td>
<td>The value to set for the attribute (for a character attribute).</td>
</tr>
</tbody>
</table>

Usage Notes

Altering a chain requires ALTER privileges on the chain either by being the owner of the chain, or by having the ALTER object privilege on the chain or by having the CREATE ANY JOB system privilege.

143.6.6 ALTER_RUNNING_CHAIN Procedure

This procedure alters an attribute of the specified steps of a chain. This affects only steps of the instance of the chain for the specified running chain job.

Syntax

```sql
DBMS_SCHEDULER.ALTER_RUNNING_CHAIN (  
    job_name                IN VARCHAR2,  
    step_name               IN VARCHAR2,  
    attribute               IN VARCHAR2,  
    value                   IN {BOOLEAN|VARCHAR2});
```

Parameters

Table 143-18  ALTER_RUNNING_CHAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of the job that is running the chain</td>
</tr>
<tr>
<td>step_name</td>
<td>The name of the step or a comma-separated list of steps to alter. If this is set to NULL and attribute is PAUSE or SKIP, then all steps of the running chain are altered.</td>
</tr>
</tbody>
</table>
Table 143-18  (Cont.) ALTER_RUNNING_CHAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>The attribute of the steps to change. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>· 'PAUSE'</td>
</tr>
<tr>
<td></td>
<td>If the PAUSE attribute is set TRUE for a step, then after the step runs, its state changes to PAUSED (and the completed attribute remains false).</td>
</tr>
<tr>
<td></td>
<td>If PAUSE is reset to FALSE for a paused chain step (using ALTER_RUNNING_CHAIN), the state is set to completion (SUCCEEDED, FAILED, or STOPPED) and the completed attribute is set to TRUE. Setting PAUSE has no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps. If step_name is set to NULL, PAUSE is set to TRUE for all steps of this running chain.</td>
</tr>
<tr>
<td></td>
<td>· 'PAUSE_BEFORE'</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE for a step that has not yet run and if any of the rule conditions that start the step are true, then its state changes to PAUSED and the step does not run.</td>
</tr>
<tr>
<td></td>
<td>If PAUSE BEFORE is reset to FALSE for a chain step that has paused before starting, then the step starts running if any of the rule conditions that start the step are true.</td>
</tr>
<tr>
<td></td>
<td>Setting PAUSE BEFORE has no effect on steps that are running or have already run. This allows execution of a chain to be suspended before the execution of certain steps.</td>
</tr>
<tr>
<td></td>
<td>If step_name is set to NULL, then PAUSE BEFORE is set to the specified value for all steps of this running chain.</td>
</tr>
</tbody>
</table>
Table 143-18 (Cont.) ALTER_RUNNING_CHAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>'SKIP'</td>
</tr>
<tr>
<td>CONTINUED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the SKIP attribute is set to TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run. If step_name is set to NULL, SKIP is set TRUE for all steps of this running chain. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met the step immediately changes to state PAUSED.</td>
</tr>
<tr>
<td></td>
<td>'RESTART_ON_FAILURE'</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE for a step and the step fails due to an application error, then the step is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the chain step is marked FAILED. If set to FALSE (the default), a failed chain step is immediately marked FAILED.</td>
</tr>
<tr>
<td></td>
<td>'RESTART_ON_RECOVERY'</td>
</tr>
<tr>
<td></td>
<td>If the RESTART_ON_RECOVERY attribute is set to TRUE for a step, then if the step is stopped by a database shutdown, it is restarted when the database is recovered. If set to FALSE, then if the step is stopped by a database shutdown, the step is marked as stopped when the database is recovered and the chain continues.</td>
</tr>
<tr>
<td></td>
<td>'STATE'</td>
</tr>
<tr>
<td></td>
<td>This changes the state of the steps. The state can only be changed if the step is not running. The state can only be changed to one of the following:</td>
</tr>
<tr>
<td></td>
<td>'NOT_STARTED', 'SUCCEEDED', 'FAILED error_code'</td>
</tr>
<tr>
<td></td>
<td>If the state is being changed to FAILED, an error code must be included (this must be a positive integer).</td>
</tr>
<tr>
<td>value</td>
<td>The value to set for the attribute. Valid values are: TRUE, FALSE, 'NOT_STARTED', 'SUCCEEDED', or 'FAILED error_code'</td>
</tr>
</tbody>
</table>

Usage Notes

Altering a running chain requires you to have alter privileges on the job that is running (either as the owner, or as a user with ALTER privileges on the job or the CREATE ANY JOB system privilege).

143.6.7 CLOSE_WINDOW Procedure

This procedure closes an open window prematurely. A closed window means that it is no longer in effect. When a window is closed, the Scheduler switches the resource plan to the
one that is in effect outside the window, or in the case of overlapping windows, to
another window.

Syntax

```
DBMS_SCHEDULER.CLOSE_WINDOW (  
    window_name             IN VARCHAR2);
```

Parameters

Table 143-19 CLOSE_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_name</td>
<td>The name of the window</td>
</tr>
</tbody>
</table>

Usage Notes

If you try to close a window that does not exist or is not open, an error is generated.

A job that is running does not stop when the window it is running in closes, unless the
attribute stop_on_window_close is set to TRUE for the job. However, the resources
allocated to the job can change if the resource plan changes.

When a running job has a group of type WINDOW as its schedule, the job is not stopped
when its window is closed if another window in the same window group becomes
active. This is the case even if the job has the attribute stop_on_window_close set to
TRUE.

Closing a window requires the MANAGE_SCHEDULER privilege.

143.6.8 COPY_JOB Procedure

This procedure copies all attributes of an existing job to a new job. The new job is
created disabled, while the state of the existing job is unaltered.

Syntax

```
DBMS_SCHEDULER.COPY_JOB (  
    old_job                IN VARCHAR2,  
    new_job                IN VARCHAR2);  
```

Parameters

Table 143-20 COPY_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_job</td>
<td>The name of the existing job</td>
</tr>
<tr>
<td>new_job</td>
<td>The name of the new job</td>
</tr>
</tbody>
</table>

Usage Notes

To copy a job, you must have privileges to create a job in the schema of the new job
(the CREATE JOB system privilege if it is in your own schema, otherwise, the CREATE ANY
JOB system privilege). If the old job is not in the your own schema, then you must also have ALTER privileges on the old job or the CREATE ANY JOB system privilege.

### 143.6.9 CREATE_CHAIN Procedure

This procedure creates a new chain. The chain name can be optionally qualified with a schema name (for example, myschema.mynname).

A chain is always created as disabled and must be enabled with the **ENABLE Procedure** before it can be used.

**Syntax**

```sql
DBMS_SCHEDULER.CREATE_CHAIN (
    chain_name              IN VARCHAR2,
    rule_set_name           IN VARCHAR2 DEFAULT NULL,
    evaluation_interval     IN INTERVAL DAY TO SECOND DEFAULT NULL,
    comments                IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name to assign to the new chain, which can optionally be qualified with a schema. This must be unique in the SQL namespace, therefore, there cannot already be a table or other object with this name and schema.</td>
</tr>
<tr>
<td>rule_set_name</td>
<td>In the normal case, no rule set should be passed in. The Scheduler automatically creates a rule set and associated empty evaluation context. You then use <strong>DEFINE_CHAIN_RULE</strong> to add rules and <strong>DROP_CHAIN_RULE</strong> to remove them. Advanced users can create a rule set that describes their chain dependencies and pass it in here. This allows greater flexibility in defining rules. For example, conditions can refer to external variables, and tables can be exposed through the evaluation context. If you pass in a rule set, you must ensure that it is in the format of a chain rule set. (For example, all steps must be listed as variables in the evaluation context). If no rule set is passed in, the rule set created is of the form <strong>SCHED_RULESET{N}</strong> and the evaluation context created is of the form <strong>SCHED_EVCTX{N}</strong>. See Oracle Streams Concepts and Administration for information on rules and rule sets.</td>
</tr>
<tr>
<td>evaluation_interval</td>
<td>If this is NULL, reevaluation of the rules of a running chain are performed only when the job starts and when a step completes. A non-NULL value causes rule evaluations to also occur periodically at the specified interval. Because evaluation may be CPU-intensive, this should be conservatively set to the highest possible value or left at NULL if possible. <strong>evaluation_interval</strong> cannot be less than a minute or greater than a day.</td>
</tr>
<tr>
<td>comments</td>
<td>An optional comment describing the purpose of the chain</td>
</tr>
</tbody>
</table>

**Usage Notes**

To create a chain in your own schema, you must have the **CREATE JOB** system privilege. To create a chain in a different schema you must have the **CREATE ANY JOB** system privilege. If you do not provide a **rule_set_name**, a rule set and evaluation context is created in the schema that the chain is being created in, so you must have the privileges required to create
these objects. See the \texttt{DBMS\_RULE\_ADM\_CREATE\_RULE\_SET} and \texttt{DBMS\_RULE\_ADM\_CREATE\_EVALUATION\_CONTEXT} procedures for more information.

143.6.10 CREATE\_CREDENTIAL Procedure

This deprecated procedure creates a stored username/password pair. Credentials are assigned to jobs so that they can authenticate with a local or remote host operating system or a remote Oracle database.

\textbf{Note:}

This procedure is deprecated with Oracle Database 12c Release 1 (12.1). While the procedure remains available in this package, for reasons of backward compatibility, Oracle recommends using the alternative enhanced functionality provided in the \texttt{DBMS\_CREDENTIAL} package, specifically the \texttt{CREATE\_CREDENTIAL} Procedure.

\textbf{Syntax}

\begin{verbatim}
DBMS\_SCHEDULER\_CREATE\_CREDENTIAL (  
    credential_name         IN VARCHAR2,  
    username                IN VARCHAR2,  
    password                IN VARCHAR2,  
    database_role           IN VARCHAR2 DEFAULT NULL,  
    windows_domain          IN VARCHAR2 DEFAULT NULL,  
    comments                IN VARCHAR2 DEFAULT NULL);
\end{verbatim}

\textbf{Parameters}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential_name</td>
<td>The name to assign to the credential. It can optionally be prefixed with a schema name. It cannot be set to NULL. It is converted to uppercase unless enclosed in double quotation marks.</td>
</tr>
<tr>
<td>username</td>
<td>The user name for logging into the host operating system or remote Oracle database. This cannot be set to NULL and is case-sensitive. It cannot contain double quotes or spaces. Maximum length is 64.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the user name. This cannot be set to NULL and is case sensitive. The password is stored obfuscated and is not displayed in the Scheduler dictionary views. Maximum length is 128.</td>
</tr>
<tr>
<td>database_role</td>
<td>The value of the database_role attribute is used as the system privilege for logging into a remote database to run a remote database job. Valid values are: SYSDBA and SYSOPER</td>
</tr>
<tr>
<td>windows_domain</td>
<td>For a Windows remote executable target, this is the domain that the specified user belongs to. The domain is converted to uppercase automatically. Maximum length is 64.</td>
</tr>
<tr>
<td>comments</td>
<td>A text string that can be used to describe the credential. Scheduler does not use this parameter. Maximum length is 240.</td>
</tr>
</tbody>
</table>
Usage Notes

Credentials reside in a particular schema and can be created by any user with the `CREATE JOB` system privilege. To create a credential in a schema other than your own, you must have the `CREATE ANY JOB` privilege.

143.6.11 CREATE_DATABASE_DESTINATION Procedure

This procedure creates a database destination. A database destination represents an Oracle database on which remote database jobs run.

The host that the remote database resides on must have a running Scheduler agent that is registered with the database that this procedure is called from.

Syntax

```sql
DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION (
    destination_name        IN VARCHAR2,
    agent                   IN VARCHAR2,
    tns_name                IN VARCHAR2,
    comments                IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_name</td>
<td>The name to assign to the database destination. It can optionally be prefixed with a schema name. Cannot be NULL. It is converted to uppercase unless enclosed in double quotation marks.</td>
</tr>
<tr>
<td>agent</td>
<td>The external destination name of the Scheduler agent to connect. Equivalent to an agent name.</td>
</tr>
<tr>
<td>tns_name</td>
<td>An Oracle Net connect identifier that is resolved to the Oracle database instance being connected to. The exact syntax depends on the Oracle Net configuration. The connect identifier can be a complete Oracle Net connect descriptor (network address and database service name) or a net service name, which is an alias for a connect descriptor. The alias must be resolved in the tnsnames.ora file on the local computer. The maximum size for tns_name is 2000 characters. If tns_name is NULL, the agent connects to the default Oracle database on its host. You specify the default database by assigning values to the ORACLE_HOME and ORACLE_SID parameters in the agent configuration file, schagent.conf, located in the agent home directory. See Oracle Database Net Services Administrator's Guide for more information on connect identifiers.</td>
</tr>
<tr>
<td>comments</td>
<td>A text string that describes the database destination. Scheduler does not use this argument.</td>
</tr>
</tbody>
</table>
Usage Notes

Database destinations reside in a particular schema and can be created by any user with the CREATE JOB system privilege. To create a database destination in a schema other than your own, you must have the CREATE ANY JOB privilege.

143.6.12 CREATE_EVENT_SCHEDULE Procedure

This procedure creates an event schedule, which is used to start a job when a particular event is raised.

Syntax

```
DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
    schedule_name           IN VARCHAR2,
    start_date              IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    event_condition         IN VARCHAR2 DEFAULT NULL,
    queue_spec              IN VARCHAR2,
    end_date                IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    comments                IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 143-24 CREATE_EVENT_SCHEDULE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_name</td>
<td>The name to assign to the schedule. The name must be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.</td>
</tr>
<tr>
<td>start_date</td>
<td>This attribute specifies the date and time that this schedule becomes valid. Occurrences of the event before this date and time are ignored in the context of this schedule.</td>
</tr>
<tr>
<td>event_condition</td>
<td>This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression, provided that the message payload is an object type, and that you prefix object attributes in the expression with <code>tab.user_data</code>. For more information on rules, see the <code>DBMS_AQADM.ADD_SUBSCRIBER</code> procedure.</td>
</tr>
<tr>
<td>queue_spec</td>
<td>This argument specifies either a file watcher name or the queue into which events that start this particular job are enqueued (the source queue). If the source queue is a secure queue, the queue_spec argument is a string containing a pair of values of the form <code>queue_name, agent name</code>. For non-secure queues, only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner's schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue.</td>
</tr>
<tr>
<td>end_date</td>
<td>The date and time after which jobs do not run and windows do not open. An event schedule that has no end_date is valid forever. end_date must be after the start_date. If it is not, then an error is generated when the schedule is created.</td>
</tr>
</tbody>
</table>
Table 143-24 (Cont.) CREATE_EVENT_SCHEDULE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments</td>
<td>This attribute specifies an optional comment about the schedule. By default, this attribute is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

You must have the CREATE JOB privilege to create a schedule in your own schema or the CREATE ANY JOB privilege to create a schedule in someone else's schema by specifying schema.schedule_name. Once a schedule has been created, it can be used by other users. The schedule is created with access to PUBLIC. Therefore, there is no need to explicitly grant access to the schedule.

See Also:

"CREATE_FILE_WATCHER Procedure"

143.6.13 CREATE_FILE_WATCHER Procedure

This procedure creates a file watcher, which is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job. After you create a file watcher, you reference it in an event-based job or event schedule.

Syntax

```
DBMS_SCHEDULER.CREATE_FILE_WATCHER (  
   file_watcher_name            IN VARCHAR2,  
   directory_path               IN VARCHAR2,  
   file_name                    IN VARCHAR2,  
   credential_name              IN VARCHAR2,  
   destination                  IN VARCHAR2 DEFAULT NULL,  
   min_file_size                IN PLS_INTEGER DEFAULT 0,  
   steady_state_duration        IN INTERVAL DAY TO SECOND DEFAULT NULL,  
   comments                     IN VARCHAR2 DEFAULT NULL,  
   enabled                      IN BOOLEAN DEFAULT TRUE);  
```

Parameters

Table 143-25 CREATE_FILE_WATCHER Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_watcher_name</td>
<td>The name to assign to the file watcher. The name must be unique in the SQL namespace. For example, a file watcher cannot have the same name as a table in a schema. This can optionally be prefixed with a schema name. Cannot be NULL.</td>
</tr>
</tbody>
</table>
Table 143-25  (Cont.) CREATE_FILE_WATCHER Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory_path</td>
<td>Directory in which the file is expected to arrive. The single wildcard ‘?’ at the beginning of the path denotes the Oracle home path. For example, ‘?/rdbms/log’ denotes the rdbms/log subdirectory of the Oracle home directory.</td>
</tr>
<tr>
<td>file_name</td>
<td>Name of the file to look for. Two wildcards are permitted anywhere in the file name: ‘?’ denotes any single character, and ‘*’ denotes zero or more characters. This attribute cannot be NULL.</td>
</tr>
<tr>
<td>credential_name</td>
<td>Name of a valid credential object. The file watcher uses the credential to authenticate itself with the host operating system to access the watched-for file. The file watcher owner must have EXECUTE privileges on the credential. Cannot be NULL.</td>
</tr>
<tr>
<td>destination</td>
<td>Name of an external destination. You create an external destination by registering a remote Scheduler agent with the database. See the view ALL_SCHEDULER_EXTERNAL_DESTS for valid external destination names. If this parameter is NULL, the file watcher is created on the local host.</td>
</tr>
<tr>
<td>min_file_size</td>
<td>Minimum size in bytes that the file must be before the file watcher considers the file found. Default is 0.</td>
</tr>
<tr>
<td>steady_state_duration</td>
<td>Minimum time interval that the file must remain unchanged before the file watcher considers the file found. Cannot exceed one hour. If NULL, an internal value is used. The minimum value is 10 seconds. Oracle recommends similar steady_state_duration values for all file watchers for efficient file watcher job operation. Also, the repeat interval of the file watcher schedule must be equal or greater than the steady_state_duration value.</td>
</tr>
<tr>
<td>comments</td>
<td>Optional comment.</td>
</tr>
<tr>
<td>enabled</td>
<td>If TRUE (the default), the file watcher is enabled.</td>
</tr>
</tbody>
</table>

Usage Notes

You must have the CREATE JOB system privilege to create a file watcher in your own schema. You require the CREATE ANY JOB system privilege to create a file watcher in a schema different from your own (except the SYS schema, which is disallowed).

143.6.14 CREATE_GROUP Procedure

This procedure creates a group. Groups contain members, which you can specify when you create the group or at a later time. There are three types of groups: window groups, database destination groups, and external destination groups.

You can use a group name in other DBMS_SCHEDULER package procedures to specify a list of objects. For example, to specify multiple destinations for a remote database job, you provide a group name for the DESTINATION_NAME parameter of the job.

Syntax

DBMS_SCHEDULER.CREATE_GROUP ( 
    group_name           IN VARCHAR2, 
    group_type           IN VARCHAR2, 
)
### Parameters

#### Table 143-26 CREATE_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_name</td>
<td>The name to assign to the group. It can optionally be prefixed with a schema name. It cannot be NULL. It is converted to uppercase unless enclosed in double quotation marks.</td>
</tr>
<tr>
<td>group_type</td>
<td>The type of members in the group. All members must be of the same type. Possible types are:</td>
</tr>
<tr>
<td></td>
<td>'DB_DEST' Database destination: Members are database destinations, for running remote database jobs.</td>
</tr>
<tr>
<td></td>
<td>'EXTERNAL_DEST' External destination: Members are external destinations, for running remote external jobs.</td>
</tr>
<tr>
<td></td>
<td>'WINDOW' Members are Scheduler windows. You must have the MANAGE SCHEDULER privilege to create a group of this type.</td>
</tr>
<tr>
<td>member</td>
<td>Optional comma-separated list of group members. The default is NULL. If NULL, use the ADD_GROUP_MEMBER procedure to add members. You can also use ADD_GROUP_MEMBER to add additional members at a later time. The keyword LOCAL can be used as a member in database destination groups and external destination groups.</td>
</tr>
<tr>
<td></td>
<td>In database destination groups, LOCAL represents the source database on which the job is created. It cannot be preceded with a credential.</td>
</tr>
<tr>
<td></td>
<td>In external destination groups, LOCAL represents the host on which the source database resides. It can be optionally preceded with a credential name. If no credential is provided, jobs that use this group as their destination must have a default credential.</td>
</tr>
<tr>
<td>comments</td>
<td>A text string that describes the group. Scheduler does not use this argument.</td>
</tr>
</tbody>
</table>
Usage Notes

Groups reside in a particular schema and can be created by any user with the CREATE JOB system privilege. To create a group in a schema other than your own, you must have the CREATE ANY JOB privilege. The group name must be unique among all Scheduler objects.

You can grant the SELECT or READ privilege on a group so that other users can reference the group when creating jobs or schedules. To enable other users to modify a group, you can grant the ALTER privilege on the group.

Each group member must be unique within the group. For destination groups, the credential/destination name pairs must be unique within the group. An error is generated if any of the group members do not exist. For destination groups, both the credential and destination portions of a member must exist.

Another group of the same type can be a group member. The Scheduler immediately expands the included group name into its list of members.

Groups are created enabled, but you can disable them.

Example

The following PL/SQL block creates a group named production_dest1, whose members are database destinations for a collection of production databases.

```plsql
BEGIN
    DBMS_SCHEDULER.CREATE_GROUP(
        GROUP_NAME    => 'production_dest1',
        GROUP_TYPE    => 'DB_DEST',
        MEMBER        => 'LOCAL, oracle_cred@prodhost1, prodhost2',
        COMMENTS      => 'All sector1 production machines');
END;
```

143.6.15 CREATE_INCOMPATIBILITY Procedure

This procedure creates an incompatibility definition.

Syntax

```plsql
DBMS_SCHEDULER.CREATE_INCOMPATIBILITY (
    incompatibility_name    IN VARCHAR2,
    object_name             IN VARCHAR2,
    constraint_level        IN VARCHAR2 DEFAULT 'JOB_LEVEL',
    enabled                 IN BOOLEAN DEFAULT TRUE,
    comments                IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 143-27  CREATE_INCOMPATIBILITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incompatibility_name</td>
<td>The name of the incompatibility definition.</td>
</tr>
<tr>
<td>object_name</td>
<td>One or more (comma-separated) programs or jobs.</td>
</tr>
</tbody>
</table>
Table 143-27  (Cont.) CREATE_INCOMPATIBILITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint_level</td>
<td>One or more (comma-separated) programs or jobs.</td>
</tr>
<tr>
<td>enabled</td>
<td>Specifies whether the constraint is initially enabled (true) or not enabled (false).</td>
</tr>
<tr>
<td>comments</td>
<td>Optional descriptive comment.</td>
</tr>
</tbody>
</table>

Usage Notes

If `object_name` contains multiple (comma-separated) values, they must be either all programs or all jobs that are incompatible with each other (that is, they cannot be run at the same time). For jobs, the list must consist of two or more jobs, and `constraint_level` must be `JOB_LEVEL`. For programs, `constraint_level` can be either `JOB_LEVEL` or `PROGRAM_LEVEL`. When set to the default value `JOB_LEVEL`, only a single job that is based on the program (or programs) mentioned in `object_name` can run at the same time. When `constraint_level` is set to `PROGRAM_LEVEL`, the programs are incompatible, but the jobs based on the same program are not incompatible.

For example, if the value of `object_name` is `P1,P2,P3` and `constraint_level` is `PROGRAM_LEVEL`, many jobs based on P1 can be running at the same time, but if any P1 based job is running, none based on P2 or P3 can be running. Or, similarly, many jobs based on P3 can be running at the same time, but none based on P1 or P2. If `constraint_level` is set to `JOB_LEVEL`, then only a single job out of all the jobs based on programs P1, P2 and P3 can be running at a time.

See Also:

Using Incompatibility Definitions in Oracle Database Administrator's Guide

143.6.16 CREATE_JOB Procedure

This procedure creates a single job.

If you create the job as enabled by setting the `enabled` attribute to `TRUE`, the Scheduler automatically runs the job according to its schedule. If you create the job disabled, the job does not run until you enable it with the `SET_ATTRIBUTE` Procedure.

The procedure is overloaded. The different functionality of each form of syntax is presented along with the syntax declaration.

Syntax

Creates a job in a single call without using an existing program or schedule:

```sql
DBMS_SCHEDULER.CREATE_JOB (  
  job_name             IN VARCHAR2,  
  job_type             IN VARCHAR2,  
  job_action           IN VARCHAR2,  
  number_of_arguments  IN PLS_INTEGER DEFAULT 0,  
  start_date           IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,  
)  
```
repeat_interval     IN VARCHAR2     DEFAULT NULL,
end_date             IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
job_class            IN VARCHAR2     DEFAULT 'DEFAULT_JOB_CLASS',
enabled              IN BOOLEAN      DEFAULT FALSE,
auto_drop            IN BOOLEAN      DEFAULT TRUE,
comments             IN VARCHAR2     DEFAULT NULL,
credential_name      IN VARCHAR2     DEFAULT NULL,
destination_name     IN VARCHAR2     DEFAULT NULL);

Creates a job using a named schedule object and a named program object:

DBMS_SCHEDULER.CREATE_JOB (  
  job_name                IN VARCHAR2,
  program_name            IN VARCHAR2,
  schedule_name           IN VARCHAR2,
  job_class               IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
  enabled                 IN BOOLEAN               DEFAULT FALSE,
  auto_drop               IN BOOLEAN               DEFAULT TRUE,
  comments                IN VARCHAR2              DEFAULT NULL,
  job_style               IN VARCHAR2              DEFAULT 'REGULAR',
  credential_name         IN VARCHAR2              DEFAULT NULL,
  destination_name        IN VARCHAR2              DEFAULT NULL);

Creates a job using a named program object and an inlined schedule:

DBMS_SCHEDULER.CREATE_JOB (  
  job_name             IN VARCHAR2,
  program_name         IN VARCHAR2,
  start_date           IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  repeat_interval      IN VARCHAR2                 DEFAULT NULL,
  end_date             IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  job_class            IN VARCHAR2                 DEFAULT 'DEFAULT_JOB_CLASS',
  enabled              IN BOOLEAN                  DEFAULT FALSE,
  auto_drop            IN BOOLEAN                  DEFAULT TRUE,
  comments             IN VARCHAR2                 DEFAULT NULL,
  job_style            IN VARCHAR2                 DEFAULT 'REGULAR',
  credential_name      IN VARCHAR2                 DEFAULT NULL,
  destination_name     IN VARCHAR2                 DEFAULT NULL);

Creates a job using a named schedule object and an inlined program:

DBMS_SCHEDULER.CREATE_JOB (  
  job_name                IN VARCHAR2,
  schedule_name           IN VARCHAR2,
  job_type                IN VARCHAR2,
  job_action              IN VARCHAR2,
  number_of_arguments     IN PLS_INTEGER       DEFAULT 0,
  job_class               IN VARCHAR2          DEFAULT 'DEFAULT_JOB_CLASS',
  enabled                 IN BOOLEAN           DEFAULT FALSE,
  auto_drop               IN BOOLEAN           DEFAULT TRUE,
  comments                IN VARCHAR2          DEFAULT NULL,
  credential_name         IN VARCHAR2          DEFAULT NULL,
  destination_name        IN VARCHAR2          DEFAULT NULL);

Creates a job using an inlined program and an event:

DBMS_SCHEDULER.CREATE_JOB (  
  job_name                IN VARCHAR2,
  job_type                IN VARCHAR2,
  job_action              IN VARCHAR2,
  number_of_arguments     IN PLS_INTEGER       DEFAULT 0,
  start_date              IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
Creates a job using a named program object and an event:

```sql
DBMS_SCHEDULER.CREATE_JOB (  
  job_name                IN VARCHAR2,  
  program_name            IN VARCHAR2,  
  start_date              IN TIMESTAMP WITH TIME ZONE,  
  event_condition         IN VARCHAR2,  
  queue_spec              IN VARCHAR2,  
  end_date                IN TIMESTAMP WITH TIME ZONE,  
  job_class               IN VARCHAR2          DEFAULT 'DEFAULT_JOB_CLASS',  
  enabled                 IN BOOLEAN           DEFAULT FALSE,  
  auto_drop               IN BOOLEAN           DEFAULT TRUE,  
  comments                IN VARCHAR2          DEFAULT NULL,  
  credential_name         IN VARCHAR2          DEFAULT NULL,  
  destination_name        IN VARCHAR2          DEFAULT NULL);
```

### Parameters

**Table 143-28 CREATE_JOB Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name to assign to the job. The name must be unique in the SQL namespace. For example, a job cannot have the same name as a table in a schema. If the job being created will reside in another schema, it must be qualified with the schema name. If job_name is not specified, an error is generated. If you want to have a name generated by the Scheduler, you can use the GENERATE_JOB_NAME procedure to generate a name and then use the output in the CREATE_JOB procedure. The GENERATE_JOB_NAME procedure generates a number from a sequence, which is the job name. You can prefix the number with a string. The job name will then be the string with the number from the sequence appended to it. See &quot;GENERATE_JOB_NAME Function&quot; for more information.</td>
</tr>
</tbody>
</table>
Table 143-28  (Cont.) CREATE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_type</td>
<td>This attribute specifies the type of job that you are creating. If it is not specified, an error is generated. See job_action in the next row for related information. The supported values are:</td>
</tr>
<tr>
<td></td>
<td>· <strong>'PLSQL_BLOCK'</strong></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is an anonymous PL/SQL block. Job or program arguments are not supported when the job or program type is <strong>PLSQL_BLOCK</strong>. In this case, the number of arguments must be 0.</td>
</tr>
<tr>
<td></td>
<td>· <strong>'STORED_PROCEDURE'</strong></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported.</td>
</tr>
<tr>
<td></td>
<td>· <strong>'EXECUTABLE'</strong></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is going to be run outside the database using an external executable. External jobs are anything that can be executed from the command line of the operating system. Any data arguments are not supported with a job or program type of <strong>EXECUTABLE</strong>. The job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.</td>
</tr>
<tr>
<td></td>
<td>· <strong>'CHAIN'</strong></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is a chain. Arguments are not supported for a chain, so number_of_arguments must be 0.</td>
</tr>
<tr>
<td></td>
<td>· <strong>'EXTERNAL_SCRIPT'</strong></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is an external script that uses the command shell of the computer running the job. For Windows this is cmd.exe and for UNIX based systems the sh shell, unless a different interpreter is specified by prefixing the first line of the script with <code>#!</code>.</td>
</tr>
<tr>
<td></td>
<td>· <strong>'SQL_SCRIPT'</strong></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is a SQL<em>Plus script. The job must point to a credential that contains a valid operating system username and password. The SQL</em>Plus script is run by the SQL<em>Plus executable. The job may point to a connect credential that contains a database credential. If so, this credential is used to connect to the database before running the SQL</em>Plus script. Note that if you choose to use connect credential, you must use set_attribute to specify the Connect_Credential_Name attribute. If you do not have connect credential, you must include an explicit SQL*Plus connect statement providing a valid database userid / password. The job owner must have the CREATE EXTERNAL JOB system privilege.</td>
</tr>
<tr>
<td></td>
<td>· <strong>'BACKUP_SCRIPT'</strong></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is an RMAN backup script. The script runs a connect statement that uses either a password or OS authentication before it executes any target commands. The job points to a credential that contains a valid operating system username and password. The RMAN session runs under this operating system user. The Scheduler uses the RMAN executable from the current Oracle home to run the script and throws an error if this is missing. The job owner must have the CREATE EXTERNAL JOB system privilege.</td>
</tr>
</tbody>
</table>
Table 143-28  (Cont.) CREATE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_action</td>
<td>This attribute specifies the action of the job. If job_action is not specified for an inline program, then an error is generated when creating the job. The job action is executed inside an autonomous transaction, and all autonomous transaction guidelines and restrictions apply. For example, online DDL operations are not allowed inside an autonomous transaction, and therefore cannot be used in the job action. The following actions are possible:</td>
</tr>
<tr>
<td></td>
<td>- For a PL/SQL block: The action is to execute PL/SQL code. These blocks must end with a semicolon. For example, my_proc(); or BEGIN my_proc(); END; or DECLARE arg pls_integer:= 10; BEGIN my_proc2(arg); END; Note that the Scheduler wraps job_action in its own block and passes the following to PL/SQL for execution: DECLARE ... BEGIN job_action END; This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except event_message in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value. See Table 143-40 for details on available metadata attributes.</td>
</tr>
<tr>
<td></td>
<td>- For a stored procedure: The action is the name of the stored procedure. You have to specify the schema if the procedure resides in another schema than the job. If case sensitivity is needed, enclose the schema name and the store procedure name in double quotes. For example, job_action_action=&gt;&quot;Schema:Procedure&quot;. PL/SQL procedures with INOUT or OUT arguments are not supported as job_action when the job or program type is STOREDPROCEDURE.</td>
</tr>
<tr>
<td></td>
<td>- For an executable: The action is the name of the external executable, including the full path name, but excluding any command-line arguments. If the action starts with a single question mark (?), the question mark is replaced by the path to the Oracle home directory for a local job or to the Scheduler agent home for a remote job. If the action contains an at-sign (@) and the job is local, the at-sign is replaced with the SID of the current Oracle instance. NOTE: Shell script syntax is not supported, only syntax for the name of and path to an executable is supported.</td>
</tr>
<tr>
<td></td>
<td>- For a chain: The action is the name of a Scheduler chain object. You must specify the schema of the chain if it resides in a different schema than the job.</td>
</tr>
<tr>
<td></td>
<td>- For an external script: The job_action must be either the path to an operating system script or an inline operating system script. If the job_action is a path to a script, then the script must reside on every computer that the job runs on. The job_action may contain calls to SQL*Plus or RMAN executables directly, without having to specify its full path, given that they are stored on their default location for every computer that runs the job. The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called. The job must point to a credential that contains a valid operating system username and password.</td>
</tr>
<tr>
<td></td>
<td>- For a SQL script:</td>
</tr>
</tbody>
</table>
Table 143-28  (Cont.) CREATE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_action</td>
<td>The <strong>job_action</strong> must be either the path to a SQL<em>Plus script or an inline SQL</em>Plus script. If the <strong>job_action</strong> is a path to a script, then the script must reside on every computer that the job runs on. The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called. If the arguments are named, they are also bound to named variables in the SQL*Plus session. For a backup script: <strong>job_action</strong> is either the path to a RMAN script or an inline RMAN script. If the <strong>program_action</strong> is a path to a script, then the script must reside on every computer that the program runs on. The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called.</td>
</tr>
<tr>
<td>number_of_arguments</td>
<td>This attribute specifies the number of arguments that the job expects. The range is 0-255, with the default being 0.</td>
</tr>
<tr>
<td>program_name</td>
<td>The name of the program associated with this job. If the program is of type EXECUTABLE, the job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.</td>
</tr>
<tr>
<td>start_date</td>
<td>This attribute specifies the first date and time on which this job is scheduled to start. If <strong>start_date</strong> and <strong>repeat_interval</strong> are left null, then the job is scheduled to run as soon as the job is enabled. For repeating jobs that use a calendaring expression to specify the repeat interval, <strong>start_date</strong> is used as a reference date. The first time the job runs is the first match of the calendaring expression that is on or after the current date and time. The Scheduler cannot guarantee that a job executes on an exact time because the system may be overloaded and thus resources unavailable.</td>
</tr>
<tr>
<td>event_condition</td>
<td>This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression provided that the message payload is an object type, and that you prefix object attributes in the expression with <code>tab.user_data</code>. For more information on rules, see the DBMS_AQADM.ADD_SUBSCRIBER procedure.</td>
</tr>
<tr>
<td>queue_spec</td>
<td>This argument specifies either of the following: The source queue where events that start this particular job are enqueued. If it is secure, then the <strong>queue_spec</strong> argument is a pair of values of the form <strong>queue_name</strong>, <strong>agent_name</strong>. If it is not secure, then only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner’s schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue. A file watcher name. For more information on this option, see Oracle Database Administrator’s Guide.</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>This attribute specifies how often the job repeats. You can specify the repeat interval by using calendaring or PL/SQL expressions. The expression specified is evaluated to determine the next time the job should run. If <strong>repeat_interval</strong> is not specified, the job runs only once at the specified start date. See “Calendaring Syntax” for further information.</td>
</tr>
<tr>
<td>schedule_name</td>
<td>The name of the schedule, window, or window group associated with this job.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>job_class</td>
<td>The class this job is associated with.</td>
</tr>
<tr>
<td>end_date</td>
<td>This attribute specifies the date and time after which the job expires and is no longer run. After the end_date, if auto_drop is TRUE, the job is dropped. If auto_drop is FALSE, the job is disabled and the state of the job is set to completed. If no value for end_date is specified, the job repeats forever unless max_runs or max_failures is set, in which case the job stops when either value is reached. The value for end_date must be after the value for start_date. If end_date is less than start_date, then an error will be generated. If end_date is the same as start_date, then the job will not execute and no error will be generated.</td>
</tr>
<tr>
<td>comments</td>
<td>This attribute specifies a comment about the job. By default, this attribute is NULL.</td>
</tr>
<tr>
<td>job_style</td>
<td>Style of the job being created. This argument can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>- 'REGULAR' creates a regular job. This is the default.</td>
</tr>
<tr>
<td></td>
<td>- 'LIGHTWEIGHT' creates a lightweight job. This value is permitted only when the job references a program object. Use lightweight jobs when you have many short-duration jobs that run frequently. Under certain circumstances, using lightweight jobs can deliver a small performance gain.</td>
</tr>
<tr>
<td></td>
<td>- 'IN_MEMORY_RUNTIME' creates an in-memory runtime job. These jobs are based on lightweight job structures, so the same rules and restrictions apply; however, they further boost performance by keeping an in-memory cache, so they minimize disk access for pre-run and post-run actions.</td>
</tr>
<tr>
<td></td>
<td>- 'IN_MEMORY_FULL' creates an in-memory full job. In-memory full jobs require a program and cannot have a schedule or repeat interval. They run automatically when the job is enabled, and after running they are discarded. They keep all the job information in memory and are not backed up on disk, meaning that they are lost when the instance is rebooted. They are designed to run actions that must be performed immediately with the least amount of overhead possible.</td>
</tr>
<tr>
<td>credential_name</td>
<td>The default credential to use with the job. Applicable only to remote database jobs, remote external jobs, local external jobs, script jobs, and event-based jobs that process file arrival events. The credential must exist. For local database jobs, it must be NULL. For local external jobs only, if this attribute is NULL (the default), then a preferred (default) credential is selected. See Oracle Database Administrator's Guide for information about preferred credentials for local external jobs. See also: &quot;CREATE_CREDENTIAL Procedure&quot;</td>
</tr>
</tbody>
</table>
Table 143-28  (Cont.) CREATE_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| destination_name | The database destination or external destination for the job. Use for remote database jobs and remote external jobs only. Must be NULL for jobs running on the local database or for local external jobs (executables). This attribute can be a single destination name or the name of a group of type 'EXTERNAL_DEST' or 'DB_DEST'. The single destination or group must already exist. The following applies to this attribute:  
  • If it is a database destination, it must have been created by the CREATE_DATABASE_DESTINATION Procedure.  
  • If it is an external destination, it must have been implicitly created by registering a remote Scheduler agent with the local database.  
  • If it is a group, each member of the group must exist, and the job must run on all destinations named in the group. See "CREATE_GROUP Procedure".  
  If the credential_name argument of CREATE_JOB is NULL, each destination must be preceded by a credential, in the following format: credential.destination  
  The credential must already exist. If the credential_name argument is provided, then it serves as the default credential for every destination that is not preceded by a credential.  
  You can query the views *_SCHEDULER_DB_DESTS and ALL_SCHEDULER_EXTERNAL_DESTS for existing destinations and *_SCHEDULER_GROUP_MEMBERS for existing groups and their members.  
  **destination_job attribute is deprecated in Oracle Database 11gR2 and superseded by destination_name.**  
  enabled       | This attribute specifies whether the job is created enabled or not. The possible settings are TRUE or FALSE. By default, this attribute is set to FALSE and, therefore, the job is created as disabled. A disabled job means that the metadata about the job has been captured, and the job exists as a database object. However, the Scheduler ignores the job and the job coordinator does not pick it for processing. In order for the job coordinator to process the job, the job must be enabled. You can enable a job by setting this argument to TRUE or by using the ENABLE procedure.  
  auto_drop     | This flag, if TRUE, causes a job to be automatically dropped after it has completed or has been automatically disabled. A job is considered completed if:  
  • Its end date (or the end date of the job schedule) has passed. Note that a job with a Window schedule will not be auto-dropped when the window closes, because this is not considered to be the end of the Window.  
  • It has run max_runs number of times. max_runs must be set with SET_ATTRIBUTE.  
  • It is not a repeating job and has run once.  
  A job is disabled when it has failed max_failures times. max_failures is also set with SET_ATTRIBUTE.  
  If this flag is set to FALSE, the jobs are not dropped and their metadata is kept until the job is explicitly dropped with the DROP_JOB procedure.  
  By default, jobs are created with auto_drop set to TRUE.  

---
Usage Notes

Jobs are created as disabled by default. You must explicitly enable them so that they will become active and scheduled. Before enabling a job, ensure that all program arguments, if any, are defined, either by defining default values in the program object or by supplying values with the job.

The JOB_QUEUE_PROCESSES initialization parameter specifies the maximum number of processes that can be created for the execution of jobs. Beginning with Oracle Database 11g Release 2, JOB_QUEUE_PROCESSES applies to DBMS_SCHEDULER jobs. Setting this parameter to 0 disables DBMS_SCHEDULER jobs.

To create a job in your own schema, you need to have the CREATE JOB privilege. A user with the CREATE ANY JOB privilege can create a job in any schema. If the job being created will reside in another schema, the job name must be qualified with the schema name. For a job of type EXECUTABLE (or for a job that points to a program of type EXECUTABLE), the job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.

Associating a job with a particular class or program requires EXECUTE privileges for that class or program.

Not all possible job attributes can be set with CREATE_JOB. Some must be set after the job is created. For example, job arguments must be set with the SET_JOB_ARGUMENT_VALUE Procedure or the SET_JOB_ANYDATA_VALUE Procedure. Other job attributes, such as job_priority and max_runs, are set with the SET_ATTRIBUTE Procedure.

To create multiple jobs efficiently, use the CREATE_JOBS procedure.

Note:
The Scheduler runs event-based jobs for each occurrence of an event that matches the event condition of the job. However, events that occur while the job is already running are ignored; the event gets consumed, but does not trigger another run of the job.

143.6.17 CREATE_JOB_CLASS Procedure

This procedure creates a job class. Job classes are created in the SYS schema.

Syntax

```
DBMS_SCHEDULER.CREATE_JOB_CLASS (
    job_class_name IN VARCHAR2,
    resource_consumer_group IN VARCHAR2 DEFAULT NULL,
    service IN VARCHAR2 DEFAULT NULL,
    logging_level IN PLS_INTEGER DEFAULT DBMS_SCHEDULER.LOGGING_RUNS,
    log_history IN PLS_INTEGER DEFAULT NULL,
    comments IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 143-29 CREATE_JOB_CLASS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_class_name</td>
<td>The name to assign to the job class. Job classes can only be created in the SYS schema. This attribute specifies the name of the job class and uniquely identifies the job class. The name must be unique in the SQL namespace. For example, a job class cannot have the same name as a table in a schema.</td>
</tr>
<tr>
<td>resource_consumer_group</td>
<td>This attribute specifies the resource consumer group that his class is associated with. A resource consumer group is a set of synchronous or asynchronous sessions that are grouped together based on their processing needs. A job class has a many-to-one relationship with a resource consumer group. The resource consumer group that the job class associates with determines the resources that are allocated to the job class. If a resource consumer group is dropped, job classes associated with it are then associated with the default resource consumer group. If no resource consumer group is specified, job classes are associated with the default resource consumer group. If the specified resource consumer group does not exist when creating the job class, an error occurs.</td>
</tr>
<tr>
<td>service</td>
<td>This attribute specifies the database service that the jobs in this class have affinity to. In an Oracle RAC environment, this means that the jobs in this class only run on those database instances that are assigned to the specific service. Note that a service can be mapped to a resource consumer group, so you can also control resources allocated to jobs by specifying a service. See DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING for details. If both the resource_consumer_group and service attributes are specified, and if the service is mapped to a resource consumer group, the resource_consumer_group attribute takes precedence. If no service is specified, the job class belongs to the default service, which means it has no service affinity and any one of the database instances within the cluster might run the job. If the service that a job class belongs to is dropped, the job class will then belong to the default service. If the specified service does not exist when creating the job class, then an error occurs.</td>
</tr>
</tbody>
</table>
| logging_level      | This attribute specifies how much information is logged. The possible options are:  
  - DBMS_SCHEDULER.LOGGING_OFF  
    No logging is performed for any jobs in this class.  
  - DBMS_SCHEDULER.LOGGING_RUNS  
    The Scheduler writes detailed information to the job log for all runs of each job in this class. This is the default.  
  - DBMS_SCHEDULER.LOGGING_FAILED_RUNS  
    The Scheduler logs only jobs that failed in this class.  
  - DBMS_SCHEDULER.LOGGING_FULL  
    In addition to recording every run of a job, the Scheduler records all operations performed on all jobs in this class. Every time a job is created, enabled, disabled, altered (with SET_ATTRIBUTE), stopped, and so, an entry is recorded in the log. |
Table 143-29  (Cont.) CREATE_JOB_CLASS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_history</td>
<td>This attribute controls the number of days that job log entries for jobs in this class are retained. It helps prevent the job log from growing indiscriminately. The range of valid values is 0 through 1000000. If set to 0, no history is kept. If NULL (the default), retention days are set by the log_history Scheduler attribute (set with SET_SCHEDULER_ATTRIBUTE).</td>
</tr>
<tr>
<td>comments</td>
<td>This attribute is for an optional comment about the job class. By default, this attribute is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

For users to create jobs that belong to a job class, the job owner must have execute privileges on the job class. Therefore, after the job class has been created, execute privileges must be granted on the job class so that users create jobs belonging to that class. You can also grant the execute privilege to a role.

Creating a job class requires the manage scheduler system privilege.

143.6.18 CREATE_JOBS Procedure

This procedure creates multiple jobs and sets the values of their arguments in a single call.

Syntax

```sql
DBMS_SCHEDULER.CREATE_JOBS (
    jobdef_array IN SYS.JOB_DEFINITION_ARRAY,
    commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 143-30  CREATE_JOBS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobdef_array</td>
<td>The array of job definitions. See &quot;Data Structures&quot; for a description of the JOB_DEFINITION_ARRAY and JOB_DEFINITION datatypes.</td>
</tr>
</tbody>
</table>
| commit_semantics | The commit semantics. The following types are supported:  
  • STOP_ON_FIRST_ERROR returns on the first error. Previous successfully created jobs are committed to disk. This is the default.  
  • TRANSACTIONAL returns on the first error and everything that happened before that error is rolled back.  
  • ABSORB_ERRORS tries to absorb any errors and attempts to create the rest of the jobs on the list. It commits all successfully created jobs. If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for details. |

Usage Notes

This procedure creates many jobs in the context of a single transaction. To realize the desired performance gains, the jobs being created must be grouped in batches of sufficient size.
Calling `CREATE_JOBS` with a small array size may not be much faster than calling `CREATE_JOB` once for each job.

You cannot use this procedure to create multiple-destination jobs. That is, the `destination` attribute of the `job_definition` object cannot reference a destination group.

**Examples**

See *Oracle Database Administrator's Guide*.

### 143.6.19 CREATE_PROGRAM Procedure

This procedure creates a program.

**Syntax**

```sql
DBMS_SCHEDULER.CREATE_PROGRAM (  
    program_name             IN VARCHAR2,  
    program_type             IN VARCHAR2,  
    program_action           IN VARCHAR2,  
    number_of_arguments      IN PLS_INTEGER DEFAULT 0,  
    enabled                  IN BOOLEAN DEFAULT FALSE,  
    comments                 IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_name</td>
<td>The name to assign to the program. The name must be unique in the SQL namespace. For example, a program cannot have the same name as a table in a schema. If no name is specified, then an error occurs.</td>
</tr>
</tbody>
</table>
### Table 143-31  (Cont.) CREATE_PROGRAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_type</td>
<td>This attribute specifies the type of program you are creating. If it is not specified then you get an error. These are the supported values for <code>program_type</code>:</td>
</tr>
<tr>
<td></td>
<td>• 'PLSQL_BLOCK'</td>
</tr>
<tr>
<td></td>
<td>This specifies that the program is a PL/SQL block. Job or program arguments are not supported when the job or program type is <code>PLSQL_BLOCK</code>. In this case, the number of arguments must be 0.</td>
</tr>
<tr>
<td></td>
<td>• 'STORED_PROCEDURE'</td>
</tr>
<tr>
<td></td>
<td>This specifies that the program is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported. PL/SQL procedures with <code>INOUT</code> or <code>OUT</code> arguments are not supported.</td>
</tr>
<tr>
<td></td>
<td>• 'EXECUTABLE'</td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is going to be run outside the database using an external executable. External programs imply anything that can be executed from the operating system command line. <code>AnyData</code> arguments are not supported with job or program type <code>EXECUTABLE</code>.</td>
</tr>
<tr>
<td></td>
<td>• 'EXTERNAL_SCRIPT'</td>
</tr>
<tr>
<td></td>
<td><code>project 25230, this bullet and next two</code></td>
</tr>
<tr>
<td></td>
<td>This specifies that the job is an external script that uses the command shell of the computer running the job. For Windows this is <code>cmd.exe</code> and for UNIX based systems the <code>sh</code> shell, unless a different interpreter is specified by prefixing the first line of the script with <code>#!</code>.</td>
</tr>
<tr>
<td></td>
<td>• 'SQL_SCRIPT'</td>
</tr>
<tr>
<td></td>
<td>This specifies that the program is a SQL<em>Plus script. A job using this program must point to a credential that contains a valid operating system username and password. The SQL</em>Plus script is run by SQL<em>Plus executable. The job using this program may point to a connect credential that contains a database credential. If so, this credential is used to connect to the database before running the SQL</em>Plus script.</td>
</tr>
<tr>
<td></td>
<td>Note that if you choose to use connect credential, you must use <code>set_attribute</code> to specify the <code>Connect_Credential_Name</code> attribute. If you do not have connect credential, you must include an explicit SQL*Plus connect statement providing a valid database userid / password.</td>
</tr>
<tr>
<td></td>
<td>• 'BACKUP_SCRIPT'</td>
</tr>
<tr>
<td></td>
<td>This specifies that the program is an RMAN backup script. The script runs a connect statement that uses either a password or OS authentication before it executes any target commands. The Scheduler uses the RMAN executable from the current Oracle home to run the script and throws an error if this is missing.</td>
</tr>
</tbody>
</table>
Table 143-31  (Cont.) CREATE_PROGRAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| program_action | This attribute specifies the action of the program. If program_action is not specified, an error is generated.  

The following actions are possible:

- For a PL/SQL block, the action is to execute PL/SQL code. These blocks must end with a semicolon.  
For example, my_proc(); or BEGIN my_proc(); END; or DECLARE arg pls_integer:= 10; BEGIN my_proc2(arg); END;.

Note that the Scheduler wraps job_action in its own block and passes the following to PL/SQL for execution: DECLARE ... BEGIN job_action END; This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except event_message in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value. See Table 143-40 for details on available metadata attributes.

- If it is an anonymous block, special Scheduler metadata may be accessed using the following variable names: job_name, job_owner, job_start, window_start, window_end. For more information, see the "DEFINE_METADATA_ARGUMENT Procedure".

- For a stored procedure, the action is the name of the stored procedure. You have to specify the schema if the procedure resides in a schema other than the job.  
If case sensitivity is needed, enclose the schema name and the store procedure name in double quotes. For example, program_action=>'"Schema"."Procedure"'.

- For an executable, the action is the name of the external executable, including the full path name, but excluding any command-line arguments. If the action starts with a single question mark ('?'), the question mark is replaced by the path to the Oracle home directory for a local job or to the Scheduler agent home for a remote job. If the action contains an at sign ('@') and the job is local, the at sign is replaced with the SID of the current Oracle instance.

- For an external script, the action must be either the path to an operating system script or an inline operating system script. If the program_action is a path to a script, then the script must reside on every computer that the program runs on. The program_action may contain calls to SQL*Plus or RMAN executables directly, without having to specify its full path, given that they are stored on their default location for every computer that runs the job.

The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called. The program points
Table 143-31  (Cont.) CREATE_PROGRAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>to a credential that contains a valid operating system username and password.</td>
</tr>
<tr>
<td></td>
<td>• For a SQL script, the action must be either the path to a SQL<em>Plus script or an inline SQL</em>Plus script. If the</td>
</tr>
<tr>
<td></td>
<td>program_action is a path to a script, then the script must reside on every computer that the program runs on. The job can only have</td>
</tr>
<tr>
<td></td>
<td>arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called. If the</td>
</tr>
<tr>
<td></td>
<td>arguments are named, they are also bound to named variables in the SQL*Plus session.</td>
</tr>
<tr>
<td></td>
<td>• For a backup script, the action must be either the path to a RMAN script or an inline RMAN script. If the</td>
</tr>
<tr>
<td></td>
<td>program_action is a path to a script, then the script must reside on every computer that the program runs on. The job can only have</td>
</tr>
<tr>
<td></td>
<td>arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called.</td>
</tr>
<tr>
<td>number_of_arguments</td>
<td>This attribute specifies the number of arguments the program takes. If this parameter is not specified, then the default is 0. A program</td>
</tr>
<tr>
<td></td>
<td>can have a maximum of 255 arguments. If the program_type is PLSQL_BLOCK, then this parameter is ignored.</td>
</tr>
<tr>
<td>enabled</td>
<td>This flag specifies whether the program should be created as enabled or not. If the flag is set to TRUE, then validity checks are made and</td>
</tr>
<tr>
<td></td>
<td>the program is created as ENABLED if all the checks be successful. By default, this flag is set to FALSE, meaning not created enabled. You</td>
</tr>
<tr>
<td></td>
<td>can also call the ENABLE procedure to enable the program before it can be used.</td>
</tr>
<tr>
<td>comments</td>
<td>A comment about the program. By default, this attribute is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

To create a program in their own schema, users need the CREATE JOB privilege. A user with the CREATE ANY JOB privilege can create a program in any schema. A program is created in a disabled state by default (unless the enabled parameter is set to TRUE). It cannot be executed by a job until it is enabled.

To use your programs, other users must have EXECUTE privileges, therefore once a program has been created, you have to grant EXECUTE privileges on it.

See Also:

"DEFINE_PROGRAM_ARGUMENT Procedure"
143.6.20 CREATE_RESOURCE Procedure

This procedure allows users to specify the resources used by jobs or to create a new resource.

Syntax

DBMS_SCHEDULER.CREATE_RESOURCE ( resource_name    IN VARCHAR2, units            IN PLS_INTEGER, status           IN VARCHAR2 DEFAULT 'ENFORCE_CONSTRAINTS', constraint_level IN VARCHAR2 DEFAULT 'JOB_LEVEL', comments         IN VARCHAR2 DEFAULT NULL);

Parameters

Table 143-32  CREATE_RESOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource_name</td>
<td>The name of the resource.</td>
</tr>
<tr>
<td>units</td>
<td>The number of units of this resource that the job or program uses.</td>
</tr>
<tr>
<td>status</td>
<td>The status of the resource.</td>
</tr>
<tr>
<td></td>
<td>• ‘ENFORCE_CONSTRAINTS’. This is the default value, and when set, will force the scheduler to enforce resource limits. When the maximum number of units of this resource has been reached no additional jobs using this resource will get started.</td>
</tr>
<tr>
<td></td>
<td>• ‘IGNORE_CONSTRAINTS’. When set, the scheduler will ignore any constraints on this resource.</td>
</tr>
<tr>
<td></td>
<td>• ‘BLOCKED_ALL_JOBS’. No jobs having a constraint on this resource will be allowed to run. The resource is considered to be permanently blocking until switched to one of the other two states.</td>
</tr>
<tr>
<td>constraint_level</td>
<td>Level of the constraint: JOB_LEVEL or PROGRAM_LEVEL. For incompatibilities, for JOB_LEVEL, the incompatibility members must be jobs; for PROGRAM_LEVEL the incompatibility members must be programs.</td>
</tr>
<tr>
<td>comments</td>
<td>Descriptive comment about the resource.</td>
</tr>
</tbody>
</table>

Usage Notes

The following example creates a new resource.

BEGIN
    DBMS_SCHEDULER.CREATE_RESOURCE ( resource_name => 'my_resource', units => 3, state => 'ENFORCE_CONSTRAINTS', comments => 'Resource1'
    )
END;
/
143.6.21 CREATE_SCHEDULE Procedure

This procedure creates a schedule.

Syntax

```sql
DBMS_SCHEDULER.CREATE_SCHEDULE ( 
    schedule_name          IN VARCHAR2,
    start_date             IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
    repeat_interval        IN VARCHAR2,
    end_date               IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
    comments               IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 143-33 CREATE_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_name</td>
<td>The name to assign to the schedule. The name must be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.</td>
</tr>
<tr>
<td>start_date</td>
<td>This attribute specifies the first date and time on which this schedule becomes valid. For a repeating schedule, the value for start_date is a reference date. In this case, the start of the schedule is not the start_date; it depends on the repeat interval specified. start_date is used to determine the first instance of the schedule. If start_date is specified in the past and no value for repeat_interval is specified, the schedule is invalid. For a repeating job or window, start_date can be derived from the repeat_interval if it is not specified. If start_date is null, then the date that the job or window is enabled is used. start_date and repeat_interval cannot both be null.</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>This attribute specifies how often the schedule repeats. It is expressed using calendaring syntax. See &quot;Calendaring Syntax&quot; for further information. PL/SQL expressions are not allowed as repeat intervals for named schedules.</td>
</tr>
<tr>
<td>end_date</td>
<td>The date and time after which jobs will not run and windows will not open. A non-repeating schedule that has no end_date is valid forever. end_date has to be after the start_date. If this is not the case, then an error is generated when the schedule is created.</td>
</tr>
<tr>
<td>comments</td>
<td>This attribute specifies an optional comment about the schedule. By default, this attribute is NULL.</td>
</tr>
</tbody>
</table>
Usage Notes

This procedure requires the CREATE JOB privilege to create a schedule in your own
schema or the CREATE ANY JOB privilege to create a schedule in someone else’s
schema by specifying schema.schedule_name. Once a schedule has been created, it
can be used by other users. The schedule is created with access to PUBLIC. Therefore,
there is no need to explicitly grant access to the schedule.

143.6.22 CREATE_WINDOW Procedure

This procedure creates a recurring time window and associates it with a resource plan.
You can then use the window to schedule jobs that run under the associated resource
plan. Windows are created in the SYS schema.

The procedure is overloaded.

Syntax

Creates a window using a named schedule object:

```sql
DBMS_SCHEDULER.CREATE_WINDOW (  
    window_name       IN VARCHAR2,  
    resource_plan     IN VARCHAR2,  
    schedule_name     IN VARCHAR2,  
    duration          IN INTERVAL DAY TO SECOND,  
    window_priority   IN VARCHAR2 DEFAULT 'LOW',  
    comments          IN VARCHAR2 DEFAULT NULL);
```

Creates a window using an inlined schedule:

```sql
DBMS_SCHEDULER.CREATE_WINDOW (  
    window_name       IN VARCHAR2,  
    resource_plan     IN VARCHAR2,  
    start_date        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,  
    repeat_interval   IN VARCHAR2,  
    end_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,  
    duration          IN INTERVAL DAY TO SECOND,  
    window_priority   IN VARCHAR2 DEFAULT 'LOW',  
    comments          IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 143-34  CREATE_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_name</td>
<td>The name to assign to the window. The name must be unique in the SQL namespace. All windows are in the SYS schema, so the preface 'SYS' is optional.</td>
</tr>
</tbody>
</table>
Table 143-34  (Cont.) CREATE_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource_plan</td>
<td>This attribute specifies the resource plan that automatically activates when the window opens. When the window closes, the system switches to the appropriate resource plan, which is usually the plan that was in effect before the window opened, but can also be the plan of a different window. Only one resource plan can be associated with a window. It may be NULL or the empty string (&quot;&quot;), when it is NULL, the resource plan in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window. If the window is open and the resource plan is dropped, then the resource allocation for the duration of the window is not affected.</td>
</tr>
<tr>
<td>start_date</td>
<td>This attribute specifies the first date and time on which this window is scheduled to open. If the value for start_date specified is in the past or is not specified, the window opens as soon as it is created. For repeating windows that use a calendaring expression to specify the repeat interval, the value for start_date is a reference date. The first time the window opens depends on the repeat interval specified and the value for start_date.</td>
</tr>
<tr>
<td>duration</td>
<td>This attribute specifies how long the window stays open. For example, 'interval '5' hour' for five hours. There is no default value for this attribute. Therefore, if no value is specified when the window is created, an error occurs. The duration is of type interval day to seconds and ranges from one minute to 99 days.</td>
</tr>
<tr>
<td>schedule_name</td>
<td>This attribute specifies the name of the schedule associated with the window.</td>
</tr>
</tbody>
</table>
| repeat_interval | This attribute specifies how often the window repeats. It is expressed using the Scheduler calendaring syntax. See "Calendaring Syntax" for more information.  

A PL/SQL expression cannot be used to specify the repeat interval for a window. The expression specified is evaluated to determine the next time the window opens. If no repeat_interval is specified, the window opens only once at the specified start date. |
| end_date      | This attribute specifies the date and time after which the window no longer opens. When the value for end_date is reached, the window is disabled. In the *_SCHEDULER_WINDOWS views, the enabled flag of the window is set to FALSE.  

A non-repeating window that has no value for end_date opens only once for the duration of the window. For a repeating window, if no end_date is specified, then the window keeps repeating forever. The end_date must be after the start_date. If it is not, then an error is generated when the window is created. |
Table 143-34  (Cont.) CREATE_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_priority</td>
<td>This attribute is only relevant when two windows overlap. Because only one window can be in effect at one time, the window priority determines which window opens. The two possible values for this attribute are 'HIGH' and 'LOW'. A high priority window has precedence over a low priority window, therefore, the low priority window does not open if it overlaps a high priority window. By default, windows are created with priority 'LOW'.</td>
</tr>
<tr>
<td>comments</td>
<td>This attribute specifies an optional comment about the window. By default, this attribute is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

Creating a window requires the MANAGE SCHEDULER privilege.

Scheduler windows are the principal mechanism used to automatically switch resource plans according to a schedule. You can also manually activate a resource plan by using the ALTER SYSTEM SET RESOURCE_MANAGER_PLAN statement or the DBMS_RESOURCE_MANAGER.SWITCH_PLAN package procedure. Note that either of these manual methods can also disable resource plan switching by Scheduler windows. For more information, see Oracle Database Administrator's Guide and “SWITCH_PLAN Procedure”.

143.6.23 DEFINE ANYDATA_ARGUMENT Procedure

This procedure defines a name or default value for a program argument that is of a complex type and must be encapsulated within an ANYDATA object. A job that references the program can override the default value.

Syntax

DBMS_SCHEDULER.DEFINE_ANYDATA_ARGUMENT (program_name            IN VARCHAR2,
argument_position       IN PLS_INTEGER,
argument_name           IN VARCHAR2 DEFAULT NULL,
argument_type           IN VARCHAR2,
default_value           IN SYS.ANYDATA,
out_argument            IN BOOLEAN DEFAULT FALSE);

Parameters

Table 143-35  DEFINE ANYDATA_ARGUMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_name</td>
<td>The name of the program to be altered. A program with this name must exist.</td>
</tr>
<tr>
<td>argument_position</td>
<td>The position of the argument as it is passed to the executable. Argument numbers go from one to the number_of_arguments specified for the program. This must be unique, so it can replace any argument already defined at this position.</td>
</tr>
</tbody>
</table>
Table 143-35  (Cont.) DEFINE_ANYDATA_ARGUMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument_name</td>
<td>The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures, including the <code>SET_JOB_ANYDATA_VALUE</code> Procedure.</td>
</tr>
<tr>
<td>argument_type</td>
<td>The datatype of the argument being defined. This is not verified or used by the Scheduler. It is only used by the user of the program when deciding what value to assign to the argument.</td>
</tr>
<tr>
<td>default_value</td>
<td>The default value to be assigned to the argument encapsulated within an AnyData object. This is optional.</td>
</tr>
<tr>
<td>out_argument</td>
<td>This parameter is reserved for future use. It must be set to <code>FALSE</code>.</td>
</tr>
</tbody>
</table>

Usage Notes

All program arguments from one to the `number_of_arguments` value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have `ALTER` privileges on that program. You can also define a program argument if you have the `CREATE ANY JOB` privilege.

See Also:

- "DEFINE_PROGRAM_ARGUMENT Procedure"
- "SET_JOB_ANYDATA_VALUE Procedure"

143.6.24 DEFINE_CHAIN_EVENT_STEP Procedure

This procedure adds or replaces a chain step and associates it with an event schedule or an inline event.

Once started in a running chain, this step does not complete until the specified event has occurred. Every step in a chain must be defined before the chain can be enabled and used. Defining a step gives it a name and specifies what happens during the step. If a step already exists with this name, the new step replaces the old one.

Syntax

```sql
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (  
    chain_name              IN VARCHAR2,  
    step_name               IN VARCHAR2,  
    event_schedule_name     IN VARCHAR2,  
    timeout                 IN INTERVAL DAY TO SECOND DEFAULT NULL);  
```

```sql
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (  
    chain_name              IN VARCHAR2,  
    step_name               IN VARCHAR2,  
    event_schedule_name     IN VARCHAR2,  
    timeout                 IN INTERVAL DAY TO SECOND DEFAULT NULL);  
```
Parameters

Table 143-36  DEFINE_CHAIN_EVENT_STEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain that the step is in</td>
</tr>
<tr>
<td>step_name</td>
<td>The name of the step</td>
</tr>
<tr>
<td>event_schedule_name</td>
<td>The name of the event schedule that the step waits for</td>
</tr>
<tr>
<td>timeout</td>
<td>This parameter is reserved for future use</td>
</tr>
<tr>
<td>event_condition</td>
<td>See the CREATE_EVENT_SCHEDULE Procedure</td>
</tr>
<tr>
<td>queue_spec</td>
<td>See the CREATE_EVENT_SCHEDULE Procedure</td>
</tr>
</tbody>
</table>

Usage Notes

Defining a chain step requires ALTER privileges on the chain either as the owner of the chain, or as a user with the ALTER object privilege on the chain or the CREATE any job system privilege.

You can base a chain step on a file watcher as well. To do this, provide the file watcher name directly in the queue_spec parameter, or use a file watcher schedule for the event_schedule_name parameter.

See Also:

"DEFINE_CHAIN_STEP Procedure"

143.6.25 DEFINE_CHAIN_RULE Procedure

This procedure adds a new rule to an existing chain, specified as a condition-action pair. The condition is expressed using either SQL or the Scheduler chain condition syntax and indicates the prerequisites for the action to occur. The action is a result of the condition being met.

An actual rule object is created to store the rule in the schema where the chain resides. If a rule name is given, this name is used for the rule object. If an existing rule name in the schema of the chain is given, the existing rule is altered. (A schema different than the schema of the chain cannot be specified). If no rule name is given, one is generated in the form SCHED_RULE$N.

Syntax

DBMS_SCHEDULER.DEFINE_CHAIN_RULE (  
  chain_name IN VARCHAR2,  
  condition IN VARCHAR2,  
  action IN VARCHAR2,
Parameters

Table 143-37  DEFINE_CHAIN_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain to alter.</td>
</tr>
</tbody>
</table>
| condition   | A boolean expression which must evaluate to `TRUE` for the action to be performed. Every chain must have a rule that evaluates to `TRUE` to start the chain. For this purpose, you can use a rule that has `TRUE` as its condition if you are using Scheduler chain condition syntax, or `1=1` as its condition if you are using SQL syntax.  
  • Scheduler Chain Condition Syntax
    See "Scheduler Chain Condition Syntax" and Oracle Database Administrator's Guide for details
  • SQL WHERE Clause Syntax
    Conditions expressed with SQL must use the syntax of a `SELECT` statement `WHERE` clause.
    You can refer to chain step attributes by using the chain step name as a bind variable.
    The bind variable syntax is `:step_name.attribute.(step_name refers to a typed object.)` Possible attributes are: `completed, state, start_date, end_date, error_code, and duration.`
    Possible values for the `state` attribute include: 'NOT_STARTED', 'SCHEDULED', 'RUNNING', 'PAUSED', 'STALLLED', 'SUCCEEDED', 'FAILED', and 'STOPPED'. If a step is in the state 'SUCCEEDED', 'FAILED', or 'STOPPED', its completed attribute is set to 'TRUE', otherwise completed is 'FALSE'.  
| action      | The action to be performed when the rule evaluates to `TRUE`. The action must consist of at least one keyword with an optional value and an optional delay clause.  
  Possible actions include:
  • `[AFTER delay_interval] START step_1[,step_2 ...]`
  • `STOP step_1[,step_2 ...]`
  • `END [{end_value}step_name.error_code]`
  At the beginning of the START action, a delay clause can specify a delay interval before performing the action. `delay_interval` is a formatted datetime interval of the form `HH:MM:SS`.  
  The END action ends the chain with an error code equal to either the supplied `end_value` or the error code that `step_name` completes with. The default error code is 0, indicating a successful chain run.  
| rule_name   | The name of the rule being created. If no `rule_name` is given, one is generated in the form `SCHED_RULES_{N}`.                                   |
| comments    | An optional comment describing the rule. This is stored in the rule object created.                                                           |

Scheduler Chain Condition Syntax

The Scheduler chain condition syntax provides an easy way to construct a condition using the states and error codes of steps in the current chain.
Chain Condition Syntax

The following are the available constructs for Scheduler chain condition syntax, which are all boolean expressions:

```
TRUE
FALSE
stepname [NOT] SUCCEEDED
stepname [NOT] FAILED
stepname [NOT] STOPPED
stepname [NOT] COMPLETED
stepname ERROR_CODE IN (integer, integer, integer ...)
stepname ERROR_CODE NOT IN (integer, integer, integer ...)
stepname ERROR_CODE = integer
stepname ERROR_CODE != integer
stepname ERROR_CODE <> integer
stepname ERROR_CODE > integer
stepname ERROR_CODE >= integer
stepname ERROR_CODE < integer
stepname ERROR_CODE <= integer
```

These boolean operators are available to create more complex conditions:

```
expression AND expression
expression OR expression
NOT (expression)
```

integer can be positive or negative. Parentheses may be used for clarity or to enforce ordering. You must use parentheses with the NOT operator.

PL/SQL code that runs as part of a step can set the value of ERROR_CODE for that step with the RAISE_APPLICATION_ERROR statement.

Usage Notes

Defining a chain rule requires ALTER privileges on the chain (either as the owner, or as a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

You must define at least one rule that starts the chain and at least one that ends it. See the section "Adding Rules to a Chain" in Oracle Database Administrator's Guide for more information.

Examples

The following are examples of using rule conditions and rule actions.

Rule Conditions Using Scheduler Chain Condition Syntax

```
'step1 completed'
-- satisfied when step step1 has completed. (step1 completed is also TRUE when any
-- of the following are TRUE: step1 succeeded, step1 failed, step1 stopped.)

'step1 succeeded and step2 succeeded'
-- satisfied when steps step1 and step2 have both succeeded

'step1 error_code > 100'
-- satisfied when step step1 has failed with an error_code greater than 100

'step1 error_code IN (1, 3, 5, 7)'
-- satisfied when step step1 has failed with an error_code of 1, 3, 5, or 7
```
Rule Conditions Using SQL Syntax

':step1.completed = 'TRUE'' AND :step1.end_date >SYSDATE-1/24'
--satisfied when step step1 completed less than an hour ago

':step1.duration > interval ''5'' minute'
--satisfied when step step1 has completed and took longer than 5 minutes to complete

Rule Actions

'AFTER 01:00:00 START step1, step2'
--After an hour start steps step1 and step2

'STOP step1'
--Stop step step1

END step4.error_code'
--End the chain with the error code that step step4 finished with. If step4 has not completed, the chain will be ended unsuccessfullly with error code 27435.

'END' or 'END 0'
--End the chain successfully (with error_code 0)

'END 100'
--End the chain unsuccessfullly with error code 100.

143.6.26 DEFINE_CHAIN_STEP Procedure

This procedure adds or replaces a chain step and associates it with a program or a nested chain. When the chain step is started, the specified program or chain is run. If a step already exists with the name supplied in the chain_name argument, the new step replaces the old one.

The chain owner must have EXECUTE privileges on the program or chain associated with the step. Only one program or chain can run during a step.

You cannot set all possible step attributes with this procedure. Use the ALTER_CHAIN procedure to set additional chain step attributes, such as credential_name and destination_name.

Syntax

DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
chain_name              IN VARCHAR2,
step_name               IN VARCHAR2,
program_name            IN VARCHAR2);

Parameters

Table 143-38  DEFINE_CHAIN_STEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain to alter.</td>
</tr>
<tr>
<td>step_name</td>
<td>The name of the step being defined. If a step already exists with this name, the new step replaces the old one.</td>
</tr>
</tbody>
</table>
### Table 143-38  (Cont.) DEFINE_CHAIN_STEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_name</td>
<td>The name of a program or chain to run during this step. The chain owner must have <code>EXECUTE</code> privileges on this program or chain.</td>
</tr>
</tbody>
</table>

#### Usage Notes

Defining a chain step requires `ALTER` privileges on the chain (either as the owner, or a user with `ALTER` privileges on the chain or the `CREATE ANY JOB` system privilege).

#### See Also:
- "ALTER_CHAIN Procedure"
- "DEFINE_CHAIN_EVENT_STEP Procedure"

## 143.6.27 DEFINE_METADATA_ARGUMENT Procedure

This procedure defines a special metadata argument for the program. The Scheduler can pass Scheduler metadata through this argument to your stored procedure or other executable. You cannot set values for jobs using this argument.

### Syntax

```sql
DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT (
    program_name            IN VARCHAR2,
    metadata_attribute      IN VARCHAR2,
    argument_position       IN PLS_INTEGER,
    argument_name           IN VARCHAR2 DEFAULT NULL);
```

### Parameters

#### Table 143-39  DEFINE_METADATA_ARGUMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_name</td>
<td>The name of the program to be altered</td>
</tr>
<tr>
<td>metadata_attribute</td>
<td>The metadata to be passed. Valid metadata attributes are: 'job_name', 'job_subname', 'job_owner', 'job_start', 'window_start', 'window_end', and 'event_message'. Table 143-40 describes these attributes in detail.</td>
</tr>
<tr>
<td>argument_position</td>
<td>The position of the argument as it is passed to the executable. The position cannot be greater than the number of arguments specified for the program. It must be unique, so it replaces any argument already defined at this position.</td>
</tr>
<tr>
<td>argument_name</td>
<td>The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures.</td>
</tr>
</tbody>
</table>
### Table 143-40  Metadata Attributes

<table>
<thead>
<tr>
<th>Metadata Attribute</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>VARCHAR2</td>
<td>Name of the currently running job</td>
</tr>
<tr>
<td>job_subname</td>
<td>VARCHAR2</td>
<td>Subname of the currently running job. The name + subname form a unique identifier for a job that is running a chain step. NULL if the job is not part of a chain.</td>
</tr>
<tr>
<td>job_owner</td>
<td>VARCHAR2</td>
<td>Owner of the currently running job</td>
</tr>
<tr>
<td>job_scheduled_start</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>When the currently running job was scheduled to start</td>
</tr>
<tr>
<td>job_start</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>When the currently running job started</td>
</tr>
<tr>
<td>window_start</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>If the job was started by a window, the time that the window opened</td>
</tr>
<tr>
<td>window_end</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>If the job was started by a window, the time that the window is scheduled to close</td>
</tr>
<tr>
<td>event_message</td>
<td>(See Description)</td>
<td>For an event-based job, the message content of the event that started the job. The datatype of this attribute depends on the queue used for the event. It has the same type as the USER_DATA column of the queue table. In the case of a file arrival event, event_message is of type SYS.SCHEDULER_FILEWATCHER_RESULT. See &quot;SCHEDULER_FILEWATCHER_RESULT Object Type&quot;.</td>
</tr>
</tbody>
</table>

### Usage Notes

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

All metadata attributes except event_message can be used in PL/SQL blocks that you enter into the job_action or program_action attributes of jobs or programs, respectively. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value.

### 143.6.28 DEFINE_PROGRAM_ARGUMENT Procedure

This procedure defines a name or default value for a program argument. If no default value is defined for a program argument, the job that references the program must supply an argument value. (The job can also override a default value.)

This procedure is overloaded.

#### Syntax

Defines a program argument without a default value:

```sql
PROCEDURE define_program_argument(
  program_name            IN VARCHAR2,
  argument_position       IN PLS_INTEGER,
  argument_name           IN VARCHAR2 DEFAULT NULL,
  argument_type           IN VARCHAR2,
  out_argument            IN BOOLEAN DEFAULT FALSE);
```
Defines a program argument with a default value:

```sql
PROCEDURE define_program_argument(
    program_name            IN VARCHAR2,
    argument_position       IN PLS_INTEGER,
    argument_name           IN VARCHAR2 DEFAULT NULL,
    argument_type           IN VARCHAR2,
    default_value           IN VARCHAR2,
    out_argument            IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 143-41  DEFINE_PROGRAM_ARGUMENT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_name</td>
<td>The name of the program to be altered. A program with this name must exist.</td>
</tr>
<tr>
<td>argument_position</td>
<td>The position of the argument as it is passed to the executable. Argument numbers go from one to the number_of_arguments specified for the program. This must be unique so it replaces any argument already defined at this position.</td>
</tr>
<tr>
<td>argument_name</td>
<td>The name to assign to the argument. It is optional, but must be unique for the program if specified. If you assign a name, the name can then be used by other package procedures, including the SET_JOB_ARGUMENT_VALUE Procedure.</td>
</tr>
<tr>
<td>argument_type</td>
<td>The datatype of the argument being defined. This is not verified or used by the Scheduler. The program user uses argument_type when deciding what value to assign to the argument. Any valid SQL datatype is allowed.</td>
</tr>
<tr>
<td>default_value</td>
<td>The default value to be assigned to the argument if none is specified by the job.</td>
</tr>
<tr>
<td>out_argument</td>
<td>This parameter is reserved for future use. It must be set to FALSE.</td>
</tr>
</tbody>
</table>

### Usage Notes

All program arguments from 1 to the number_of_arguments value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

DEFINE_PROGRAM_ARGUMENT only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

**See Also:**

- "DEFINE_ANYDATA_ARGUMENT Procedure"
- "SET_JOB_ARGUMENT_VALUE Procedure"
143.6.29 DISABLE Procedure

This procedure disables a program, job, chain, window, database destination, external destination, file watcher, or group. When an object is disabled, its enabled attribute is set to FALSE.

Syntax

```
DBMS_SCHEDULER.DISABLE (
    name              IN VARCHAR2,
    force             IN BOOLEAN DEFAULT FALSE,
    commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

**Table 143-42 DISABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the object being disabled. Can be a comma-delimited list.</td>
</tr>
<tr>
<td></td>
<td>If a job class name is specified, then all the jobs in the job class are</td>
</tr>
<tr>
<td></td>
<td>disabled. The job class is not disabled.</td>
</tr>
<tr>
<td></td>
<td>If a group name is specified, then the group is disabled, but the enabled</td>
</tr>
<tr>
<td></td>
<td>state of the group members is unaffected.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, objects are disabled even if other objects depend on them. See the</td>
</tr>
<tr>
<td></td>
<td>usage notes for more information.</td>
</tr>
<tr>
<td>commit_semantics</td>
<td>The commit semantics. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>• STOP_ON_FIRST_ERROR: The procedure returns on the first error and the</td>
</tr>
<tr>
<td></td>
<td>previous disable operations that were successful are committed to disk.</td>
</tr>
<tr>
<td></td>
<td>This is the default.</td>
</tr>
<tr>
<td></td>
<td>• TRANSACTIONAL: The procedure returns on the first error and everything</td>
</tr>
<tr>
<td></td>
<td>that happened before that error is rolled back.</td>
</tr>
<tr>
<td></td>
<td>This type is only supported when disabling a job or a list of jobs. In</td>
</tr>
<tr>
<td></td>
<td>addition, this type is not supported when force is set to TRUE.</td>
</tr>
<tr>
<td></td>
<td>• ABSORB_ERRORS: The procedure tries to absorb any errors and disable</td>
</tr>
<tr>
<td></td>
<td>the rest of the jobs and commits all the disable operations that were</td>
</tr>
<tr>
<td></td>
<td>successful. If errors occur, you can query the view</td>
</tr>
<tr>
<td></td>
<td>SCHEDULER_BATCH_ERRORS for details.</td>
</tr>
<tr>
<td></td>
<td>This type is only supported when disabling a job or a list of jobs.</td>
</tr>
</tbody>
</table>

Usage Notes

Windows must be preceded by SYS.

Disabling an object that is already disabled does not generate an error.

The purpose of the force option is to point out dependencies. No dependent objects are altered.

To run DISABLE for a window or a group of type WINDOW, you must have the MANAGE_SCHEDULER privilege.

You can use DISABLE with any schema except the SYS schema.
Jobs

Disabling a job means that, although the metadata of the job is there, it should not run and the job coordinator will not pick up these jobs for processing. When a job is disabled, its state in the job queue is changed to disabled.

If force is set to FALSE and the job is currently running, an error is returned.

If force is set to TRUE, the job is disabled, but the currently running instance is allowed to finish.

For jobs with multiple destinations, you cannot disable a child job at a specific destination. Instead, you can disable the destination.

Programs

When a program is disabled, the status is changed to disabled. A disabled program implies that, although the metadata is still there, jobs that point to this program cannot run.

If force is set to FALSE, the program must not be referenced by any job, otherwise an error will occur.

If force is set to TRUE, those jobs that point to the program will not be disabled, however, they will fail at runtime because their program will not be valid.

Running jobs that point to the program are not affected by the DISABLE call and are allowed to continue

No arguments that pertain to the program are affected when the program is disabled.

File Watchers

If force is set to FALSE, the file watcher must not be referenced by any job, otherwise an error will occur. If you force disabling a file watcher, jobs that depend on it become disabled.

Windows

This means that the window will not open, however, the metadata of the window is still there, so it can be reenabled.

If force is set to FALSE, the window must not be open or referenced by any job otherwise an error occurs.

If force is set to TRUE, disabling a window that is open will succeed but the window will not be closed. It will prevent the window from opening in the future until it is reenabled.

When the window is disabled, those jobs that have the window as their schedule will not be disabled.

Window Groups

When a group of type WINDOW is disabled, jobs (other than a running job) that have the window group as their schedule will not run when the member windows open. However, a job that has one of the window group members as its schedule still runs.

The metadata of the window group is still there, so it can be reenabled. Note that the members of the window group will still open.
If `force` is set to `FALSE`, the window group must not have any members that are open or referenced by any job, otherwise an error will occur.

If `force` is set to `TRUE`:

- The window group is disabled and the open window will be not closed or disabled. It will be allowed to continue to its end.
- The window group is disabled but those jobs that have the window group as their schedule will not be disabled.

Job Chains

When a chain is disabled, the metadata for the chain is still there, but jobs that point to it will not be able to be run. This allows changes to the chain to be made safely without the risk of having an incompletely specified chain run. If `force` is set to `FALSE`, the chain must not be referenced by any job, otherwise an error will occur. If `force` is set to `TRUE`, those jobs that point to the chain will not be disabled, however, they will fail at runtime. Running jobs that point to this chain are not affected by the `DISABLE` call and are allowed to complete.

Database Destinations

When you disable a database destination:

- The destination is skipped when a multiple destination job runs.
- If all destinations are disabled for a job, the Scheduler generates an error when it attempts to run the job.
- The `REFS_ENABLED` column in `<_SCHEDULER_JOB_DESTS` is set to `FALSE` for all jobs that reference the database destination.

External Destinations

When you disable an external destination:

- Dependent database destinations remain enabled, but the Scheduler generates an error when it attempts to run a job with a database destination that depends on the external destination.
- The `REFS_ENABLED` column in `<_SCHEDULER_JOB_DESTS` is set to `FALSE` for all external jobs that reference the external destination and for all database jobs with a database destination that depends on the external destination.

Groups

If you disable an external destination group or database destination group, the Scheduler generates an error when it attempts to run a job that names the group as its destination.

143.6.30 DROP_AGENT_DESTINATION Procedure

This procedure drops one or more external destinations, also known as agent destinations. It should be used only when the preferred method of dropping an external destination, using the `schagent` utility to unregister a Scheduler agent with a database, is unavailable due to failures.

This procedure can be called only by the `SYS` user or a user with the `MANAGE_SCHEDULER` privilege.
Note:
External destinations are created on a source database only implicitly by registering an agent with the database. There is no user-callable CREATE_AGENT_DESTINATION procedure.

Syntax

```sql
DBMS_SCHEDULER.DROP_AGENT_DESTINATION (destination_name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_name</td>
<td>A comma-separated list of external destinations to drop. Because user SYS owns all external destinations, do not prefix them with a schema name.</td>
</tr>
<tr>
<td></td>
<td>The procedure stops processing if it encounters an external destination that does not exist. All external destinations processed before the error are dropped.</td>
</tr>
<tr>
<td></td>
<td>Cannot be NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

When an external destination is dropped:

- All database destinations that refer to the external destination are disabled and their agent attribute is set to NULL.
- Members of external destination groups that refer to the destination are removed from the group.
- All job instances in the *_SCHEDULER_JOB_DESTS views that refer to the external destination are also dropped.
- Jobs running against the destination are stopped.

143.6.31 DROP_CHAIN Procedure

This procedure drops an existing chain.

Syntax

```sql
DBMS_SCHEDULER.DROP_CHAIN (chain_name IN VARCHAR2, force IN BOOLEAN DEFAULT FALSE);
```
Parameters

**Table 143-44 DROP_CHAIN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain to drop. Can also be a comma-delimited list of chains.</td>
</tr>
<tr>
<td>force</td>
<td>If force is set to FALSE, the chain must not be referenced by any job, otherwise an error will occur.</td>
</tr>
<tr>
<td></td>
<td>If force is set to TRUE, all jobs pointing to the chain are disabled before the chain is dropped.</td>
</tr>
<tr>
<td></td>
<td>Running jobs that point to this chain are stopped before the chain is dropped.</td>
</tr>
</tbody>
</table>

**Usage Notes**

Dropping a chain requires alter privileges on the chain (either as the owner, or a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

All steps associated with the chain are dropped. If no rule set was specified when the chain was created, then the automatically created rule set and evaluation context associated with the chain are also dropped, so the user must have the privileges required to do this. See the `DBMS_RULE_ADM.DROP_RULE_SET` and `DBMS_RULE_ADM.DROP_EVALUATION_CONTEXT` procedures for more information.

If force is FALSE, no jobs may be using this chain. If force is TRUE, any jobs that use this chain are disabled before the chain is dropped (and any of these jobs that are running will be stopped).

143.6.32 DROP_CHAIN_RULE Procedure

This procedure removes a rule from an existing chain. The rule object corresponding to this rule will also be dropped. The chain will not be disabled. If dropping this rule makes the chain invalid, the user should first disable the chain to ensure that it does not run.

**Syntax**

```sql
DBMS_SCHEDULER.DROP_CHAIN_RULE (  
    chain_name              IN VARCHAR2,  
    rule_name               IN VARCHAR2,  
    force                   IN BOOLEAN DEFAULT FALSE);
```

**Parameters**

**Table 143-45 DROP_CHAIN_RULE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain to alter</td>
</tr>
<tr>
<td>rule_name</td>
<td>The name of the rule to drop</td>
</tr>
<tr>
<td>force</td>
<td>If force is set to TRUE, the drop operation proceeds even if the chain is currently running. The running chain is not stopped or interrupted. If force is set to FALSE and the chain is running, an error is generated.</td>
</tr>
</tbody>
</table>
Usage Notes

Dropping a chain rule requires alter privileges on the chain (either as the owner or as a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

Dropping a chain rule also drops the underlying rule database object so you must have the privileges to drop this rule object. See the DBMS_RULE_ADM.DROP_RULE procedure for more information.

143.6.33 DROP_CHAIN_STEP Procedure

This procedure drops a chain step. If this chain step is still used in the chain rules, the chain will be disabled.

Syntax

```oracle
DBMS_SCHEDULER.DROP_CHAIN_STEP (  
    chain_name              IN VARCHAR2,
    step_name               IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 143-46  DROP_CHAIN_STEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain to alter</td>
</tr>
<tr>
<td>step_name</td>
<td>The name of the step being dropped. Can be a comma-separated list.</td>
</tr>
<tr>
<td>force</td>
<td>If force is set to TRUE, this succeeds even if this chain is currently running. The running chain will not be stopped or interrupted. If force is set to FALSE and this chain is currently running, an error is thrown.</td>
</tr>
</tbody>
</table>

Usage Notes

Dropping a chain step requires ALTER privileges on the chain (either as the owner or as a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

143.6.34 DROP_CREDENTIALIAL Procedure

This deprecated procedure drops a credential.

Note:

This procedure is deprecated with Oracle Database 12c Release 1 (12.1). While the procedure remains available in this package, for reasons of backward compatibility, Oracle recommends using the alternative enhanced functionality provided in the DBMS_CREDENTIAL package, specifically the DROP_CREDENTIALIAL Procedure.
Syntax

```
DBMS_SCHEDULER.DROP_CREDENTIAL (
    credential_name         IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 143-47  DROP_CREDENTIAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential_name</td>
<td>The name of the credential being dropped. This can optionally be prefixed with a schema name. This cannot be set to NULL.</td>
</tr>
<tr>
<td>force</td>
<td>If set to FALSE, the credential must not be referenced by any job, or an error will occur. If set to TRUE, the credential is dropped whether or not there are jobs referencing it. Jobs that reference the credential will continue to point to a nonexistent credential and throw an error at runtime.</td>
</tr>
</tbody>
</table>

Usage Notes

Only the owner of a credential or a user with the CREATE ANY JOB system privilege may drop the credential.

Running jobs that point to the credential are not affected by this procedure and are allowed to continue.

143.6.35 DROP_DATABASE_DESTINATION Procedure

This procedure drops one or more database destinations.

Syntax

```
DBMS_SCHEDULER.DROP_DATABASE_DESTINATION (
    destination_name        IN VARCHAR2);
```

Parameters

Table 143-48  DROP_DATABASE_DESTINATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_name</td>
<td>The name of the destination to drop. Can be a comma-separated list of database destinations to drop. Each database destination can optionally be prefixed with a schema name. The procedure stops processing if it encounters a database destination that does not exist. All database destinations processed before the error are dropped. Cannot be NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

Only the owner or a user with the CREATE ANY JOB system privilege may drop the database destination.
When a database destination is dropped:

- All job instances that refer to the destination in the *_SCHEDULER_JOB_DESTS views are also dropped.
- Jobs running against the destination are stopped.
- Members of database destination groups that refer to the destination are removed from the group.

**See Also:**

CREATE_DATABASE_DESTINATION Procedure

---

### 143.6.36 DROP_FILE_WATCHER Procedure

This procedure drops one or more file watchers.

**Syntax**

```sql
DBMS_SCHEDULER.DROP_FILE_WATCHER (
    file_watcher_name       IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_watcher_name</td>
<td>The file watcher to drop. Can be a comma-separated list of file watchers. Each file watcher name can optionally be prefixed with a schema name. Cannot be NULL.</td>
</tr>
<tr>
<td>force</td>
<td>If set to FALSE, the file watcher must not be referenced by any job, or an error occurs. If set to TRUE, the file watcher is dropped whether or not there are jobs referencing it. In this case, jobs that reference the dropped file watcher are disabled.</td>
</tr>
</tbody>
</table>

**Usage Notes**

Only the owner of a file watcher or a user with the CREATE ANY JOB system privilege may drop the file watcher.

Running jobs that point to the file watcher are not affected by this procedure and are allowed to continue.

**See Also:**

"CREATE_FILE_WATCHER Procedure"
143.6.37 DROP_GROUP Procedure

This procedure drops one or more groups.

Syntax

DBMS_SCHEDULER.DROP_GROUP (  
group_name       IN VARCHAR2,  
force            IN BOOLEAN DEFAULT FALSE);

Parameters

Table 143-50  DROP_GROUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_name</td>
<td>A group to drop. Can be a comma-separated list of group names. Each group name can optionally be prefixed with a schema name.</td>
</tr>
<tr>
<td></td>
<td>The procedure stops processing if it encounters a group that does not exist. All groups processed before the error are dropped.</td>
</tr>
<tr>
<td></td>
<td>Cannot be NULL.</td>
</tr>
<tr>
<td>force</td>
<td>If FALSE, the group must not be referenced by any job, otherwise an error occurs. If TRUE, the group is dropped whether or not there are jobs</td>
</tr>
<tr>
<td></td>
<td>referencing it. In this case, all jobs referencing the group are disabled and all job instances that reference the group are removed from the</td>
</tr>
<tr>
<td></td>
<td>*_SCHEDULER_JOB_DESTS views.</td>
</tr>
</tbody>
</table>

Usage Notes

Only the owner or a user with the CREATE ANY JOB system privilege may drop a group. You must have the MANAGE SCHEDULER privilege to drop a group of type WINDOW.

See Also:

"CREATE_FILE_WATCHER Procedure"

143.6.38 DROP_INCOMPATIBILITY Procedure

This procedure drops an existing incompatibility definition.

Syntax

DBMS_SCHEDULER.DROP_INCOMPATIBILITY (  
incompatibility_name IN VARCHAR2);
Parameters

Table 143-51  DROP_INCOMPATIBILITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incompatibility_name</td>
<td>The name of the incompatibility definition.</td>
</tr>
</tbody>
</table>

Usage Notes

See Also:
Using Incompatibility Definitions in Oracle Database Administrator’s Guide

143.6.39 DROP_JOB Procedure

This procedure drops one or more jobs or all jobs in one or more job classes. Dropping a job also drops all argument values set for that job.

Syntax

```sql
DBMS_SCHEDULER.DROP_JOB (  
  job_name IN VARCHAR2,  
  force IN BOOLEAN DEFAULT FALSE,  
  defer IN BOOLEAN DEFAULT FALSE,  
  commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 143-52  DROP_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of a job or job class. Can be a comma-delimited list. For a job class, the SYS schema should be specified. If the name of a job class is specified, the jobs that belong to that job class are dropped, but the job class itself is not dropped.</td>
</tr>
<tr>
<td>force</td>
<td>If force is set to TRUE, the Scheduler first attempts to stop the running job instances (by issuing the STOP_JOB call with the force flag set to false), and then drops the jobs.</td>
</tr>
<tr>
<td>defer</td>
<td>If defer is set to TRUE, the Scheduler allows the running jobs to complete and then drops the jobs.</td>
</tr>
</tbody>
</table>
Table 143-52  (Cont.) DROP_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit_semantics</td>
<td>The commit semantics. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>· STOP_ON_FIRST_ERROR returns on the first error and previous successful drop operations are committed to disk. This is the default.</td>
</tr>
<tr>
<td></td>
<td>· TRANSACTIONAL returns on the first error. Everything that happened before that error is rolled back. This type is not supported when force is set to TRUE.</td>
</tr>
<tr>
<td></td>
<td>· ABSORB_ERRORS tries to absorb any errors and drop the rest of the jobs, and commits all the successful drops. If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for details.</td>
</tr>
<tr>
<td></td>
<td>Only STOP_ON_FIRST_ERROR is permitted when job classes are included in the job_name list.</td>
</tr>
</tbody>
</table>

Usage Notes

If both force and defer are set to FALSE and a job is running at the time of the call, the attempt to drop that job fails. The entire call to DROP_JOB may then fail, depending on the setting of commit_semantics.

Setting both force and defer to TRUE results in an error.

Dropping a job requires ALTER privileges on the job either as the owner of the job or as a user with the ALTER object privilege on the job or the CREATE ANY JOB system privilege.

143.6.40 DROP_JOB_CLASS Procedure

This procedure drops a job class. Dropping a job class means that all the metadata about the job class is removed from the database.

Syntax

```sql
DBMS_SCHEDULER.DROP_JOB_CLASS (  
job_class_name IN VARCHAR2,  
force IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 143-53  DROP_JOB_CLASS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_class_name</td>
<td>The name of the job class. Can be a comma-delimited list.</td>
</tr>
<tr>
<td>force</td>
<td>If force is set to FALSE, a class being dropped must not be referenced by any jobs, otherwise an error occurs.</td>
</tr>
<tr>
<td></td>
<td>If force is set to TRUE, jobs belonging to the class are disabled and their class is set to the default class. Only if this is successful is the class dropped.</td>
</tr>
<tr>
<td></td>
<td>Running jobs that belong to the job class are not affected.</td>
</tr>
</tbody>
</table>
Usage Notes

Dropping a job class requires the `MANAGE_SCHEDULER` system privilege.

### 143.6.41 DROP_PROGRAM Procedure

This procedure drops a program. Any arguments that pertain to the program are also dropped when the program is dropped.

**Syntax**

```sql
DBMS_SCHEDULER.DROP_PROGRAM (
    program_name            IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>program_name</code></td>
<td>The name of the program to be dropped. Can be a comma-delimited list.</td>
</tr>
<tr>
<td><code>force</code></td>
<td>If <code>force</code> is set to <code>FALSE</code>, the program must not be referenced by any job, otherwise an error occurs. If <code>force</code> is set to <code>TRUE</code>, all jobs referencing the program are disabled before the program is dropped. Running jobs that point to the program are not affected by the <code>DROP_PROGRAM</code> call and are allowed to continue.</td>
</tr>
</tbody>
</table>

**Usage Notes**

Dropping a program requires that you be the owner of the program or have `ALTER` privileges on that program. You can also drop a program if you have the `CREATE ANY JOB` privilege.

### 143.6.42 DROP_PROGRAM_ARGUMENT Procedure

This procedure drops a program argument. An argument can be specified by either name (if one has been given) or position.

The procedure is overloaded.

**Syntax**

- Drops a program argument by position:

  ```sql
  DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (  
    program_name            IN VARCHAR2,
    argument_position       IN PLS_INTEGER);
  ```

- Drops a program argument by name:

  ```sql
  DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (  
    program_name            IN VARCHAR2,
    argument_name           IN VARCHAR2);
  ```
Parameters

Table 143-55 DROP_PROGRAM_ARGUMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_name</td>
<td>The name of the program to be altered. A program with this name must exist.</td>
</tr>
<tr>
<td>argument_name</td>
<td>The name of the argument being dropped</td>
</tr>
<tr>
<td>argument_position</td>
<td>The position of the argument to be dropped</td>
</tr>
</tbody>
</table>

Usage Notes

Dropping a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also drop a program argument if you have the CREATE ANY JOB privilege.

143.6.43 DROP_RESOURCE Procedure

This procedure drops a resource.

Syntax

```sql
DBMS_SCHEDULER.DROP_RESOURCE (
    resource_name  IN VARCHAR2,
    force          IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 143-56 DROP_RESOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource_name</td>
<td>The name of the resource to be dropped. Can be a comma-delimited list.</td>
</tr>
<tr>
<td>force</td>
<td>If force is set to FALSE, the resource must not have any existing constraints, otherwise an error occurs. If force is set to TRUE, the resource will be dropped and any constraints defined on this resource will also be dropped.</td>
</tr>
</tbody>
</table>

Usage Notes

Only the owner or a user with the CREATE ANY JOB system privilege may drop the resource.

See Also:

Creating or dropping a Resource in *Oracle Database Administrator’s Guide*
143.6.44 DROP_SCHEDULE Procedure

This procedure drops a schedule.

Syntax

```sql
DBMS_SCHEDULER.DROP_SCHEDULE (
    schedule_name    IN VARCHAR2,
    force            IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 143-57  DROP_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_name</td>
<td>The name of the schedule. Can be a comma-delimited list.</td>
</tr>
<tr>
<td>force</td>
<td>If <code>force</code> is set to <code>FALSE</code>, the schedule must not be referenced by any</td>
</tr>
<tr>
<td></td>
<td>job or window, otherwise an error will occur.</td>
</tr>
<tr>
<td></td>
<td>If <code>force</code> is set to <code>TRUE</code>, any jobs or windows that use this schedule are</td>
</tr>
<tr>
<td></td>
<td>disabled before the schedule is dropped</td>
</tr>
<tr>
<td></td>
<td>Running jobs and open windows that point to the schedule are not affected.</td>
</tr>
</tbody>
</table>

Usage Notes

You must be the owner of the schedule being dropped or have `ALTER` privileges for the schedule or the `CREATE ANY JOB` privilege.

143.6.45 DROP_WINDOW Procedure

This procedure drops a window. All metadata about the window is removed from the database. The window is removed from any groups that reference it.

Syntax

```sql
DBMS_SCHEDULER.DROP_WINDOW (
    window_name             IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 143-58  DROP_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_name</td>
<td>The name of the window. Can be a comma-delimited list.</td>
</tr>
</tbody>
</table>
Table 143-58  (Cont.) DROP_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>If <code>force</code> is set to <code>FALSE</code>, the window must be not be open or referenced by any job, otherwise an error occurs. If <code>force</code> is set to <code>TRUE</code>, the window is dropped and those jobs that have the window as their schedule are disabled. However, jobs that have a window group, of which the dropped window is a member, as their schedule, are not disabled. If the window is open then, the Scheduler attempts to first close the window and then drop it. When the window is closed, normal close window rules apply. Running jobs that have the window as their schedule is allowed to continue, unless the <code>stop_on_window_close</code> flag is set to <code>TRUE</code> for the job. If this is the case, the job is stopped when the window is dropped.</td>
</tr>
</tbody>
</table>

Usage Notes

Dropping a window requires the `MANAGE SCHEDULER` privilege.

143.6.46 ENABLE Procedure

This procedure enables a program, job, chain, window, database destination, external destination, file watcher, or group.

When an object is enabled, its `enabled` attribute is set to `TRUE`. By default, jobs, chains, and programs are created disabled and database destinations, external destinations, file watchers, windows, and groups are created enabled.

If a job was disabled and you enable it, the Scheduler begins to automatically run the job according to its schedule. Enabling a disabled job also resets the job `RUN_COUNT`, `FAILURE_COUNT` and `RETRY_COUNT` columns in the `_SCHEDULER_JOBS` data dictionary views.

Validity checks are performed before enabling an object. If the check fails, the object is not enabled, and an appropriate error is returned. This procedure does not return an error if the object was already enabled.

Syntax

```sql
DBMS_SCHEDULER.ENABLE (  
    name IN VARCHAR2,  
    commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```
Parameters

Table 143-59   ENABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the Scheduler object being enabled. Can be a comma-delimited list of names. If a job class name is specified, then all the jobs in the job class are enabled. If a group name is specified, then the group is enabled, but the enabled state of the group members is unaffected.</td>
</tr>
<tr>
<td>commit_semantics</td>
<td>The commit semantics. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>·  STOP_ON_FIRST_ERROR - The procedure returns on the first error and previous successful enable operations are committed to disk. This is the default.</td>
</tr>
<tr>
<td></td>
<td>·  TRANSACTIONAL - The procedure returns on the first error and everything that happened before that error is rolled back. This type is only supported when enabling a job or a list of jobs.</td>
</tr>
<tr>
<td></td>
<td>·  ABSORB_ERRORS - The procedure tries to absorb any errors and enable the rest of the jobs. It commits all the enable operations that were successful. If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for details. This type is only supported when enabling a job or a list of jobs.</td>
</tr>
</tbody>
</table>

Usage Notes

Window names must be preceded by SYS.

To run ENABLE for a window or group of type WINDOW, you must have the MANAGE SCHEDULER privilege. For a job of type EXECUTABLE (or for a job that points to a program of type EXECUTABLE), the job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.

To enable a file watcher, the file watcher owner must have the EXECUTE privilege on the designated credential.

You can use ENABLE with any schema except the SYS schema.

143.6.47 END_DETACHED_JOB_RUN Procedure

This procedure ends a detached job run. A detached job points to a detached program, which is a program with the detached attribute set to TRUE.

A detached job run does not end until this procedure or the STOP_JOB Procedure is called.

Syntax

```sql
DBMS_SCHEDULER.END_DETACHED_JOB_RUN (  
job_name          IN VARCHAR2,  
error_number      IN PLS_INTEGER DEFAULT 0,  
additional_info   IN VARCHAR2 DEFAULT NULL);
```
### Parameters

**Table 143-60 END_DETACHED_JOB_RUN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of the job to end. Must be a detached job that is running.</td>
</tr>
<tr>
<td>error_number</td>
<td>If zero, then the job run is logged as succeeded. If -1013, then the job run is logged as stopped. If any other number, then the job run is logged as failed with that error number.</td>
</tr>
<tr>
<td>additional_info</td>
<td>This text is stored in the additional_info column of the *_scheduler_job_run_details views for this job run.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure requires that you either own the job or have alter privileges on it. You can also end any detached job run if you have the create any job privilege.

**See Also:**

Oracle Database Administrator's Guide for information about detached jobs.

### 143.6.48 EVALUATE_CALENDAR_STRING Procedure

You can define repeat intervals of jobs, windows or schedules using the Scheduler calendaring syntax. This procedure evaluates the calendar expression and tells you the next execution date and time of a job or window. This is very useful for testing the correct definition of the calendar string without actually scheduling the job or window.

This procedure can also get multiple steps of the repeat interval by passing the next_run_date returned by one invocation as the return_date_after argument of the next invocation.

See the calendaring syntax described in "Operational Notes".

**Syntax**

```sql
DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING (    calendar_string    IN  VARCHAR2,    start_date         IN  TIMESTAMIP WITH TIME ZONE,    return_date         IN  TIMESTAMIP WITH TIME ZONE,    next_run_date      OUT TIMESTAMIP WITH TIME ZONE);
```

**Parameters**

**Table 143-61 EVALUATE_CALENDAR_STRING Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calendar_string</td>
<td>The calendar expression to be evaluated. The string must be in the calendaring syntax described in &quot;Operational Notes&quot;.</td>
</tr>
</tbody>
</table>
Table 143-61  (Cont.) EVALUATE_CALENDAR_STRING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>The date and time after which the repeat interval becomes valid. It can also be used to fill in specific items that are missing from the calendar string. Can optionally be NULL.</td>
</tr>
<tr>
<td>return_date_after</td>
<td>The return_date_after argument helps the Scheduler determine which one of all possible matches (all valid execution dates) to return from those determined by the start_date and the calendar string. When a NULL value is passed for this argument, the Scheduler automatically fills in systimestamp as its value.</td>
</tr>
<tr>
<td>next_run_date</td>
<td>The first timestamp that matches the calendar string and start date that occur after the value passed in for the return_date_after argument.</td>
</tr>
</tbody>
</table>

Examples

The following code fragment can be used to determine the next five dates a job will run given a specific calendar string.

```sql
SET SERVEROUTPUT ON;
ALTER SESSION set NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
BEGIN
   DECLARE
      start_date        TIMESTAMP;
      return_date_after TIMESTAMP;
      next_run_date     TIMESTAMP;
      FOR i IN 1..5 LOOP
         start_date := to_timestamp_tz('01-JAN-2003 10:00:00','DD-MON-YYYY HH24:MI:SS');
         return_date_after := start_date;
         DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING('FREQ=DAILY;BYHOUR=9;BYMINUTE=30;BYDAY=MON,TUE,WED,THU,FRI', start_date, return_date_after, next_run_date);
         DBMS_OUTPUT.PUT_LINE('next_run_date: ' || next_run_date);
         return_date_after := next_run_date;
      END LOOP;
END;
/
```

```
next_run_date: 02-JAN-03 09.30.00.000000 AM
next_run_date: 03-JAN-03 09.30.00.000000 AM
next_run_date: 06-JAN-03 09.30.00.000000 AM
next_run_date: 07-JAN-03 09.30.00.000000 AM
next_run_date: 08-JAN-03 09.30.00.000000 AM
```

PL/SQL procedure successfully completed.

Usage Notes

No specific Scheduler privileges are required.
143.6.49 EVALUATE_RUNNING_CHAIN Procedure

This procedure forces reevaluation of the rules of a running chain to trigger any rules for which the conditions have been satisfied. The job passed as an argument must point to a chain and must be running. If the job is not running, an error is thrown. (RUN_JOB can be used to start the job.)

If any of the steps of the chain are themselves running chains, another EVALUATE_RUNNING_CHAIN is performed on each of the nested running chains.

Syntax

```
DBMS_SCHEDULER.EVALUATE_RUNNING_CHAIN (job_name IN VARCHAR2);
```

Parameters

Table 143-62  EVALUATE_RUNNING_CHAIN Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of the running job (pointing to a chain) to reevaluate the rules for</td>
</tr>
</tbody>
</table>

Usage Notes

Running EVALUATE_RUNNING_CHAIN on a job requires alter privileges on the job (either as the owner, or as a user with ALTER privileges on the job or the CREATE ANY JOB system privilege).

Note:

The Scheduler automatically evaluates a chain:

- At the start of the chain job
- When a chain step completes
- When an event occurs that is associated with an event step in the chain

For most chains, this is sufficient. EVALUATE_RUNNING_CHAIN should be used only under the following circumstances:

- After manual intervention of a running chain with the ALTER_RUNNING_CHAIN procedure
- When chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler.

In these cases, EVALUATE_RUNNING_CHAIN may not be needed if you set the evaluation_interval attribute when you created the chain.
143.6.50 GENERATE_JOB_NAME Function

This function returns a unique name for a job.

The name will be of the form \{prefix\}N where \(N\) is a number from a sequence. If no prefix is specified, the generated name will, by default, be \(\text{JOB}_1, \text{JOB}_2, \text{JOB}_3, \) and so on. If 'SCOTT' is specified as the prefix, the name will be \(\text{SCOTT}_1, \text{SCOTT}_2, \) and so on.

Syntax

```sql
DBMS_SCHEDULER.GENERATE_JOB_NAME (prefix IN VARCHAR2 DEFAULT 'JOB$_') RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>The prefix to use when generating the job name</td>
</tr>
</tbody>
</table>

Usage Notes

If the prefix is explicitly set to NULL, the name is just the sequence number. In order to successfully use such numeric names, they must be surrounded by double quotes throughout the DBMS_SCHEDULER calls. A prefix cannot be longer than 18 characters and cannot end with a digit.

Note that, even though the GENERATE_JOB_NAME function never returns the same job name twice, there is a small chance that the returned name matches an already existing database object.

No specific Scheduler privileges are required to use this function.

143.6.51 GET_AGENT_INFO Function

This function can return job information specific to an agent, such as how many are running and so on, depending on the attribute selected.

Syntax

```sql
DBMS_SCHEDULER.GET_AGENT_INFO (agent_name IN VARCHAR2, attribute IN VARCHAR2) RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>The name of an external destination where the agent is running</td>
</tr>
</tbody>
</table>
### Table 143-64  (Cont.) GET_AGENT_INFO Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| attribute | Possible Attributes values  
  - VERSION: Returns the agent version number. Requires the CREATE JOB system privilege.  
  - UPTIME: Returns the time the agent has been up and running. Requires the CREATE JOB system privilege.  
  - NUMBER_OF_RUNNING_JOBS: Returns the number of jobs that the agent is currently running. Requires the CREATE JOB system privilege.  
  - TOTAL_JOBS_RUN: Returns the number of jobs run by the agent since it was started. Requires the CREATE JOB system privilege.  
  - RUNNING_JOBS: Returns a comma-separated list of the names of the jobs running currently. Requires the MANAGE SCHEDULER system privilege.  
  - ALL: Returns all the information the previous options return. Requires the MANAGE SCHEDULER system privilege. |

**Usage Notes**

This function returns the same information as the `schagent` utility status option. See *Oracle Database Administrator's Guide*.

### 143.6.52 GET_AGENT_VERSION Function

This function returns the version string of a Scheduler agent that is registered with the database and is currently running. `GET_AGENT_VERSION` throws an error if the agent is not registered with the database or if the agent is not currently running.

**Syntax**

```sql
DBMS_SCHEDULER.GET_AGENT_VERSION (agent_host IN VARCHAR2) RETURN VARCHAR2;
```

**Parameters**

### Table 143-65  GET_AGENT_VERSION Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_host</td>
<td>Either the hostname and port on which the agent is running in the form hostname:port or the name of the agent as shown in the destination_name column of the ALL_SCHEDULER_EXTERNAL_DESTS view which lists all Scheduler agents registered with the database.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This function requires the CREATE EXTERNAL JOB system privilege.
143.6.53 GET_ATTRIBUTE Procedure

This procedure retrieves the value of an attribute of a Scheduler object. It is overloaded to retrieve values of various types.

Syntax

```sql
DBMS_SCHEDULER.GET_ATTRIBUTE (
    name           IN VARCHAR2,
    attribute      IN VARCHAR2,
    value          OUT {VARCHAR2|PLS_INTEGER|BOOLEAN|DATE|TIMESTAMP|
                    TIMESTAMP WITH TIME ZONE|TIMESTAMP WITH LOCAL TIME ZONE|
                    INTERVAL DAY TO SECOND});
```

```sql
DBMS_SCHEDULER.GET_ATTRIBUTE (
    name           IN VARCHAR2,
    attribute      IN VARCHAR2,
    value          OUT VARCHAR2,
    value2         OUT VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the object</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute being retrieved. See the SET_ATTRIBUTE Procedure for tables of attribute values.</td>
</tr>
<tr>
<td>value</td>
<td>The existing value of the attribute</td>
</tr>
<tr>
<td>value2</td>
<td>The value2 argument is for an optional second value. Most attributes have only one value associated with them, but some can have two.</td>
</tr>
</tbody>
</table>

Usage Notes

To run `GET_ATTRIBUTE` for a job class, you must have the `MANAGE SCHEDULER` privilege or have `EXECUTE` privileges on the class. For a schedule, window, or group, no privileges are necessary. Otherwise, you must be the owner of the object or have `ALTER` or `EXECUTE` privileges on that object or have the `CREATE ANY JOB` privilege.

See the `SET_ATTRIBUTE Procedure` for tables of attribute values that you can retrieve for the various Scheduler object types.

143.6.54 GET_FILE Procedure

This procedure retrieves a file from the operating system file system of a specified host. The file is copied to a destination, or its contents are returned in a procedure output parameter.

You can also use this procedure to retrieve the standard output or error text for a run of an external job that has an associated credential.
This procedures differs from the equivalent UTL_FILE procedure in that it uses a credential and can retrieve files from remote hosts that have only a Scheduler agent (and not an Oracle database) installed.

**Syntax**

```
DBMS_SCHEDULER.GET_FILE (  
    source_file                  IN VARCHAR2,  
    source_host                  IN VARCHAR2,  
    credential_name              IN VARCHAR2,  
    file_contents                IN OUT NOCOPY {BLOB|CLOB});
```

```
DBMS_SCHEDULER.GET_FILE (  
    source_file                  IN VARCHAR2,  
    source_host                  IN VARCHAR2,  
    credential_name              IN VARCHAR2,  
    destination_file_name        IN VARCHAR2,  
    destination_directory_object IN VARCHAR2,  
    destination_permissions      IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

**Table 143-67  GET_FILE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_file</td>
<td>Fully qualified path name of the file to retrieve from the operating system. The file name is case-sensitive and is not converted to uppercase. If the file name starts with a question mark ('?'), the question mark is replaced by the path to the Oracle home if getting a file from the local host, or to the Scheduler agent home if getting a file from a remote host. If the format of this parameter is external_log_id_stdout, then the stdout from the designated external job run is returned. If the format of this parameter is external_log_id_stderr, the error text from the designated external job run is returned. You obtain the value of external_log_id from the ADDITIONAL_INFO column of the *_SCHEDULER_JOB_RUN_DETAILS views. This column contains a set of name/value pairs in an indeterminate order, so you must parse this column for the external_log_id name/value pair, and then append either &quot;_stdout&quot; or &quot;_stderr&quot; to its value. The external job must have an associated credential. The credential_name parameter of GET_FILE must name the same credential that is used by the job, and the source_host parameter must be the same as the destination attribute of the job.</td>
</tr>
<tr>
<td>source_host</td>
<td>If the file is to be retrieved from a remote host, then this parameter must be a valid an external destination name. (An external destination is created when you register a remote Scheduler agent with the database. You can view external destination names in the views *_SCHEDULER_EXTERNAL_DESTS.) If source_host is NULL or set to 'localhost', then the file is retrieved from the file system of the local host. To determine the port number of a Scheduler agent, view the schagent.conf file, which is located in the Scheduler agent home directory on the remote host.</td>
</tr>
<tr>
<td>credential_name</td>
<td>The name of the credential to use for accessing the file system.</td>
</tr>
<tr>
<td>file_contents</td>
<td>The variable into which the file contents is read.</td>
</tr>
</tbody>
</table>
Table 143-67  (Cont.) GET_FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_file_name</td>
<td>The file to which the file contents is written.</td>
</tr>
<tr>
<td>destination_directory_object</td>
<td>The directory object that specifies the path to the destination file, when destination_file_name is used. The caller must have the necessary privileges on the directory object.</td>
</tr>
<tr>
<td>destination_permissions</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

Usage Notes

The caller must have the CREATE EXTERNAL JOB system privilege and have EXECUTE privileges on the credential.

143.6.55 GET_SCHEDULER_ATTRIBUTE Procedure

This procedure retrieves the value of a Scheduler attribute.

Syntax

```sql
DBMS_SCHEDULER.GET_SCHEDULER_ATTRIBUTE ( attribute IN VARCHAR2, value OUT VARCHAR2 );
```

Parameters

Table 143-68  GET_SCHEDULER_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>The name of the attribute</td>
</tr>
<tr>
<td>value</td>
<td>The existing value of the attribute</td>
</tr>
</tbody>
</table>

Usage Notes

To run GET_SCHEDULER_ATTRIBUTE, you must have the MANAGE SCHEDULER privilege.

Table 143-69 lists the Scheduler attributes that you can retrieve. For more detail on these attributes, see Table 143-101 and the section "Configuring the Scheduler" in Oracle Database Administrator's Guide.

Table 143-69  Scheduler Attributes Retrievable with GET_SCHEDULER_ATTRIBUTE

<table>
<thead>
<tr>
<th>Scheduler Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_open_window</td>
<td>Name of the currently open window</td>
</tr>
<tr>
<td>default_timezone</td>
<td>Default time zone used by the Scheduler for repeat intervals and windows</td>
</tr>
</tbody>
</table>
**Table 143-69  (Cont.) Scheduler Attributes Retrievable with GET_SCHEDULER_ATTRIBUTE**

<table>
<thead>
<tr>
<th>Scheduler Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>email_sender</td>
<td>The default e-mail address of the sender for job state e-mail notifications</td>
</tr>
<tr>
<td>email_server</td>
<td>The SMTP server address that the Scheduler uses to send e-mail notifications for job state events. E-mail notifications cannot be sent if this attribute is NULL.</td>
</tr>
<tr>
<td>event_expiry_time</td>
<td>Time in seconds before an event generated by the Scheduler and enqueued onto the Scheduler event queue expires. May be NULL.</td>
</tr>
<tr>
<td>log_history</td>
<td>Retention period in days for job and window logs. The range of valid values is 0 through 1000000.</td>
</tr>
<tr>
<td>max_job_slave_processes</td>
<td>This Scheduler attribute is not used.</td>
</tr>
</tbody>
</table>

### 143.6.56 OPEN_WINDOW Procedure

This procedure manually opens a window, unrelated to its schedule.

The window opens and the resource plan associated with it takes effect immediately, for the duration specified or for the normal duration of the window, if no duration is given. Only an enabled window can be manually opened.

**Syntax**

```sql
DBMS_SCHEDULER.OPEN_WINDOW (  
    window_name             IN VARCHAR2,  
    duration                IN INTERVAL DAY TO SECOND,  
    force                   IN BOOLEAN DEFAULT FALSE);  
```

**Parameters**

**Table 143-70  OPEN_WINDOW Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_name</td>
<td>The name of the window</td>
</tr>
<tr>
<td>duration</td>
<td>The duration of the window. It is of type interval day to second. If it is NULL, then the window opens for the regular duration as specified in the window metadata.</td>
</tr>
</tbody>
</table>
Table 143-70  (Cont.) OPEN_WINDOW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>If force is set to FALSE, then opening an already open window generates an error. If force is set to TRUE: You can open a window that is already open. The window stays open for the duration specified in the call, from the time the OPEN_WINDOW command was issued. For example: window1 was created with a duration of four hours. It has how been open for two hours. If, at this point, you reopen window1 using the OPEN_WINDOW call and do not specify a duration, then window1 stays open for four hours because it was created with that duration. If you specified a duration of 30 minutes, the window will close in 30 minutes. The Scheduler automatically closes any window that is open at that time, even if it has a higher priority. For the duration of this manually opened window, the Scheduler does not open any other scheduled windows even if they have a higher priority.</td>
</tr>
</tbody>
</table>

Usage Notes

Opening a window manually has no impact on regular scheduled runs of the window. The next open time of the window is not updated and is determined by the regular scheduled opening.

When a window that was manually opened closes, the rules about overlapping windows are applied to determine which other window should be opened at that time if any at all.

If there are jobs running when the window opens, the resources allocated to them might change if there is a switch in resource plan.

If a window fails to switch resource plans because the designated resource plan no longer exists or because resource plan switching by windows is disabled (for example, by using the ALTER SYSTEM statement with the force option), the failure to switch resource plans is recorded in the window log.

Opening a window requires the MANAGE SCHEDULER privilege.

143.6.57 PURGE_LOG Procedure

The PURGE_LOG procedure purges rows from the job and window log that were not purged automatically by the scheduler.

By default, the Scheduler automatically purges all rows in the job log and window log that are older than 30 days. The PURGE_LOG procedure can be used to purge additional rows from the job and window log.

Rows in the job log table pertaining to the steps of a chain are purged only when the entry for the main chain job is purged (either manually or automatically).

Syntax

```sql
DBMS_SCHEDULER.PURGE_LOG (log_history IN PLS_INTEGER DEFAULT 0,
```

```sql
```
Parameters

Table 143-71  PURGE_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_history</td>
<td>This specifies how much history (in days) to keep. The valid range is 0 - 1000000. If set to 0, no history is kept.</td>
</tr>
<tr>
<td>which_log</td>
<td>This specifies the log type. Valid values are: job_log, window_log, and job_and_window_log.</td>
</tr>
<tr>
<td>job_name</td>
<td>This specifies which job-specific entries must be purged from the job log. This can be a comma-delimited list of job names and job classes. Whenever job_name has a value other than NULL, the which_log argument implicitly includes the job log.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure requires the MANAGE_SCHEDULER privilege.

Examples

The following completely purges all rows from both the job log and the window log:

```
DBMS_SCHEDULER.PURGE_LOG();
```

The following purges all rows from the window log that are older than 5 days:

```
DBMS_SCHEDULER.PURGE_LOG(5, 'window_log');
```

The following purges all rows from the window log that are older than 1 day and all rows from the job log that are related to jobs in jobclass1 and older than 1 day:

```
DBMS_SCHEDULER.PURGE_LOG(1, 'job_and_window_log', 'sys.jobclass1');
```

143.6.58 PUT_FILE Procedure

This procedure saves a file to the operating system file system of a specified remote host or of the local computer.

It differs from the equivalent UTL_FILE procedure in that it uses a credential and can save files to a remote host that has only a Scheduler agent (and not an Oracle Database) installed.

Syntax

```
DBMS_SCHEDULER.PUT_FILE (  
  destination_file         IN VARCHAR2,  
  destination_host         IN VARCHAR2,  
  credential_name          IN VARCHAR2,  
  file_contents            IN {BLOB|CLOB},  
  destination_permissions  IN VARCHAR2 DEFAULT NULL);  
```

```
DBMS_SCHEDULER.PUT_FILE (  
  destination_file         IN VARCHAR2,  
  destination_host         IN VARCHAR2,  
  credential_name          IN VARCHAR2,  
```
source_file_name         IN VARCHAR2,
source_directory_object  IN VARCHAR2,
destination_permissions  IN VARCHAR2 DEFAULT NULL);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_file</td>
<td>Fully qualified path name of the file to save to the operating system file system. The file name is case-sensitive. If the file name starts with a question mark (&quot;?&quot;), the question mark is replaced by the path to the Oracle home if saving to the local host, or to the Scheduler agent home if saving to a remote host.</td>
</tr>
<tr>
<td>destination_host</td>
<td>If NULL or set to 'localhost', the file is saved to the file system of the local computer. To save to a remote host, this parameter must be a valid external destination name. (An external destination is created when you register a remote Scheduler agent with the database. You can view external destination names in the views *_SCHEDULER_EXTERNAL_DESTS.)</td>
</tr>
<tr>
<td>credential_name</td>
<td>The name of the credential to use for accessing the destination file system.</td>
</tr>
<tr>
<td>file_contents</td>
<td>The variable from which the file contents is read.</td>
</tr>
<tr>
<td>source_file_name</td>
<td>The file from which the file contents is written</td>
</tr>
<tr>
<td>source_directory_object</td>
<td>The directory object that specifies the path to the source file, when source_file_name is used. The caller must have the necessary privileges on the directory object.</td>
</tr>
<tr>
<td>destination_permissions</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

Usage Notes

The caller must have the CREATE EXTERNAL JOB system privilege and have EXECUTE privileges on the credential.

143.6.59 REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure

This procedure unsubscribes a user from the Scheduler event queue SYS.SCHEDULER$_EVENT_QUEUE.

Syntax

```sql
DBMS_SCHEDULER.REMOVE_EVENT_QUEUE_SUBSCRIBER ( subscriber_name       IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 143-73  REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriber_name</td>
<td>Name of the Oracle Streams Advanced Queuing (AQ) agent to remove the</td>
</tr>
<tr>
<td></td>
<td>subscription from. If NULL, the user name of the calling user is used.</td>
</tr>
</tbody>
</table>

Usage Notes

After the agent is unsubscribed, it is deleted. If the agent does not exist or is not currently subscribed to the Scheduler event queue, an error is raised.

143.6.60 REMOVE_FROM_INCOMPATIBILITY Procedure

This procedure removes jobs or programs from an existing incompatibility definition.

Syntax

```
DBMS_SCHEDULER.REMOVE_FROM_INCOMPATIBILITY (
    incompatibility_name    IN VARCHAR2,
    object_name             IN VARCHAR2);
```

Parameters

Table 143-74  REMOVE_FROM_INCOMPATIBILITY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incompatibility_name</td>
<td>The name of the incompatibility definition.</td>
</tr>
<tr>
<td>object_name</td>
<td>One or more (comma-separated) programs or jobs</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure does not raise an error if any specified objects do not already exist in the incompatibility definition.

See Also:

Using Incompatibility Definitions in Oracle Database Administrator's Guide

143.6.61 REMOVE_GROUP_MEMBER Procedure

This procedure removes one or more members from an existing group.

Syntax

```
DBMS_SCHEDULER.REMOVE_GROUP_MEMBER ( 
    group_name              IN VARCHAR2,
    member                  IN VARCHAR2);
```


### Parameters

#### Table 143-75  REMOVE_GROUP_MEMBER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_name</td>
<td>The name of the group.</td>
</tr>
<tr>
<td>member_name</td>
<td>The name of the member to remove from group. Comma-separated list of members to remove. An error is returned if any of the members is not part of the group. A group of the same type can be named as a member. The Scheduler immediately expands the included group name into its list of members. If the member is a destination, any job instances that run on this destination are removed from the *_SCHEDULER_JOB_DESTS views.</td>
</tr>
</tbody>
</table>

#### Usage Notes

The following users may remove members from a group:

- The group owner
- A user that has been granted the ALTER object privilege on the group
- A user with the CREATE ANY JOB system privilege

You must have the MANAGE SCHEDULER privilege to remove a member from a group of type WINDOW.

### See Also:

"CREATE_GROUP Procedure"

### 143.6.62 REMOVE_JOB_EMAIL_NOTIFICATION Procedure

This procedure removes e-mail notifications for a job. You can remove all e-mail notifications or remove notifications only for specified recipients or specified events.

#### Syntax

```sql
DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION (  
  job_name IN VARCHAR2,  
  recipients IN VARCHAR2 DEFAULT NULL,  
  events IN VARCHAR2 DEFAULT NULL);
```

#### Parameters

#### Table 143-76  ADD_JOB_EMAIL_NOTIFICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Name of the job to remove e-mail notifications for. Cannot be NULL.</td>
</tr>
</tbody>
</table>
Table 143-76 (Cont.) ADD_JOB_EMAIL_NOTIFICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipients</td>
<td>E-mail address to remove e-mail notification for. Comma-separated list of e-mail addresses.</td>
</tr>
<tr>
<td>events</td>
<td>Job state event to remove e-mail notification for. Comma-separate list of job state events.</td>
</tr>
</tbody>
</table>

Usage Notes

When you specify multiple recipients and multiple events, the notification for each specified event is removed for each specified recipient. The procedure ignores any recipients or events that are specified but that were not previously added.

If recipients is NULL, e-mail notifications for the specified events are removed for all existing recipients. If events is NULL, notifications for all events are removed for the specified recipients. If both recipients and events are NULL, all e-mail notifications are removed for the job.

For example, if recipients is 'jsmith@example.com, rjones@example.com' and events is 'JOB_FAILED, JOB_BROKEN', then notifications for both the JOB_FAILED and JOB_BROKEN events are removed for both jsmith and rjones. If recipients is NULL, then notifications for both the JOB_FAILED and JOB_BROKEN events are removed for jsmith, rjones, and any other previously defined recipients for these events.

To call this procedure, you must be the job owner or a user with the CREATE ANY JOB system privilege or ALTER object privilege on the job.

See Also:

"ADD_JOB_EMAIL_NOTIFICATION Procedure"

143.6.63 RESET_JOB_ARGUMENT_VALUE Procedure

This procedure resets (clears) the value previously set to an argument for a job.

RESET_JOB_ARGUMENT_VALUE is overloaded.

Syntax

Clears a previously set job argument value by argument position:

```sql
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (  
    job_name                IN VARCHAR2,  
    argument_position       IN PLS_INTEGER);
```

Clears a previously set job argument value by argument name:

```sql
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (  
    job_name                IN VARCHAR2,  
    argument_name           IN VARCHAR2);
```
Parameters

Table 143-77  RESET_JOB_ARGUMENT_VALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of the job being altered</td>
</tr>
<tr>
<td>argument_position</td>
<td>The position of the program argument being reset</td>
</tr>
<tr>
<td>argument_name</td>
<td>The name of the program argument being reset</td>
</tr>
</tbody>
</table>

Usage Notes

If the corresponding program argument has no default value, the job is disabled.
Resetting a program argument of a job belonging to another user requires ALTER privileges on that job. Arguments can be specified by position or by name.

RESET_JOB_ARGUMENT_VALUE requires that you be the owner of the job or have ALTER privileges on that job. You can also reset a job argument value if you have the CREATE ANY JOB privilege.

RESET_JOB_ARGUMENT_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

143.6.64 RUN_CHAIN Procedure

This procedure immediately runs a chain or part of a chain by creating a run-once job with the job name given.

If no job_name is given, one is generated of the form RUN_CHAIN$_chainnameN, where chainname is the first 8 characters of the chain name and N is an integer.

If a list of start steps is given, only those steps are started when the chain begins running. Steps not in the list that would normally have started are skipped and paused (so that they or the steps after them do not run).

If start_steps is NULL, then the chain starts normally—that is, it performs an initial evaluation to see which steps to start running).

If a list of initial step states is given, the newly created chain job sets every listed step to the state specified for that step before evaluating the chain rules to see which steps to start. (Steps in the list are not started.)

Syntax

Runs a chain, with a list of start steps.

DBMS_SCHEDULER.RUN_CHAIN (  
    chain_name    IN VARCHAR2,  
    start_steps   IN VARCHAR2,  
    job_name      IN VARCHAR2 DEFAULT NULL);  

Runs a chain, with a list of initial step states.

DBMS_SCHEDULER.RUN_CHAIN (  
    chain_name    IN VARCHAR2,  

Parameters

Table 143-78  RUN_CHAIN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_name</td>
<td>The name of the chain to run</td>
</tr>
<tr>
<td>job_name</td>
<td>The name of the job to create to run the chain</td>
</tr>
<tr>
<td>start_steps</td>
<td>Comma-separated list of the steps to start when the chain starts running</td>
</tr>
<tr>
<td>step_state_list</td>
<td>List of chain steps with an initial state (SUCCEEDED or FAILED) to set for each.</td>
</tr>
</tbody>
</table>

Set the attributes of sys.schedulers$ step_type as follows:

- **step_name**: The name of the step
- **step_type**: 'SUCCEEDED' or 'FAILED'
- **error_number**: where error_number is a positive or negative integer.

Usage Notes

Running a chain requires CREATE JOB if the job is being created in the user's schema, or CREATE ANY JOB otherwise. In addition, the owner of the job being created needs execute privileges on the chain (as the owner of the chain, or as a user with the EXECUTE privilege on the chain or the EXECUTE ANY PROGRAM system privilege).

Examples

The following example illustrates how to start a chain in the middle by providing the initial state of some chain steps.

```sql
declare
    initial_step_states sys.schedulers$ step_type_list;
begin
    initial_step_states := sys.schedulers$ step_type_list(
        sys.schedulers$ step_type('step1', 'SUCCEEDED'),
        sys.schedulers$ step_type('step2', 'FAILED 27486'),
        sys.schedulers$ step_type('step3', 'SUCCEEDED'),
        sys.schedulers$ step_type('step5', 'SUCCEEDED'));
    dbms_scheduler.run_chain('my_chain', initial_step_states);
end;
/
```

143.6.65 RUN_JOB Procedure

This procedure runs a job immediately.

If a job is enabled, the Scheduler runs it automatically. It is not necessary to call RUN_JOB to run a job according to its schedule. Use RUN_JOB to run a job outside of its normal schedule.

Syntax

```sql
DBMS_SCHEDULER.RUN_JOB (  
    job_name                IN VARCHAR2,
    use_current_session     IN BOOLEAN DEFAULT TRUE);
```
Parameters

Table 143-79  RUN_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>A job name or a comma-separate list of entries, where each is the name of an existing job, optionally preceded by a schema name and dot separator. If you specify a multiple-destination job, the job runs on all destinations. In this case, the use_current_session argument must be FALSE.</td>
</tr>
</tbody>
</table>
| use_current_session| This specifies whether or not the job run should occur in the same session that the procedure was invoked from. The job always runs as the job owner, in the job owner's schema, unless it has credential specified, then the job runs using the user named in the credential. When use_current_session is set to TRUE:  
  • You can test a job and see any possible errors on the command line.  
  • state, run_count, last_start_date, last_run_duration, and failure_count of *_scheduler_jobs are not updated.  
  • RUN_JOB can be run in parallel with a regularly scheduled job run.  
When use_current_session is set to FALSE:  
  • You need to check the job log to find error information.  
  • All relevant fields in *_scheduler_jobs are updated.  
  • RUN_JOB fails if a regularly scheduled job is running.  
For jobs that have a specified destination or destination group, or point to chains or programs with the detached attribute set to TRUE, use_current_session must be FALSE. |

Usage Notes

Jobs do not have to be enabled. If a job is disabled, the following validity checks are performed before running it:

• The job points to a valid job class.
• The job owner has EXECUTE privileges on the job class.
• If a program or chain is referenced, the program/chain exists.
• If a program or chain is referenced, the job owner has privileges to execute the program/chain.
• All argument values have been set (or have defaults).
• The job owner has the CREATE EXTERNAL JOB privilege if this is an external job.

A TRUE value for use_current_session is not permitted for the following types of jobs:

• Jobs that specify a destination or destination group in the destination_name attribute
• Jobs that point to chains (chain jobs)
• Jobs that make use of detached programs (detached jobs).

above bug fix 1261887 6.12.11
When use_current_session is TRUE, the call to RUN_JOB blocks until the job completes. Any errors that occur during the execution of the job are returned as errors to the RUN_JOB procedure.

Using RUN_JOB with use_current_session=TRUE does not update the job state and the job will not appear in *_SCHEDULER_RUNNING_JOBS views.

above bug fix 19185117 9.15.14

When use_current_session is FALSE, RUN_JOB returns immediately and the job is picked up by the job coordinator and passed on to a job slave for execution. The Scheduler views and logs must be queried for the outcome of the job.

Multiple user sessions can use RUN_JOB in their sessions simultaneously when use_current_session is set to TRUE.

RUN_JOB requires that you own the job or have ALTER privileges on that job. You can also run a job if you have the CREATE ANY JOB privilege.

Example

The following is an example of using RUN_JOB.

BEGIN
    DBMS_SCHEDULER.RUN_JOB(
        JOB_NAME => 'EODJOB, DSS.ETLJOB',
        USE_CURRENT_SESSION => FALSE);
END;

143.6.66 SET_AGENT_REGISTRATION_PASS Procedure

This procedure sets the agent registration password for a database.

A Scheduler agent must register with the database before the database can submit jobs to the agent. The agent must provide this password when registering.

Syntax

DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS (  
    registration_password   IN VARCHAR2,  
    expiration_date         IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,  
    max_uses                IN NUMBER DEFAULT NULL);  

Parameters

Table 143-80  SET_AGENT_REGISTRATION_PASS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registration_pass</td>
<td>This is the password that remote agents must specify in order to successfully register with the database. If this is NULL, then no agents will be able to register with the database.</td>
</tr>
<tr>
<td>expiration_date</td>
<td>If this is set to a non-NULL value, then the registration_password is not valid after this date. After this date, no agents can register with the database. This cannot be set to a date in the past.</td>
</tr>
</tbody>
</table>
Table 143-80 (Cont.) SET_AGENT_REGISTRATION_PASS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_uses</td>
<td>This is the maximum number of successful registrations that can be performed with this password. After the number of successful registrations has been performed with this password, then no agents can register with the database. This cannot be set to 0 or a negative value. If this is set to NULL, then there will be no limit on the number of successful registrations.</td>
</tr>
</tbody>
</table>

Usage Notes

To prevent abuse, this password can be set to expire after a given date or a maximum number of successful registrations. This procedure will overwrite any password already set. This requires the MANAGE_SCHEDULER system privilege.

By default, max_uses is set to NULL, which means that there is no limit to the number of successful registrations.

Oracle recommends that an agent registration password be reset after every agent registration or every known set of agent registrations. Furthermore, Oracle recommends that this password be set to NULL if no new agents are being registered.

143.6.67 SET_ATTRIBUTE Procedure

This procedure modifies an attribute of a Scheduler object. It is overloaded to accept values of various types.

To set an attribute to NULL, use the SET_ATTRIBUTE_NULL procedure. The attributes that can be set depend on the object being altered. All object attributes can be changed, except the object name.

Syntax

```plaintext
DBMS_SCHEDULER.SET_ATTRIBUTE (name           IN VARCHAR2,
                                attribute      IN VARCHAR2,
                                value          IN {BOOLEAN|DATE|TIMESTAMP|
                                                  |TIMESTAMP WITH TIME ZONE|TIMESTAMP WITH LOCAL TIME ZONE|
                                                  |INTERVAL DAY TO SECOND});

DBMS_SCHEDULER.SET_ATTRIBUTE (name           IN VARCHAR2,
                                attribute      IN VARCHAR2,
                                value          IN VARCHAR2,
                                value2         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 143-81 SET_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the object.</td>
</tr>
<tr>
<td>attribute</td>
<td>See Table 143-83 through Table 143-93.</td>
</tr>
</tbody>
</table>
Table 143-81  (Cont.) SET_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The new value being set for the attribute. This cannot be NULL. To set an attribute value to NULL, use the SET_ATTRIBUTE_NULL procedure.</td>
</tr>
<tr>
<td>value2</td>
<td>The value2 argument is for an optional second value. Most attributes have only one value associated with them, but some can have two.</td>
</tr>
</tbody>
</table>

Table 143-82 is a directory of Scheduler object types and tables of attributes for the object types.

These object types can be viewed with Scheduler Data Dictionary Views, listed in Oracle Database Administrator's Guide.

Table 143-82  Attribute Tables for Scheduler Object Types

<table>
<thead>
<tr>
<th>Scheduler Object Type</th>
<th>Table of Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>Table 143-83</td>
</tr>
<tr>
<td>Program</td>
<td>Table 143-85</td>
</tr>
<tr>
<td>Schedule</td>
<td>Table 143-86</td>
</tr>
<tr>
<td>File Watcher</td>
<td>Table 143-87</td>
</tr>
<tr>
<td>Job Class</td>
<td>Table 143-88</td>
</tr>
<tr>
<td>Window</td>
<td>Table 143-89</td>
</tr>
<tr>
<td>Chain</td>
<td>Table 143-90</td>
</tr>
<tr>
<td>Database Destination</td>
<td>Table 143-91</td>
</tr>
<tr>
<td>External Destination</td>
<td>Table 143-92</td>
</tr>
<tr>
<td>Group</td>
<td>Table 143-93</td>
</tr>
<tr>
<td>Credential</td>
<td>Table 143-94</td>
</tr>
<tr>
<td>Resource</td>
<td>Table 143-95</td>
</tr>
</tbody>
</table>

Usage Notes

If an object is altered and it was in the enabled state, the Scheduler first disables it, then makes the change and reenables it. If any errors are encountered during the enable process, the object is not reenabled and an error is generated.

If an object is altered and it was in the disabled state, it remains disabled after it is altered.

To run SET_ATTRIBUTE for a window, a group of type WINDOW, or job class, you must have the MANAGE_SCHEDULER privilege. Otherwise, you must be the owner of the object being altered or have ALTER privileges on that object or have the CREATE ANY JOB privilege.

Job

If there is a running instance of the job when the SET_ATTRIBUTE call is made, it is not affected by the call. The change is only affects future runs of the job.
If any of the schedule attributes of a job are altered while the job is running, the time of the next job run is scheduled using the new schedule attributes. Schedule attributes of a job include schedule_name, start_date, end_date, and repeat_interval.

If any of the program attributes of a job are altered while the job is running, the new program attributes take effect the next time the job runs. Program attributes of a job include program_name, job_action, job_type, and number_of_arguments.

If any job argument values are altered while the job is running, the new values take effect the next time the job runs.

Granting the ALTER privilege on a job lets a user alter all attributes of that job except its program attributes (program_name, job_type, job_action, program_action, and number_of_arguments) and does not allow a user to use a PL/SQL expression to specify the schedule for a job.

Oracle recommends that you not alter a job that was automatically created for you by the database. Jobs that were created by the database have the column SYSTEM set to TRUE in job views.

Program

If any currently running jobs use the program that was altered, they continue to run with the program definition prior to the alter. The job runs with the new program definition the next time the job executes.

Schedule

If a schedule is altered, the change does not affect running jobs and open windows that use this schedule. The change only goes into effect the next time the jobs runs or the window opens.

File Watcher

If a file watcher is altered, any currently running event-based jobs started by the file arrival event are not affected. On the local system, the new file watcher attributes take effect the next time that the file watcher checks for the arrival of the file (every ten minutes by default). On remote systems, there may be an additional delay before the new file watcher attributes take effect.

Job Class

With the exception of the default job class, all job classes can be altered. To alter a job class, you must have the MANAGE SCHEDULER privilege.

When a job class is altered, running jobs that belong to the class are not affected. The change only takes effect for jobs that have not started running yet.

Window

When a window is altered, it does not affect an active window. The changes only take effect the next time the window opens.

If there is no current resource plan, when a window with a designated resource plan opens, the Resource Manager activates with that plan.

Job Attribute Values

Table 143-83 lists attribute values for jobs.
See the CREATE_JOB procedure and the CREATE_JOBS procedure for more complete descriptions of the attributes in this table.

### Table 143-83  Job Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow_runs_in_restricted_mode</td>
<td>If TRUE, the job is permitted to run when the database is in restricted mode, provided that the job owner is permitted to log in during this mode. FALSE by default.</td>
</tr>
<tr>
<td>auto_drop</td>
<td>This attribute, if TRUE, causes a job to be automatically dropped after it completes or is automatically disabled. A job is considered completed if:</td>
</tr>
<tr>
<td></td>
<td>· Its end date (or the end date of the schedule) has passed.</td>
</tr>
<tr>
<td></td>
<td>· It has run max_runs number of times. max_runs must be set with SET_ATTRIBUTE.</td>
</tr>
<tr>
<td></td>
<td>· It is not a repeating job and has run once. A job is automatically disabled when it has failed max_failures times. max_failures is also set with SET_ATTRIBUTE.</td>
</tr>
<tr>
<td></td>
<td>If this attribute is set to FALSE, the jobs are not dropped and their metadata is kept until the job is explicitly dropped with the DROP_JOB procedure.</td>
</tr>
<tr>
<td></td>
<td>By default, jobs are created with auto_drop set to TRUE.</td>
</tr>
<tr>
<td>comments</td>
<td>An optional comment.</td>
</tr>
<tr>
<td>connect_credential_name</td>
<td>feature 25230</td>
</tr>
<tr>
<td></td>
<td>This attribute may be set to point to a database credential. For a SQL*Plus or backup script job, the credential connects to the database before running the script. For other job types, it is ignored. The job owner must have execute privileges on the credential, otherwise the job fails.</td>
</tr>
<tr>
<td></td>
<td>Using a connect_credential_name is recommended since it allows the password to be stored securely in a credential in the database rather than in plain view in the job, program action, or script.</td>
</tr>
<tr>
<td>credential_name</td>
<td>This attribute specifies the name of the credential object (credential) to use for a remote database job, a remote external job, a local external job, or an event-based job that processes a file arrival event. For local external jobs only, if this attribute is NULL (the default), then a preferred (default) credential is selected.</td>
</tr>
<tr>
<td></td>
<td>See Oracle Database Administrator’s Guide for information about preferred credentials for local external jobs.</td>
</tr>
<tr>
<td>database_role</td>
<td>This attribute applies when the database participates in an Oracle Data Guard environment. If this attribute is set to 'PRIMARY', the job runs only when the database is in the role of the primary database. If set to 'LOGICAL STANDBY', the job runs only when the database is in the role of a logical standby. The default is 'PRIMARY' when the database is the primary database, and 'LOGICAL STANDBY' when the database is a logical standby.</td>
</tr>
<tr>
<td></td>
<td>Note: If you want a job to run for all database roles on a particular host, you must create two copies of the job on that host: one with a database_role of 'PRIMARY', and the other with a database_role of 'LOGICAL STANDBY'.</td>
</tr>
</tbody>
</table>
### Table 143-83  (Cont.) Job Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>*** Deprecated in Oracle Database 11g Release 2. Use destination_name instead. This attribute specifies a host on which to run a remote external job. It must be set to the host name or IP address of the destination host. It can optionally be followed by a port number, in the following format: hostname:port This attribute is set to NULL by default.</td>
</tr>
<tr>
<td>destination_name</td>
<td>The database destination or external destination for the job. Use for remote database jobs and remote external jobs only. For jobs running on the local database or for local external jobs (executables), must be NULL. See Table 143-28 for details about this attribute.</td>
</tr>
<tr>
<td>end_date</td>
<td>Specifies the date and time after which the job expires and is no longer run. After the end_date, if is TRUE, the job is dropped. If auto_drop is FALSE, the job is disabled and the STATE of the job is set to COMPLETED. If no value for end_date is specified, the job repeats forever unless max_runs or max_failures is set, in which case the job stops when either value is reached. The value for end_date must be after the value for start_date. If end_date is less than start_date, then an error will be generated. If end_date is the same as start_date, then the job will not execute and no error will be generated.</td>
</tr>
<tr>
<td>event_spec</td>
<td>This attribute takes two values: the value argument specifies the event condition and the value2 argument specifies the queue specification. For more details, see the descriptions for the event_condition and queue_spec arguments in the &quot;CREATE_JOB Procedure&quot;.</td>
</tr>
<tr>
<td>follow_default_timezone</td>
<td>If TRUE and if the job start_date is null, then when the default_timezone scheduler attribute is changed, the Scheduler recomputes the next run date and time for this job so that it is in accordance with the new time zone. For example, if the job was set to run at 02:00 in the previous time zone, it will run at 02:00 in the new time zone. If the job start_date is not null, then the time zone for the run date and time for the job is always specified by the time zone of the start_date. If FALSE, the next start date and time for the job is not recomputed when the default_timezone scheduler attribute is changed. In this case, if the old time zone is three hours earlier than the new time zone, then a job scheduled to run at 02:00 in the old time zone runs at 05:00 in the new time zone. Summer and winter transitions do not change the default time zone name.</td>
</tr>
<tr>
<td>instance_id</td>
<td>Valid only in an Oracle Real Application Clusters environment. Indicates the instance on which the job is to be run.</td>
</tr>
</tbody>
</table>
## Table 143-83  (Cont.) Job Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance_stickiness</td>
<td>This attribute should only be used for a database running in an Oracle Real Application Clusters (Oracle RAC) environment. By default, it is set to TRUE. If you set instance_stickiness to TRUE, jobs start running on the instance with the lightest load and the Scheduler thereafter attempts to run on the instance that it last ran on. If that instance is either down or so overloaded that it does not start new jobs for a significant period of time, another instance runs the job. If the interval between runs is large, instance_stickiness is ignored and the job is handled as if it were a non-sticky job. If instance_stickiness is set to FALSE, each instance of the job runs on the first instance available. For environments other than Oracle RAC, this attribute is not useful because there is only one instance.</td>
</tr>
<tr>
<td>job_action</td>
<td>The action that the job performs, depending on the job_type attribute. For example, if job_type is 'STORED_PROCEDURE', job_action contains the name of the stored procedure.</td>
</tr>
<tr>
<td>job_class</td>
<td>The class this job is associated with.</td>
</tr>
<tr>
<td>job_priority</td>
<td>This attribute specifies the priority of this job relative to other jobs in the same class as this job. If multiple jobs within a class are scheduled to be executed at the same time, the job priority determines the order in which jobs from that class are picked up for execution by the job coordinator. It can be a value from 1 through 5, with 1 being the first to be picked up for job execution. If no job priority is specified when creating a job, the default priority of 3 is assigned to it.</td>
</tr>
<tr>
<td>job_type</td>
<td>The type of this job. Valid values are: 'PLSQL_BLOCK', 'STORED_PROCEDURE', 'EXECUTABLE', 'CHAIN', 'EXTERNAL_SCRIPT', 'SQL_SCRIPT', and 'BACKUP_SCRIPT'. If this is set, program_name must be NULL.</td>
</tr>
<tr>
<td>job_weight</td>
<td>*** Deprecated in Oracle Database 11gR2. Do not change the value of this attribute from the default, which is 1. Weight of the job for parallel execution.</td>
</tr>
<tr>
<td>logging_level</td>
<td>This attribute specifies how much information is logged. The possible options are:</td>
</tr>
<tr>
<td></td>
<td>DBMS_SCHEDULER.LOGGING_OFF (The default) No logging is performed for this job. However, the logging level of the job class takes precedence and job logging may occur.</td>
</tr>
<tr>
<td></td>
<td>DBMS_SCHEDULER.LOGGING_FAILED_RUNS The Scheduler logs only jobs that failed, with the reason for failure. If the job class has a higher logging level, then the higher logging level takes precedence.</td>
</tr>
<tr>
<td></td>
<td>DBMS_SCHEDULER.LOGGING_RUNS The Scheduler writes detailed information to the job log for all runs of each job in this class. If the job class has a higher logging level, then the higher logging level takes precedence.</td>
</tr>
<tr>
<td></td>
<td>DBMS_SCHEDULER.LOGGING_FULL In addition to recording every run of a job, the Scheduler records all operations performed on the job, including create, enable, disable, alter (with SET_ATTRIBUTE), stop, and so on.</td>
</tr>
</tbody>
</table>
### Table 143-83  (Cont.) Job Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_failures</td>
<td>This attribute specifies the number of times a job can fail on consecutive scheduled runs before it is automatically disabled. Once a job is disabled, it is no longer executed and its STATE is set to BROKEN in the *_SCHEDULER_JOB views. max_failures can be an integer between 1 to 1,000,000. By default, it is set to NULL, which indicates that new instances of the job are started regardless of how many previous instances have failed.</td>
</tr>
<tr>
<td>max_run_duration</td>
<td>This attribute specifies the maximum amount of time that the job should be allowed to run. Its datatype is INTERVAL DAY TO SECOND. If this attribute is set to a non-zero and non-NULL value, and job duration exceeds this value, the Scheduler raises an event of type JOB OVER MAX_DUR. It is then up to your event handler to decide whether or not to allow the job to continue.</td>
</tr>
<tr>
<td>max_runs</td>
<td>This attribute specifies the maximum number of consecutive scheduled runs of the job. Once max_runs is reached, the job is disabled and its state is changed to COMPLETED. max_runs can be an integer between 1 and 1,000,000. By default, it is set to NULL, which means that it repeats forever or until end_date or max_failures is reached.</td>
</tr>
<tr>
<td>number_of_arguments</td>
<td>The number of arguments if the program is inlined. If this is set, program_name should be NULL.</td>
</tr>
<tr>
<td>parallel_instances</td>
<td>This is a boolean attribute that can be set only for event-based jobs. If FALSE (the default), then if an event is raised and the event-based job that processes that event is already running, the new event is ignored. If TRUE, then an instance of the job is started for every instance of the event, and each job instance is a lightweight job so multiple instances of the same event-based job can run in parallel. Each lightweight job takes its attributes (such as action, maximum run duration, and so on) from the definition of the event-based job (its parent job). After the lightweight job completes, it is dropped. There is no explicit limit to the number of lightweight jobs that can run simultaneously to process multiple instances of the event. However, limitations may be imposed by available system resources. The lightweight jobs are not visible in any of the *_SCHEDULER_JOBS views. However, they are visible in the *_SCHEDULER_RUNNING_JOBS views. The name of each lightweight job is the same as that of the parent job, and a subname is automatically generated to distinguish each lightweight job from its parent and from its siblings.</td>
</tr>
<tr>
<td>program_name</td>
<td>The name of a program object to use with this job. If this is set, job_action, job_type and number_of_arguments should be NULL.</td>
</tr>
</tbody>
</table>
Table 143-83  (Cont.) Job Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>raise_events</td>
<td>This attribute tells the Scheduler at what stages of the job execution to raise events. It is a bit vector in which zero or more of the following bits can be set. Each bit has a package constant corresponding to it.</td>
</tr>
<tr>
<td></td>
<td>· job_started CONSTANT PLS_INTEGER := 1</td>
</tr>
<tr>
<td></td>
<td>· job_succeeded CONSTANT PLS_INTEGER := 2</td>
</tr>
<tr>
<td></td>
<td>· job_failed CONSTANT PLS_INTEGER := 4</td>
</tr>
<tr>
<td></td>
<td>· job_broken CONSTANT PLS_INTEGER := 8</td>
</tr>
<tr>
<td></td>
<td>· job_completed CONSTANT PLS_INTEGER := 16</td>
</tr>
<tr>
<td></td>
<td>· job_stopped CONSTANT PLS_INTEGER := 32</td>
</tr>
<tr>
<td></td>
<td>· job_sch_lim_reached CONSTANT PLS_INTEGER := 64</td>
</tr>
<tr>
<td></td>
<td>· job_disabled CONSTANT PLS_INTEGER := 128</td>
</tr>
<tr>
<td></td>
<td>· job_chain_stalled CONSTANT PLS_INTEGER := 256</td>
</tr>
<tr>
<td></td>
<td>· job_all_events CONSTANT PLS_INTEGER := 511</td>
</tr>
<tr>
<td></td>
<td>· job_run_completed CONSTANT PLS_INTEGER := job_succeeded + job_failed + job_stopped</td>
</tr>
<tr>
<td></td>
<td>Table 143-84 describes these event types in detail.</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>Either a PL/SQL function returning the next date and time on which to run, or calendaring syntax expression. If this is set, schedule_name should be NULL. See &quot;Calendaring Syntax&quot; for more information.</td>
</tr>
<tr>
<td>restartable</td>
<td>This attribute specifies whether or not a job can be restarted in case of failure. By default, jobs are not restartable and this attribute is set to FALSE. Setting this to TRUE means that if a job fails while running, it is restarted from the beginning point of the job.</td>
</tr>
<tr>
<td></td>
<td>In the case of a chain job, if this attribute is TRUE, the chain is restarted from the beginning after an application failure. If this attribute is FALSE, or if there has been a database failure, the chain is restarted at the last running step. The restart_on_recovery attribute of that step then determines if the step is restarted or marked as stopped. (If marked as stopped, the chain evaluates rules and continues.)</td>
</tr>
<tr>
<td></td>
<td>Note that setting this attribute to TRUE might lead to data inconsistencies in some situations, for example, if data is committed within a job.</td>
</tr>
<tr>
<td></td>
<td>Retries on errors are not counted as regular runs. The run count or failure count is not incremented until the job succeeds or has failed all its six retries.</td>
</tr>
<tr>
<td></td>
<td>The restartable attribute is used by the Scheduler to determine whether to retry the job not only on regular application errors, but after a database malfunction as well. The Scheduler retries the job a maximum of six times. The first time, it waits for one second and multiplies this wait time with a factor of 10 each time thereafter.</td>
</tr>
<tr>
<td></td>
<td>Both the run count and failure count are incremented by 1 if the job has failed all its six retries. If the job immediately succeeds, or it succeeds on one of its retries, run count is incremented by 1.</td>
</tr>
<tr>
<td></td>
<td>The Scheduler stops retrying a job when:</td>
</tr>
<tr>
<td></td>
<td>· One of the retries succeeds.</td>
</tr>
<tr>
<td></td>
<td>· All of its six retries have failed.</td>
</tr>
<tr>
<td></td>
<td>· The next retry would occur after the next regularly scheduled run of the job. The Scheduler no longer retries the job if the next scheduled retry is past the next regularly scheduled run for repeating jobs.</td>
</tr>
</tbody>
</table>
Table 143-83  (Cont.) Job Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_limit</td>
<td>In heavily loaded systems, jobs are not always started at their scheduled time. This attribute enables you to have the Scheduler not start a job at all if the delay in starting the job is larger than the interval specified. It can be a value of 1 minute to 99 days. For example, if a job was supposed to start at noon and the schedule limit is set to 60 minutes, the job will not be run if it has not started to run by 1:00 p.m. If schedule_limit is not specified, the job is executed at some later date as soon as there are resources available to run it. By default, this attribute is set to null, which indicates that the job can be run at any time after its scheduled time. A scheduled job run that is skipped because of this attribute does not count against the number of runs and failures of the job. An entry in the job log reflects the skipped run.</td>
</tr>
<tr>
<td>schedule_name</td>
<td>The name of a schedule, window, or group of type WINDOW to use as the schedule for this job. If this is set, end_date, start_date and repeat_interval should all be NULL.</td>
</tr>
<tr>
<td>start_date</td>
<td>The original date and time on which this job started or is scheduled to start. If this is set, schedule_name should be NULL.</td>
</tr>
<tr>
<td>stop_on_window_close</td>
<td>This attribute only applies if the schedule of a job is a window or a window group. Setting this attribute to TRUE implies that the job should stop once the associated window is closed. The job is stopped using the stop_job procedure with force set to FALSE. By default, stop_on_window_close is set to FALSE. Therefore, if you do not set this attribute, the job continues after the window closes. Note that, although the job is allowed to continue, its resource allocation will probably change because closing a window generally also implies a change in resource plans.</td>
</tr>
<tr>
<td>store_output</td>
<td>This is a boolean attribute. If set to TRUE, then for job runs that are logged, all job output and error messages are stored in the *_JOB_RUN_DETAILS views. If set to FALSE, then the output and messages are not stored. For new jobs, this is set, by default, to TRUE.</td>
</tr>
</tbody>
</table>

The following event types are valid values for the raise_events attribute in Table 143-83.

Table 143-84  Event Types Raised by the Scheduler

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_all_events</td>
<td>Not an event, but a constant that provides an easy way for you to enable all events</td>
</tr>
<tr>
<td>job_broken</td>
<td>The job has been disabled and has changed to the BROKEN state because it exceeded the number of failures defined by the max_failures job attribute</td>
</tr>
<tr>
<td>job_chain_stalled</td>
<td>A job running a chain is in the CHAIN_STALLED state. A running chain becomes stalled if there are no steps running or scheduled to run and the chain evaluation_interval is set to NULL. No progress is made in the chain unless there is manual intervention.</td>
</tr>
<tr>
<td>job_completed</td>
<td>The job completed because it reached its max_runs or end_date</td>
</tr>
</tbody>
</table>
Table 143-84  (Cont.) Event Types Raised by the Scheduler

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_disabled</td>
<td>The job was disabled by the Scheduler or by a call to SET_ATTRIBUTE</td>
</tr>
<tr>
<td>job_failed</td>
<td>The job failed, either due to an error or an abnormal termination.</td>
</tr>
<tr>
<td>job_over_max_dur</td>
<td>The job exceeded the maximum run duration specified by its max_run_duration attribute. (Note: you do not need to enable this event with the raise_events job attribute; it is always enabled.)</td>
</tr>
<tr>
<td>job_run_completed</td>
<td>A job run either failed, succeeded, or was stopped</td>
</tr>
<tr>
<td>job_sch_lim_reached</td>
<td>The schedule limit of the job was reached. The job was not started because the delay in starting the job exceeded the value of the schedule_limit job attribute.</td>
</tr>
<tr>
<td>job_started</td>
<td>The job started</td>
</tr>
<tr>
<td>job_stopped</td>
<td>The job was stopped by a call to STOP_JOB</td>
</tr>
<tr>
<td>job_succeeded</td>
<td>The job completed successfully</td>
</tr>
</tbody>
</table>

Program Attribute Values

Table 143-85 lists program attribute values.

Note:
See the “CREATE_PROGRAM Procedure” for more complete descriptions of the attributes in this table.

Table 143-85  Program Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments</td>
<td>An optional comment. This can describe what the program does or give usage details.</td>
</tr>
<tr>
<td>detached</td>
<td>If TRUE, the program is a detached program. See Oracle Database Administrator’s Guide for information about detached jobs and detached programs.</td>
</tr>
<tr>
<td>number_of_arguments</td>
<td>The number of arguments required by the stored procedure or other executable that the program invokes</td>
</tr>
<tr>
<td>program_action</td>
<td>The action that the program performs, indicated by the program_type attribute. For example, if program_type is 'STORED_PROCEDURE', program_action contains the name of the stored procedure.</td>
</tr>
<tr>
<td>program_type</td>
<td>The type of program. This must be one of these supported program types: 'PLSQL_BLOCK', 'STORED_PROCEDURE', and 'EXECUTABLE'.</td>
</tr>
</tbody>
</table>

Schedule Attribute Values

Table 143-86 lists schedule attribute values.
Table 143-86  Schedule Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments</td>
<td>An optional comment.</td>
</tr>
<tr>
<td>end_date</td>
<td>The cutoff date and time after which the schedule does not specify any dates.</td>
</tr>
<tr>
<td>event_spec</td>
<td>This attribute takes two values: the value argument should contain the event condition and the value2 argument should contain the queue specification. For more details, see the descriptions for the <code>event_condition</code> and <code>queue_spec</code> arguments to the &quot;CREATE_JOB Procedure&quot;.</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>An attribute specifying how often the schedule should repeat, using the calendaring syntax. See &quot;Calendaring Syntax&quot; for more information.</td>
</tr>
<tr>
<td>start_date</td>
<td>The start or reference date and time used by the calendaring syntax.</td>
</tr>
</tbody>
</table>

File Watcher Attribute Values

Table 143-87 lists file watcher attribute values.

Table 143-87  File Watcher Attribute Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>Remote host name or IP address where the file is expected to arrive. If NULL, destination is the local host.</td>
</tr>
<tr>
<td>directory_path</td>
<td>Directory in which the file is expected to arrive. The single wildcard ‘?’ at the beginning of the path denotes the Oracle home path. For example, ‘?/rdbms/log’ denotes the rdbms/log subdirectory of the Oracle home directory.</td>
</tr>
<tr>
<td>file_name</td>
<td>Name of the file being looked for. Two wildcards are permitted anywhere in the file name: ‘?’ denotes any single character, and ‘*’ denotes zero or more characters. This attribute cannot be NULL.</td>
</tr>
<tr>
<td>credential_name</td>
<td>Name of a valid credential object. The file watcher uses the credential to authenticate itself with the host operating system to access the watched-for file. The file watcher owner must have the EXECUTE privilege on the credential. Cannot be NULL.</td>
</tr>
<tr>
<td>min_file_size</td>
<td>Minimum file size in bytes before the file watcher considers the file found. Default is 0.</td>
</tr>
<tr>
<td>steady_state_duration</td>
<td>Minimum time interval that the file must remain unchanged before the file watcher considers the file found. If NULL, an internal value is used. The lower limit for this attribute is 10 seconds.</td>
</tr>
<tr>
<td>comments</td>
<td>Optional comment.</td>
</tr>
</tbody>
</table>

Job Class Attribute Values
Table 143-88 lists job class attribute values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments</td>
<td>An optional comment about the class.</td>
</tr>
<tr>
<td>log_history</td>
<td>This attribute controls the number of days that job log entries for jobs in this class are retained. It helps prevent the job log from growing indiscriminately. The range of valid values is 0 through 1000000. If set to 0, no history is kept. If NULL, retention days are set by the log_history Scheduler attribute (set with SET_SCHEDULER_ATTRIBUTE).</td>
</tr>
<tr>
<td>logging_level</td>
<td>This attribute specifies how much information is logged. The valid values are:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SCHEDULER.LOGGING_OFF</td>
</tr>
<tr>
<td></td>
<td>No logging is performed for any jobs in this class.</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SCHEDULER.LOGGING_FAILED_RUNS</td>
</tr>
<tr>
<td></td>
<td>The Scheduler logs only jobs in the class that failed, with the reason for failure.</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SCHEDULER.LOGGING_RUNS</td>
</tr>
<tr>
<td></td>
<td>The Scheduler writes detailed information to the job log for all runs of each job in this class. This is the default.</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SCHEDULER.LOGGING_FULL</td>
</tr>
<tr>
<td></td>
<td>The Scheduler records all operations performed on all jobs in this class, in addition to recording every run of a job. Every time a job is created, enabled, disabled, altered (with SET_ATTRIBUTE), stopped, and so on, an entry is recorded in the log.</td>
</tr>
<tr>
<td>resource_consumer _group</td>
<td>The resource consumer group that a class is associated with. All jobs in the class run under this resource consumer group. See Oracle Database Administrator's Guide for a description of resource consumer groups and the Database Resource Manager.</td>
</tr>
<tr>
<td>service</td>
<td>The database service that the jobs in the job class have affinity to. If both the resource_consumer_group and service attributes are set for a job class, and if the service is mapped to a resource consumer group, the resource_consumer_group attribute takes precedence.</td>
</tr>
</tbody>
</table>

Window Attribute Values

Table 143-89 lists window attribute values.

Note:

See the “CREATE_WINDOW Procedure” for more complete descriptions of the attributes in this table.
Table 143-89  Window Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments</td>
<td>An optional comment about the window.</td>
</tr>
<tr>
<td>duration</td>
<td>The duration of the window.</td>
</tr>
<tr>
<td>end_date</td>
<td>The date after which the window no longer opens. If this is set, schedule_name must be NULL.</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>An attribute specifying how often the schedule should repeat, using the calendaring syntax. PL/SQL date functions are not allowed. If this is set, schedule_name must be NULL. See &quot;Calendaring Syntax&quot; for more information.</td>
</tr>
<tr>
<td>resource_plan</td>
<td>The resource plan to be associated with a window. When the window opens, the system switches to this resource plan. When the window closes, the original resource plan is restored. If a resource plan has been made active with the force option, no resource plan switch occurs. Only one resource plan can be associated with a window. It may be NULL or the empty string (&quot;&quot;). When it is NULL, the resource plan that is in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window.</td>
</tr>
<tr>
<td>schedule_name</td>
<td>The name of a schedule to use with this window. If this is set, start_date, end_date, and repeat_interval must all be NULL.</td>
</tr>
<tr>
<td>start_date</td>
<td>The next date and time on which this window is scheduled to open. If this is set, schedule_name must be NULL.</td>
</tr>
<tr>
<td>window_priority</td>
<td>The priority of the window. Must be either 'LOW' (default) or 'HIGH'.</td>
</tr>
</tbody>
</table>

Chain Attribute Values

Table 143-90 lists chain attribute values.

Note:

See the "CREATE_CHAIN Procedure" for more complete descriptions of the attributes in this table.

Table 143-90  Chain Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments</td>
<td>An optional comment describing the purpose of the chain.</td>
</tr>
<tr>
<td>evaluation_interval</td>
<td>If not NULL, provides an additional evaluation of the chain at this interval, as well as at normal evaluation times (when the job starts, when a step completes, or when an event that is associated with an event step arrives) This attribute should only be used when chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler, because the extra interval is CPU intensive. For most chains, the normal evaluation times are sufficient.</td>
</tr>
</tbody>
</table>
Table 143-90  (Cont.) Chain Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_set_name</td>
<td>In the normal case, no rule set should be passed in. The Scheduler automatically creates a rule set and associated empty evaluation context. You then use DEFINE_CHAIN_RULE to add rules and DROP_CHAIN_RULE to remove them.</td>
</tr>
<tr>
<td></td>
<td>Advanced users can create a rule set that describes their chain dependencies and pass it in here. This allows greater flexibility in defining rules. For example, conditions can refer to external variables, and tables can be exposed through the evaluation context. If you pass in a rule set, you must ensure that it is in the format of a chain rule set. (For example, all steps must be listed as variables in the evaluation context). If no rule set is passed in, the rule set created is of the form SCHED_RULESET$(N) and the evaluation context created is of the form SCHED_EVCTX$(N)</td>
</tr>
<tr>
<td></td>
<td>See Oracle Streams Concepts and Administration for information on rules and rule sets.</td>
</tr>
</tbody>
</table>

Database Destination Attribute Values

Table 143-91 lists database destination attribute values.

Note:

See the "CREATE_DATABASE_DESTINATION Procedure" for more complete descriptions of the attributes in this table.

Table 143-91  Database Destination Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent</td>
<td>The name of the external destination (also known as agent destination) that is used to connect to the remote database. You can obtain valid external destination names from the view ALL_SCHEDULER_EXTERNAL_DESTS.</td>
</tr>
<tr>
<td>connect_info</td>
<td>The TNS connect descriptor that identifies the remote database to connect to, or the net service name (alias) in tnsnames.ora that resolves to the connect descriptor.</td>
</tr>
<tr>
<td></td>
<td>Note: This corresponds to the tns_name argument of CREATE_DATABASE_DESTINATION.</td>
</tr>
<tr>
<td>enabled</td>
<td>If TRUE, the database destination is enabled.</td>
</tr>
<tr>
<td>comments</td>
<td>An optional comment about the database destination.</td>
</tr>
</tbody>
</table>

External Destination Attribute Values

Table 143-92 lists external destination attribute values.
Note:

External destinations are created only implicitly by registering a remote Scheduler agent with the local database.

Table 143-92  External Destination Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>(GET_ATTRIBUTE only) The fully qualified host name (including domain) or IP address of the computer on which the Scheduler agent resides.</td>
</tr>
<tr>
<td>port</td>
<td>(GET_ATTRIBUTE only) The TCP port number on which the agent listens.</td>
</tr>
<tr>
<td>ip_address</td>
<td>(GET_ATTRIBUTE only) The IP address of the host on which the agent resides.</td>
</tr>
<tr>
<td>enabled</td>
<td>If TRUE, the external destination is enabled.</td>
</tr>
<tr>
<td>comments</td>
<td>An optional comment about the external destination.</td>
</tr>
</tbody>
</table>

Group Attribute Values

Table 143-93 lists group attribute values.

Note:

See the “CREATE_GROUP Procedure” for more complete descriptions of the attributes in this table.

Table 143-93  Group Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_type</td>
<td>(GET_ATTRIBUTE only) The group type (either WINDOW, DB_DEST, or EXTERNAL_DEST).</td>
</tr>
<tr>
<td>member_name</td>
<td>Comma-separated list of members. Replaces the existing list of members. To add one or more members to the existing list, use ADD_GROUP_MEMBER.</td>
</tr>
<tr>
<td></td>
<td>Note: this attribute corresponds to the member argument of CREATE_GROUP.</td>
</tr>
<tr>
<td>enabled</td>
<td>If TRUE, the group is enabled.</td>
</tr>
<tr>
<td>comments</td>
<td>An optional comment about the group.</td>
</tr>
<tr>
<td>number_of_members</td>
<td>(GET_ATTRIBUTE only) The number of members in the group.</td>
</tr>
</tbody>
</table>

Credential Attribute Values

Table 143-94 lists credential attribute values.
Note:

Credential attribute values for the `SET_ATTRIBUTE` and `GET_ATTRIBUTE` procedures are deprecated with Oracle Database Release 12c Release 1 (12.1). While these attribute values remain available in this package, for reasons of backward compatibility, Oracle recommends using the alternative enhanced functionality provided in the `DBMS_CREDENTIAL` package, specifically the attribute parameter in the `UPDATE_CREDENTIAL` Procedure.

### Table 143-94  Credential Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>The user name for logging into the host operating system or remote Oracle database. Maximum length is 64.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the user name. Maximum length is 128.</td>
</tr>
<tr>
<td>comments</td>
<td>A description of the credential. Maximum length is 240.</td>
</tr>
<tr>
<td>windows_domain</td>
<td>For a Windows remote executable target, this is the domain that the specified user belongs to. Maximum length is 64.</td>
</tr>
<tr>
<td>database_role</td>
<td>The value of the <code>database_role</code> attribute is used as the system privilege for logging into a remote database to run a remote database job. Valid values are: <code>SYSDBA</code> and <code>SYSOPER</code>.</td>
</tr>
</tbody>
</table>

### Resource Attribute Values

Table 143-95 lists resource attribute values.

### Table 143-95  Resource Attribute Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource_name</td>
<td>The name of the resource</td>
</tr>
<tr>
<td>units</td>
<td>The number of units of this resource that the job or program uses.</td>
</tr>
<tr>
<td>status</td>
<td>The status of the resource. ENFORCE_CONSTRAINTS. This is the default value, and when set, will force the scheduler to enforce resource limits. When the maximum number of units of this resource has been reached, no additional jobs using this resource will get started. ENFORCE_CONSTRAINTS. When set, the scheduler will ignore any constraints on this resource. BLOCKED_ALL_JOBS. No jobs having a constraint on this resource will be allowed to run. The resource is considered to be permanently blocking until switched to one of the other two states.</td>
</tr>
<tr>
<td>constraint_level</td>
<td>Level of the constraint: <code>JOB_LEVEL</code> or <code>PROGRAM_LEVEL</code> For incompatibilities, for <code>JOB_LEVEL</code>, the incompatibility members must be jobs; for <code>PROGRAM_LEVEL</code> the incompatibility members must be programs.</td>
</tr>
<tr>
<td>comments</td>
<td>Descriptive comment about the resource.</td>
</tr>
</tbody>
</table>
143.6.68 SET_ATTRIBUTE_NULL Procedure

This procedure sets an attribute of an object to NULL.

The attributes that can be set depend on the object being altered. If the object is enabled, it is disabled before being altered and reenabled afterward. If the object cannot be reenabled, an error is generated and the object is left in a disabled state.

Syntax

```
DBMS_SCHEDULER.SET_ATTRIBUTE_NULL (  
    name              IN VARCHAR2,  
    attribute         IN VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the object</td>
</tr>
<tr>
<td>attribute</td>
<td>The attribute being changed</td>
</tr>
</tbody>
</table>

Usage Notes

To run SET_ATTRIBUTE_NULL for a window, group of type WINDOW, or job class, you must have the MANAGE_SCHEDULER privilege. Otherwise, you must be the owner of the object being altered or have ALTER privileges on that object or have the CREATE_ANY_JOB privilege.

143.6.69 SET_JOB_ANYDATA_VALUE Procedure

This procedure sets the value for an argument of the associated program for a job, encapsulated in an AnyData object.

It overrides any default value set for the program argument. NULL is a valid assignment for a program argument.

The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the DEFINE_ANYDATA_ARGUMENT Procedure

Scheduler does no type checking of the argument at any time.

SET_JOB_ANYDATA_VALUE is overloaded.

Syntax

Sets a program argument by its position.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (  
    job_name                 IN VARCHAR2,  
```
Sets a program argument by its name.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (
    job_name                IN VARCHAR2,
    argument_name           IN VARCHAR2,
    argument_value          IN SYS.ANYDATA);
```

**Parameters**

**Table 143-97  SET_JOB_ANYDATA_VALUE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of the job to be altered</td>
</tr>
<tr>
<td>argument_name</td>
<td>The name of the program argument being set</td>
</tr>
<tr>
<td>argument_position</td>
<td>The position of the program argument being set</td>
</tr>
<tr>
<td>argument_value</td>
<td>The new value to be assigned to the program argument, encapsulated in an AnyData object</td>
</tr>
</tbody>
</table>

**Usage Notes**

`SET_JOB_ANYDATA_VALUE` requires that you own the job or have `ALTER` privileges on that job. You can also set a job argument value if you have the `CREATE ANY JOB` privilege.

`SET_JOB_ANYDATA_VALUE` does not apply to lightweight jobs because lightweight jobs cannot take AnyData arguments.

**See Also:**

- "SET_JOB_ARGUMENT_VALUE Procedure"
- "DEFINE_ANYDATA_ARGUMENT Procedure"

### 143.6.70 SET_JOB_ARGUMENT_VALUE Procedure

This procedure sets the value of an argument for a job.

It overrides any default value set for the corresponding program or stored procedure argument. The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the `DEFINE_PROGRAM_ARGUMENT Procedure` or the `DEFINE_METADATA_ARGUMENT Procedure`

Scheduler does no type checking of the argument at any time.

`SET_JOB_ARGUMENT_VALUE` is overloaded.
Syntax

Sets an argument value by position:

```sql
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
    job_name                IN VARCHAR2,
    argument_position       IN PLS_INTEGER,
    argument_value          IN VARCHAR2);
```

Sets an argument value by name:

```sql
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
    job_name                IN VARCHAR2,
    argument_name           IN VARCHAR2,
    argument_value          IN VARCHAR2);
```

Parameters

Table 143-98  SET_JOB_ARGUMENT_VALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>The name of the job to be altered</td>
</tr>
<tr>
<td>argument_name</td>
<td>The name of the program argument being set</td>
</tr>
<tr>
<td>argument_position</td>
<td>The position of the program argument being set</td>
</tr>
<tr>
<td>argument_value</td>
<td>The new value to be set for the program argument. To set a non-VARCHAR value, use the SET_JOB_ANYDATA_VALUE procedure.</td>
</tr>
</tbody>
</table>

Usage Notes

SET_JOB_ARGUMENT_VALUE requires that you be the owner of the job or have ALTER privileges on that job. You can also set a job argument value if you have the CREATE ANY JOB privilege.

SET_JOB_ARGUMENT_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

SET_JOB_ARGUMENT_VALUE can be used to set arguments of lightweight jobs but only if the argument is of type VARCHAR2.

See Also:

- "SET_JOB_ANYDATA_VALUE Procedure"
- "DEFINE_PROGRAM_ARGUMENT Procedure"
143.6.71 SET_JOB_ATTRIBUTES Procedure

This procedure changes an attribute of a job.

Syntax

```
DBMS_SCHEDULER.SET_JOB_ATTRIBUTES (            
    jobattr_array     IN JOBATTR_ARRAY,           
    commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobattr_array</td>
<td>The array of job attribute changes.</td>
</tr>
<tr>
<td>commit_semantics</td>
<td>The commit semantics. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>• STOP_ON_FIRST_ERROR returns on the first error and commits previous</td>
</tr>
<tr>
<td></td>
<td>successful attribute changes to disk. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• TRANSACTIONAL returns on the first error and rolls back everything that</td>
</tr>
<tr>
<td></td>
<td>happened before that error.</td>
</tr>
<tr>
<td></td>
<td>• ABSORB_ERRORS tries to absorb any errors and complete the rest of the</td>
</tr>
<tr>
<td></td>
<td>job attribute changes on the list. It commits all the successful changes.</td>
</tr>
<tr>
<td></td>
<td>If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for</td>
</tr>
<tr>
<td></td>
<td>details.</td>
</tr>
</tbody>
</table>

Usage Notes

Calling SET_ATTRIBUTE on an enabled job disables the job, changes the attribute value, and reenables the job. SET_JOB_ATTRIBUTES changes the attribute values in the context of a single transaction.

143.6.72 SET_RESOURCECONSTRAINT Procedure

This procedure allows users to specify the resources used by jobs.

Syntax

```
DBMS_SCHEDULER.SET_RESOURCECONSTRAINT (          
    object_name       IN VARCHAR2,              
    resource_name     IN VARCHAR2,              
    units             IN NUMBER DEFAULT 1);      
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of a program or a job, or a comma separated list of these objects.</td>
</tr>
<tr>
<td>resource_name</td>
<td>The name of the resource.</td>
</tr>
</tbody>
</table>
### Table 143-100  (Cont.) SET_RESOURCE_CONSTRAINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>units</td>
<td>The number of units of this resource that the job or program uses.</td>
</tr>
</tbody>
</table>

**Usages Notes**

`object_name` can be the name or comma-separated list of names of either programs or jobs. This creates a constraint on the named resource for these programs or jobs.

`units` specifies the number of units of the resource that the program or job can use. If `units` is set to 0, then the program or job does not use this resource anymore, and the resulting constraint is deleted. Setting `units` to 0 on a resource with no previous constraint results in an error.

When multiple constraints are defined on the same resource, the object types must match. When one or more existing constraints for a resource are based on jobs and a new constraint is added for the same resource that is based on a program (or vice versa) an error will be raised.

### 143.6.73 SET_SCHEDULER_ATTRIBUTE Procedure

This procedure sets the value of a Scheduler attribute. This takes effect immediately but the resulting changes may not be seen immediately, depending on the attribute affected.

**Table 143-101** provides short attribute descriptions for the `SET_SCHEDULER_ATTRIBUTE` procedure. For complete descriptions, see section “Setting Scheduler Preferences” in *Oracle Database Administrator's Guide*.

**Syntax**

```sql
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE (attribute IN VARCHAR2, value IN VARCHAR2);
```
### Parameters

#### Table 143-101  SET_SCHEDULER_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>The name of the Scheduler attribute. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- 'default_timezone': Repeating jobs and windows that use the calendaring</td>
</tr>
<tr>
<td></td>
<td>syntax retrieve the time zone from this attribute when start_date is</td>
</tr>
<tr>
<td></td>
<td>not specified. See &quot;Calendaring Syntax&quot; for more information.</td>
</tr>
<tr>
<td></td>
<td>- 'email_server': The SMTP server address that the Scheduler uses to send</td>
</tr>
<tr>
<td></td>
<td>e-mail notifications for job state events. E-mail notifications cannot</td>
</tr>
<tr>
<td></td>
<td>be sent if this attribute is NULL.</td>
</tr>
<tr>
<td></td>
<td>- 'email_sender': The default e-mail address of the sender of job state e-</td>
</tr>
<tr>
<td></td>
<td>mail notifications.</td>
</tr>
<tr>
<td></td>
<td>- 'email_server_credential': The schema and name of an existing credential</td>
</tr>
<tr>
<td></td>
<td>object that SYS has execute object privileges on. Default is NULL.</td>
</tr>
<tr>
<td></td>
<td>The username and password stored in this credential are used to</td>
</tr>
<tr>
<td></td>
<td>authenticate with the e-mail server when sending e-mail notifications.</td>
</tr>
<tr>
<td></td>
<td>- 'email_server_encryption': This attribute indicates whether or not</td>
</tr>
<tr>
<td></td>
<td>encryption is enabled for this email server connection, and if so, at</td>
</tr>
<tr>
<td></td>
<td>what point encryption starts, and with which protocol. Values are:</td>
</tr>
<tr>
<td></td>
<td>- NONE: the default, indicating no encryption used</td>
</tr>
<tr>
<td></td>
<td>- SSL_TLS: indicating that either SSL or TLS are used, from the</td>
</tr>
<tr>
<td></td>
<td>beginning of the connection</td>
</tr>
<tr>
<td></td>
<td>- STARTTLS: indicating that the connection starts unencrypted, but the</td>
</tr>
<tr>
<td></td>
<td>command STARTTLS is sent to the e-mail server and starts encryption</td>
</tr>
<tr>
<td></td>
<td>- 'event_expiry_time': The time, in seconds, before a job state event</td>
</tr>
<tr>
<td></td>
<td>generated by the Scheduler expires from the Scheduler event queue. If</td>
</tr>
<tr>
<td></td>
<td>NULL, job state events expire after 24 hours.</td>
</tr>
<tr>
<td></td>
<td>- 'log_history': The number of days that log entries for both the job log</td>
</tr>
<tr>
<td></td>
<td>and the window log are retained. Default is 30 and the range of valid</td>
</tr>
<tr>
<td></td>
<td>values is 0 through 100000.</td>
</tr>
<tr>
<td></td>
<td>- 'max_job_slave_processes': This Scheduler attribute is not used.</td>
</tr>
<tr>
<td>value</td>
<td>The new value of the attribute</td>
</tr>
</tbody>
</table>

### Usage Notes

To run SET_SCHEDULER_ATTRIBUTE, you must have the MANAGE SCHEDULER privilege.

**See Also:**

*Oracle Database Administrator's Guide* for more detailed descriptions of Scheduler attributes.
143.6.74 STOP_JOB Procedure

This procedure stops currently running jobs or all jobs in a job class.

After stopping the job, the state of a one-time job is set to STOPPED, whereas the state of a repeating job is set to SCHEDULED or COMPLETED, depending on whether the next run of the job is scheduled.

If a job pointing to a chain is stopped, all running steps of the running chain are stopped.

If a job has multiple destinations, the database attempts to stop the job at all destinations.

For external jobs, STOP_JOB stops only the external process that was directly started by the job action. It does not stop child processes of external jobs.

For in-memory full jobs in an Oracle Real Application Clusters environment, STOP_JOB uses the instance_id attribute of the job definition to determine in which instance (or all of them if the attribute is left null) to stop the in-memory full job. (In-memory full jobs are kept cached in memory, and as such are limited to the instance currently caching them. Because of this, the same job_name can in some conditions be used for different jobs on different instances.)

Syntax

```
DBMS_SCHEDULER.STOP_JOB (
    job_name         IN VARCHAR2,
    force            IN BOOLEAN DEFAULT FALSE,
    commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 143-102  STOP_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>Name of a job to stop. Can be a comma-separate list of jobs, where each entry can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>- Job name: the name of an existing job, optionally preceded by a schema name and dot separator.</td>
</tr>
<tr>
<td></td>
<td>- Job destination ID: a number, obtained from the JOB_DEST_ID column of the *_SCHEDULER_JOB_DESTS views, that represents the unique combination of a job, a credential, and a destination.</td>
</tr>
<tr>
<td></td>
<td>- Job class: the name of a job class. Must be preceded by the SYS schema name and a dot separator.</td>
</tr>
</tbody>
</table>

If you specify a job class, all jobs that belong to that job class are stopped. If you specify a job that was created with a destination group as its destination_name attribute, all job instances on all destinations are stopped.
### Table 143-102 (Cont.) STOP_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>If <code>force</code> is set to FALSE, the Scheduler tries to gracefully stop the job using an interrupt mechanism. This method gives control back to the slave process, which can update the status of the job in the job queue to stopped. If this fails, an error is returned. If <code>force</code> is set to TRUE, the Scheduler immediately terminates the job slave. Oracle recommends that STOP_JOB with force set to TRUE be used only after a STOP_JOB with force set to FALSE has failed. Use of the <code>force</code> option requires the MANAGE_SCHEDULER system privilege.</td>
</tr>
<tr>
<td>commit_semantics</td>
<td>The commit semantics. The following two types are supported:</td>
</tr>
<tr>
<td></td>
<td>• STOP_ON_FIRST_ERROR: The procedure returns on the first error and commits previous successful stop operations to disk. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• ABSORB_ERRORS: The procedure tries to absorb any errors, stops the rest of the jobs, and commits all the successful stop operations. This type is available only if no job classes are specified in the <code>job_name</code> list. If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for details.</td>
</tr>
</tbody>
</table>

### Usage Notes

**STOP_JOB** without the `force` option requires that you be the owner of the job or have ALTER privileges on that job. You can also stop a job if you have the CREATE ANY JOB or MANAGE SCHEDULER privilege.

**STOP_JOB** with the `force` option requires that you have the MANAGE SCHEDULER privilege.

### Example

The following is an example of using **STOP_JOB**.

```sql
BEGIN
    DBMS_SCHEDULER.STOP_JOB('DSS.ETLJOB, 984, 1223, SYS.ETL_JOBCLASS');
END;
```
The `DBMS_SERVER_ALERT` package enables you to configure the Oracle Database server to issue an alert when a threshold for a specified server metric has been violated. You can configure both warning and critical thresholds for a large number of predefined metrics.

If a warning threshold is reached, the server generates a severity level 5 alert. If a critical threshold is reached, the server generates a severity level 1 alert.

The chapter contains the following topics:

- Security Model
- Object Types
- Relational Operators
- Supported Metrics
- Summary of `DBMS_SERVER_ALERT` Subprograms

### 144.1 DBMS_SERVER_ALERT Security Model

The user needs `DBA` or `IMP_FULL_DATABASE` roles to use the `DBMS_SERVER_ALERT` package.

### 144.2 DBMS_SERVER_ALERT Object Types

You qualify the metric by an individual object for the listed object types.

**Table 144-1  Object Types Defined as Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_TYPE_SYSTEM</td>
<td>Metrics collected on the system level for each instance.</td>
</tr>
<tr>
<td>OBJECT_TYPE_FILE</td>
<td>Metrics collected on the file level. These are used for <code>AVG_FILE_READ_TIME</code> and <code>AVG_FILE_WRITE_TIME</code> metrics.</td>
</tr>
<tr>
<td>OBJECT_TYPE_SERVICE</td>
<td>Metrics collected on the service level. Currently <code>ELAPSED_TIME_PER_CALL</code> and <code>CPU_TIME_PER_CALL</code> are collected.</td>
</tr>
<tr>
<td>OBJECT_TYPE_TABLESPACE</td>
<td>Metrics collected on the tablespace level. Note: Dictionary managed tablespaces are not supported.</td>
</tr>
<tr>
<td>OBJECT_TYPE_EVENT_CLASS</td>
<td>Metrics collected on wait event class level. Currently supported metrics are <code>AVG_USERS_WAITING</code> and <code>DB_TIME_WAITING</code>.</td>
</tr>
<tr>
<td>OBJECT_TYPE_SESSION</td>
<td>Metrics collected on the session level. Currently only <code>BLOADED_USERS</code> is collected. The threshold can only be set at the instance level, which means that no object name should be specified when setting the threshold for this type of metric.</td>
</tr>
<tr>
<td>OBJECT_TYPE_WRCLIENT</td>
<td>Refers to a group of metrics (WCR ...) used during replay to monitor the replay clients’ performance</td>
</tr>
</tbody>
</table>
144.3 DBMS_SERVER_ALERT Relational Operators

You can specify a relational comparison operator to determine whether or not a given metric's value violates the threshold setting. The server supports the following operators.

Table 144-2  Relational Operators Defined as Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATOR_CONTAINS</td>
<td>A metric value matching an entry in a list of threshold values is considered a violation.</td>
</tr>
<tr>
<td>OPERATOR_DO_NOT_CHECK</td>
<td>The metric value is not compared to the threshold value, and no alerts are generated. Use this operator to disable alerts for a metric.</td>
</tr>
<tr>
<td>OPERATOR_EQ</td>
<td>A metric value equal to the threshold value is considered a violation.</td>
</tr>
<tr>
<td>OPERATOR_GE</td>
<td>A metric value greater than or equal to the threshold value is considered a violation.</td>
</tr>
<tr>
<td>OPERATOR_GT</td>
<td>A metric value greater than the threshold value is considered a violation.</td>
</tr>
<tr>
<td>OPERATOR_LE</td>
<td>A metric value less than or equal to the threshold value is considered a violation.</td>
</tr>
<tr>
<td>OPERATOR_LT</td>
<td>A metric value less than the threshold value is considered a violation.</td>
</tr>
<tr>
<td>OPERATOR_NE</td>
<td>A metric value not equal to the threshold value is considered a violation.</td>
</tr>
</tbody>
</table>

144.4 DBMS_SERVER_ALERT Supported Metrics

These metrics are supported. All internal metric names are supplied as package constants.

Table 144-3  List of Supported Metrics

<table>
<thead>
<tr>
<th>Metric Name (Internal)</th>
<th>Metric Name (External)</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG_FILE_READ_TIME</td>
<td>Average File Read Time</td>
<td>Microseconds</td>
</tr>
<tr>
<td>AVG_FILE_WRITE_TIME</td>
<td>Average File Write Time</td>
<td>Microseconds</td>
</tr>
<tr>
<td>AVG_USERS_WAITING</td>
<td>Average Number of Users Waiting on a Class of Wait Events</td>
<td>Count of sessions</td>
</tr>
<tr>
<td>BLOCKED_USERS</td>
<td>Number of Users blocked by some Session</td>
<td>Number of Users</td>
</tr>
<tr>
<td>BRANCH_NODE_SPLITS_SEC</td>
<td>Branch Node Splits (for each second)</td>
<td>Splits for each Second</td>
</tr>
<tr>
<td>BRANCH_NODE_SPLITS_TXN</td>
<td>Branch Node Splits (for each transaction)</td>
<td>Splits for each Transaction</td>
</tr>
<tr>
<td>BUFFER_CACHE_HIT</td>
<td>Buffer Cache Hit (%)</td>
<td>% of cache accesses</td>
</tr>
<tr>
<td>Metric Name (Internal)</td>
<td>Metric Name (External)</td>
<td>Units</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>CONSISTENT_CHANGES_SEC</td>
<td>Consistent Changes (for each second)</td>
<td>Changes for each Second</td>
</tr>
<tr>
<td>CONSISTENT_CHANGES_TXN</td>
<td>Consistent Changes (for each transaction)</td>
<td>Changes for each Transaction</td>
</tr>
<tr>
<td>CONSISTENT_GETS_SEC</td>
<td>Consistent Gets (for each second)</td>
<td>Gets for each Second</td>
</tr>
<tr>
<td>CONSISTENT_GETS_TXN</td>
<td>Consistent Gets (for each transaction)</td>
<td>Gets for each Transaction</td>
</tr>
<tr>
<td>CR_BLOCKS_CREATED_SEC</td>
<td>CR Blocks Created (for each second)</td>
<td>Blocks for each Second</td>
</tr>
<tr>
<td>CR_BLOCKS_CREATED_TXN</td>
<td>CR Blocks Created (for each transaction)</td>
<td>Blocks for each Transaction</td>
</tr>
<tr>
<td>CR_RECORDS_APPLIED_SEC</td>
<td>CR Undo Records Applied (for each second)</td>
<td>Records for each Second</td>
</tr>
<tr>
<td>CR_RECORDS_APPLIED_TXN</td>
<td>CR Undo Records Applied (for each transaction)</td>
<td>Records for each Transaction</td>
</tr>
<tr>
<td>CURSOR_CACHE_HIT</td>
<td>Cursor Cache Hit (%)</td>
<td>% of soft parses</td>
</tr>
<tr>
<td>DATABASE_WAIT_TIME</td>
<td>Database Wait Time (%)</td>
<td>% of all database time</td>
</tr>
<tr>
<td>DATABASE_CPU_TIME</td>
<td>Database CPU Time (%)</td>
<td>% of all database time</td>
</tr>
<tr>
<td>DB_BLKGETS_SEC</td>
<td>DB Block Gets (for each second)</td>
<td>Gets for each Second</td>
</tr>
<tr>
<td>DB_BLKGETS_TXN</td>
<td>DB Block Gets (for each transaction)</td>
<td>Gets for each Transaction</td>
</tr>
<tr>
<td>DB_TIME_WAITING</td>
<td>Percent of Database Time Spent Waiting on a Class of Wait Events</td>
<td>% of Database Time</td>
</tr>
<tr>
<td>DBWR_CKPT_SEC</td>
<td>DBWR Checkpoints (for each second)</td>
<td>Checkpoints for each Second</td>
</tr>
<tr>
<td>DISK_SORT_SEC</td>
<td>Sorts to Disk (for each second)</td>
<td>Sorts for each Second</td>
</tr>
<tr>
<td>DISK_SORT_TXN</td>
<td>Sorts to Disk (for each transaction)</td>
<td>Sorts for each Transaction</td>
</tr>
<tr>
<td>ELAPSED_TIME_PER_CALL</td>
<td>Elapsed time for each user call for each service</td>
<td>Microseconds for each call</td>
</tr>
<tr>
<td>ENQUEUE_DEADLOCKS_SEC</td>
<td>Enqueue Deadlocks (for each second)</td>
<td>Deadlocks for each Second</td>
</tr>
<tr>
<td>ENQUEUE_DEADLOCKS_TXN</td>
<td>Enqueue Deadlocks (for each transaction)</td>
<td>Deadlocks for each Transaction</td>
</tr>
<tr>
<td>ENQUEUE_REQUESTS_SEC</td>
<td>Enqueue Requests (for each second)</td>
<td>Requests for each Second</td>
</tr>
<tr>
<td>ENQUEUE_REQUESTS_TXN</td>
<td>Enqueue Requests (for each transaction)</td>
<td>Requests for each Transaction</td>
</tr>
<tr>
<td>ENQUEUE_TIMEOUTS_SEC</td>
<td>Enqueue Timeouts (for each second)</td>
<td>Timeouts for each Second</td>
</tr>
<tr>
<td>ENQUEUE_TIMEOUTS_TXN</td>
<td>Enqueue Timeouts (for each transaction)</td>
<td>Timeouts for each Transaction</td>
</tr>
<tr>
<td>ENQUEUE_WAITS_SEC</td>
<td>Enqueue Waits (for each second)</td>
<td>Waits for each Second</td>
</tr>
<tr>
<td>ENQUEUE_WAITS_TXN</td>
<td>Enqueue Waits (for each transaction)</td>
<td>Waits for each Transaction</td>
</tr>
<tr>
<td>EXECUTE_WITHOUT_PARSE</td>
<td>Executes Performed Without Parsing</td>
<td>% of all executes</td>
</tr>
<tr>
<td>FULL_INDEX_SCANS_SEC</td>
<td>Fast Full Index Scans (for each second)</td>
<td>Scans for each Second</td>
</tr>
<tr>
<td>Metric Name (Internal)</td>
<td>Metric Name (External)</td>
<td>Units</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>FULL_INDEX_SCANS_TXN</td>
<td>Fast Full Index Scans (for each transaction)</td>
<td>Scans for each Transaction</td>
</tr>
<tr>
<td>GC_AVG_CR_GET_TIME</td>
<td>Global Cache CR Request</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>GC_AVG_CUR_GET_TIME</td>
<td>Global Cache Current Request</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>GC_BLOCKS_CORRUPT</td>
<td>Global Cache Blocks Corrupt</td>
<td>Blocks</td>
</tr>
<tr>
<td>GC_BLOCKS_LOST</td>
<td>Global Cache Blocks Lost</td>
<td>Blocks</td>
</tr>
<tr>
<td>HARD_PARSES_SEC</td>
<td>Hard Parsers (for each second)</td>
<td>Pares for each Second</td>
</tr>
<tr>
<td>HARD_PARSES_TXN</td>
<td>Hard Parsers (for each transaction)</td>
<td>Pares for each Transaction</td>
</tr>
<tr>
<td>LEAF_NODE_SPLITS_SEC</td>
<td>Leaf Node Splits (for each second)</td>
<td>Splits for each Second</td>
</tr>
<tr>
<td>LEAF_NODE_SPLITS_TXN</td>
<td>Leaf Node Splits (for each transaction)</td>
<td>Splits for each Transaction</td>
</tr>
<tr>
<td>LIBRARY_CACHE_HIT</td>
<td>Library Cache Hit (%)</td>
<td>% of cache accesses</td>
</tr>
<tr>
<td>LIBRARY_CACHE_MISS</td>
<td>Library Cache Miss (%)</td>
<td>% of cache accesses</td>
</tr>
<tr>
<td>LOGONS_CURRENT</td>
<td>Current Number of Logons</td>
<td>Number of Logons</td>
</tr>
<tr>
<td>LOGONS_SEC</td>
<td>Cumulative Logons (for each second)</td>
<td>Logons for each Second</td>
</tr>
<tr>
<td>LOGONS_TXN</td>
<td>Cumulative Logons (for each transaction)</td>
<td>Logons for each Transaction</td>
</tr>
<tr>
<td>LONG_TABLE_SCANS_SEC</td>
<td>Scans on Long Tables (for each second)</td>
<td>Scans for each Second</td>
</tr>
<tr>
<td>LONG_TABLE_SCANS_TXN</td>
<td>Scans on Long Tables (for each transaction)</td>
<td>Scans for each Transaction</td>
</tr>
<tr>
<td>OPEN_CURSORS_SEC</td>
<td>Cumulative Open Cursors (for each second)</td>
<td>Cursors for each Second</td>
</tr>
<tr>
<td>MEMORY_SORTS_PCT</td>
<td>Sorts in Memory (%)</td>
<td>% of sorts</td>
</tr>
<tr>
<td>NETWORK_BYTES_SEC</td>
<td>Network Bytes, for each second</td>
<td>Bytes for each Second</td>
</tr>
<tr>
<td>OPEN_CURSORS_CURRENT</td>
<td>Current Number of Cursors</td>
<td>Number of Cursors</td>
</tr>
<tr>
<td>OPEN_CURSORS_TXN</td>
<td>Cumulative Open Cursors (for each transaction)</td>
<td>Cursors for each Transaction</td>
</tr>
<tr>
<td>OS_SCHED_CPU_WAIT_TIME</td>
<td>Operating System Scheduler CPU Wait (by time)</td>
<td>Microseconds</td>
</tr>
<tr>
<td>PARSE_FAILURES_SEC</td>
<td>Parse Failures (for each second)</td>
<td>Pares for each Second</td>
</tr>
<tr>
<td>PARSE_FAILURES_TXN</td>
<td>Parse Failures (for each transaction)</td>
<td>Pares for each Transaction</td>
</tr>
<tr>
<td>PGA_CACHE_HIT</td>
<td>PGA Cache Hit (%)</td>
<td>% bytes processed in PGA</td>
</tr>
<tr>
<td>PHYS_DESIGN_WAIT_SCT</td>
<td>Physical Design Wait (by session count)</td>
<td>Count of sessions</td>
</tr>
<tr>
<td>PHYSICAL_READS_SEC</td>
<td>Physical Reads (for each second)</td>
<td>Reads for each Second</td>
</tr>
<tr>
<td>PHYSICAL_READS_TXN</td>
<td>Physical Reads (for each transaction)</td>
<td>Reads for each Transaction</td>
</tr>
<tr>
<td>PHYSICAL_WRITES_SEC</td>
<td>Physical Writes (for each second)</td>
<td>Writes for each Second</td>
</tr>
<tr>
<td>PHYSICAL_WRITES_TXN</td>
<td>Physical Writes (for each transaction)</td>
<td>Writes for each Transaction</td>
</tr>
<tr>
<td>Metric Name (Internal)</td>
<td>Metric Name (External)</td>
<td>Units</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>PHYSICAL_READS_DIR_SEC</td>
<td>Direct Physical Reads (for each second)</td>
<td>Reads for each Second</td>
</tr>
<tr>
<td>PHYSICAL_READS_DIR_TXN</td>
<td>Direct Physical Reads (for each transaction)</td>
<td>Reads for each Transaction</td>
</tr>
<tr>
<td>PHYSICAL_WRITES_DIR_SEC</td>
<td>Direct Physical Writes (for each second)</td>
<td>Writes for each Second</td>
</tr>
<tr>
<td>PHYSICAL_WRITES_DIR_TXN</td>
<td>Direct Physical Writes (for each transaction)</td>
<td>Writes for each Transaction</td>
</tr>
<tr>
<td>PHYSICAL_READS_LOB_SEC</td>
<td>Direct LOB Physical Reads (for each second)</td>
<td>Reads for each Second</td>
</tr>
<tr>
<td>PHYSICAL_READS_LOB_TXN</td>
<td>Direct LOB Physical Reads (for each transaction)</td>
<td>Reads for each Transaction</td>
</tr>
<tr>
<td>PHYSICAL_WRITES_LOB_SEC</td>
<td>Direct LOB Physical Writes (for each second)</td>
<td>Writes for each Second</td>
</tr>
<tr>
<td>PHYSICAL_WRITES_LOB_TXN</td>
<td>Direct LOB Physical Writes (for each transaction)</td>
<td>Writes for each Transaction</td>
</tr>
<tr>
<td>PROCESS_LIMIT_PCT</td>
<td>Process Limit Usage (%)</td>
<td>% of maximum value</td>
</tr>
<tr>
<td>PX_DOWNGRADED_SEC</td>
<td>Downgraded Parallel Operations (for each second)</td>
<td>Operations for each Second</td>
</tr>
<tr>
<td>PX_DOWNGRADED_25_SEC</td>
<td>Downgraded to 25% and more (for each second)</td>
<td>Operations for each Second</td>
</tr>
<tr>
<td>PX_DOWNGRADED_50_SEC</td>
<td>Downgraded to 50% and more (for each second)</td>
<td>Operations for each Second</td>
</tr>
<tr>
<td>PX_DOWNGRADED_75_SEC</td>
<td>Downgraded to 75% and more (for each second)</td>
<td>Operations for each Second</td>
</tr>
<tr>
<td>PX_DOWNGRADED_SER_SEC</td>
<td>Downgraded to serial (for each second)</td>
<td>Operations for each Second</td>
</tr>
<tr>
<td>RB_RECORDS_APPLIED_SEC</td>
<td>Rollback Undo Records Applied (for each second)</td>
<td>Records for each Second</td>
</tr>
<tr>
<td>RB_RECORDS_APPLIED_TXN</td>
<td>Rollback Undo Records Applied (for each transaction)</td>
<td>Records for each Transaction</td>
</tr>
<tr>
<td>REDO_ALLOCATION_HIT</td>
<td>Redo Log Allocation Hit</td>
<td>% of redo allocations</td>
</tr>
<tr>
<td>REDO_GENERATED_SEC</td>
<td>Redo Generated (for each second)</td>
<td>Redo Bytes for each Second</td>
</tr>
<tr>
<td>REDO_GENERATED_TXN</td>
<td>Redo Generated (for each transaction)</td>
<td>Redo Bytes for each Transaction</td>
</tr>
<tr>
<td>REDO_WRITES_SEC</td>
<td>Redo Writes (for each second)</td>
<td>Writes for each Second</td>
</tr>
<tr>
<td>REDO_WRITES_TXN</td>
<td>Redo Writes (for each transaction)</td>
<td>Writes for each Transaction</td>
</tr>
<tr>
<td>RECURSIVE_CALLS_SEC</td>
<td>Recursive Calls (for each second)</td>
<td>Calls for each Second</td>
</tr>
<tr>
<td>RECURSIVE_CALLS_TXN</td>
<td>Recursive Calls (for each transaction)</td>
<td>Calls for each Transaction</td>
</tr>
<tr>
<td>RESPONSE_TXN</td>
<td>Response (for each transaction)</td>
<td>Seconds for each Transaction</td>
</tr>
<tr>
<td>ROWS_PER_SORT</td>
<td>Rows Processed for each Sort</td>
<td>Rows for each Sort</td>
</tr>
</tbody>
</table>
### Table 144-3  (Cont.) List of Supported Metrics

<table>
<thead>
<tr>
<th>Metric Name (Internal)</th>
<th>Metric Name (External)</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESS_LOGICAL_READS_SEC</td>
<td>Session Logical Reads (for each second)</td>
<td>Reads for each Second</td>
</tr>
<tr>
<td>SESS_LOGICAL_READS_TXN</td>
<td>Session Logical Reads (for each transaction)</td>
<td>Reads for each Transaction</td>
</tr>
<tr>
<td>SESSION_CPU_SEC</td>
<td>Database CPU (for each second)</td>
<td>Microseconds for each Second</td>
</tr>
<tr>
<td>SESSION_CPU_TXN</td>
<td>Database CPU (for each transaction)</td>
<td>Microseconds for each Transaction</td>
</tr>
<tr>
<td>SESSION_LIMIT_PCT</td>
<td>Session Limit Usage (%)</td>
<td>% of maximum value</td>
</tr>
<tr>
<td>SHARED_POOL_FREE_PCT</td>
<td>Shared Pool Free(%)</td>
<td>% of shared pool</td>
</tr>
<tr>
<td>SOFT_PARSE_PCT</td>
<td>Soft Parse (%)</td>
<td>% of all parses</td>
</tr>
<tr>
<td>SQL_SRV_RESPONSE_TIME</td>
<td>Service Response (for each execution)</td>
<td>Seconds</td>
</tr>
<tr>
<td>TABLESPACE_PCT_FULL</td>
<td>Tablespace space usage</td>
<td>% full</td>
</tr>
<tr>
<td>TABLESPACE_BYT_FREE</td>
<td>Tablespace bytes space usage</td>
<td>Kilobytes free</td>
</tr>
<tr>
<td>TOTAL_TABLE_SCANS_SEC</td>
<td>Total Table Scans (for each second)</td>
<td>Scans for each Second</td>
</tr>
<tr>
<td>TOTAL_TABLE_SCANS_TXN</td>
<td>Total Table Scans (for each transaction)</td>
<td>Scans for each Transaction</td>
</tr>
<tr>
<td>TOTAL_INDEX_SCANS_SEC</td>
<td>Total Index Scans (for each second)</td>
<td>Scans for each Second</td>
</tr>
<tr>
<td>TOTAL_INDEX_SCANS_TXN</td>
<td>Total Index Scans (for each transaction)</td>
<td>Scans for each Transaction</td>
</tr>
<tr>
<td>TOTAL_PARSES_SEC</td>
<td>Total Parses (for each second)</td>
<td>Parses for each Second</td>
</tr>
<tr>
<td>TOTAL_PARSES_TXN</td>
<td>Total Parses (for each transaction)</td>
<td>Parses for each Transaction</td>
</tr>
<tr>
<td>USER_COMMITS_SEC</td>
<td>User Commits (for each second)</td>
<td>Commits for each Second</td>
</tr>
<tr>
<td>USER_COMMITS_TXN</td>
<td>User Commits (for each transaction)</td>
<td>Commits for each Transaction</td>
</tr>
<tr>
<td>USER_ROLLBACKS_SEC</td>
<td>User Rollbacks (for each second)</td>
<td>Rollbacks for each Second</td>
</tr>
<tr>
<td>USER_ROLLBACKS_TXN</td>
<td>User Rollbacks (for each transaction)</td>
<td>Rollbacks for each Transaction</td>
</tr>
<tr>
<td>USER_CALLS_SEC</td>
<td>User Calls (for each second)</td>
<td>Calls for each Second</td>
</tr>
<tr>
<td>USER_CALLS_TXN</td>
<td>User Calls (for each transaction)</td>
<td>Calls for each Transaction</td>
</tr>
<tr>
<td>USER_CALLS_PCT</td>
<td>User Calls (%)</td>
<td>% of all calls</td>
</tr>
<tr>
<td>USER_LIMIT_PCT</td>
<td>User Limit Usage (%)</td>
<td>% of maximum value</td>
</tr>
<tr>
<td>WCR_AVG_IO_LAT</td>
<td>Average IO response time (for a WRC client)</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>WCR_PCPU</td>
<td>Percentage of replay threads on CPU (for a WRC client)</td>
<td>% of total replay threads</td>
</tr>
<tr>
<td>WCRPIO</td>
<td>Percentage of replay threads doing IOs (for a WRC client)</td>
<td>% of total replay threads</td>
</tr>
</tbody>
</table>

### 144.5 Summary of DBMS_SERVER_ALERT Subprograms

This table lists the DBMS_SERVER_ALERT subprograms and briefly describes them.
### Table 144-4 DBMS_SERVER_ALERT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPAND_MESSAGE Function</td>
<td>Expands alert messages</td>
</tr>
<tr>
<td>GET_THRESHOLD Procedure</td>
<td>Gets the current threshold settings for a specified metric</td>
</tr>
<tr>
<td>SET_THRESHOLD Procedure</td>
<td>Sets the warning and critical thresholds for a specified metric</td>
</tr>
</tbody>
</table>

#### 144.5.1 EXPAND_MESSAGE Function

This function expands alert messages.

**Syntax**

```sql
DBMS_SERVER_ALERT.EXPAND_MESSAGE(
    user_language            IN   VARCHAR2,
    message_id               IN   NUMBER,
    argument_1               IN   VARCHAR2,
    argument_2               IN   VARCHAR2,
    argument_3               IN   VARCHAR2,
    argument_4               IN   VARCHAR2,
    argument_5               IN   VARCHAR2)
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_language</td>
<td>The language of the current session.</td>
</tr>
<tr>
<td>message_id</td>
<td>Id of the alert message</td>
</tr>
<tr>
<td>argument_1</td>
<td>The first argument in the alert message.</td>
</tr>
<tr>
<td>argument_2</td>
<td>The second argument in the alert message.</td>
</tr>
<tr>
<td>argument_3</td>
<td>The third argument in the alert message.</td>
</tr>
<tr>
<td>argument_4</td>
<td>The fourth argument in the alert message.</td>
</tr>
<tr>
<td>argument_5</td>
<td>The fifth argument in the alert message.</td>
</tr>
</tbody>
</table>

#### 144.5.2 GET_THRESHOLD Procedure

This procedure gets the current threshold settings for the specified metric.

**Syntax**

```sql
DBMS_SERVER_ALERT.GET_THRESHOLD(
    metrics_id               IN   BINARY_INTEGER,
    warning_operator         OUT  BINARY_INTEGER,
    warning_value            OUT  VARCHAR2,
    critical_operator        OUT  BINARY_INTEGER,
    critical_value           OUT  VARCHAR2,
    observation_period       OUT  BINARY_INTEGER,
    consecutive_occurrences  OUT  BINARY_INTEGER,
```

Chapter 144

Summary of DBMS_SERVER_ALERT Subprograms
Parameters

Table 144-6  GET_THRESHOLD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metrics_id</td>
<td>The internal name of the metric. See &quot;Supported Metrics&quot;.</td>
</tr>
<tr>
<td>warning_operator</td>
<td>The operator for the comparing the actual value with the warning threshold.</td>
</tr>
<tr>
<td>warning_value</td>
<td>The warning threshold value.</td>
</tr>
<tr>
<td>critical_operator</td>
<td>The operator for the comparing the actual value with the critical threshold.</td>
</tr>
<tr>
<td>critical_value</td>
<td>The critical threshold value.</td>
</tr>
<tr>
<td>observation_period</td>
<td>The period at which the metric values are computed and verified</td>
</tr>
<tr>
<td>consecutive_occurrences</td>
<td>against the threshold setting.</td>
</tr>
<tr>
<td>instance_name</td>
<td>The name of the instance for which the threshold is set. This is</td>
</tr>
<tr>
<td></td>
<td>NULL for database-wide alerts.</td>
</tr>
<tr>
<td></td>
<td>In cases in which this parameter is not NULL, this should be set to one of</td>
</tr>
<tr>
<td></td>
<td>the INSTANCE_NAME values found in the GV$INSTANCE View.</td>
</tr>
<tr>
<td>object_type</td>
<td>Either OBJECT_TYPE_SYSTEM or OBJECT_TYPE_SERVICE.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object.</td>
</tr>
</tbody>
</table>

Usage Notes

Note that this subprogram does not check if the value of the instance_name parameter is meaningful or valid.

144.5.3 SET_THRESHOLD Procedure

This procedure sets the warning and critical thresholds for a specified metric.

Syntax

```sql
DBMS_SERVER_ALERT.SET_THRESHOLD(
  metrics_id               IN   BINARY_INTEGER,
  warning_operator         IN   BINARY_INTEGER,
  warning_value            IN   VARCHAR2,
  critical_operator        IN   BINARY_INTEGER,
  critical_value           IN   VARCHAR2,
  observation_period       IN   BINARY_INTEGER,
  consecutive_occurrences  IN   BINARY_INTEGER,
  instance_name            IN   VARCHAR2,
  object_type              IN   BINARY_INTEGER,
  object_name              IN   VARCHAR2);
```
Parameters

Table 144-7  SET_THRESHOLD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metrics_id</td>
<td>The internal name of the metric. See &quot;Supported Metrics&quot;.</td>
</tr>
<tr>
<td>warning_operator</td>
<td>The operator for the comparing the actual value with the warning threshold (such as OPERATOR_GE). See &quot;Relational Operators&quot;.</td>
</tr>
<tr>
<td>warning_value</td>
<td>The warning threshold value. This is NULL if no warning threshold is set. A list of values may be specified for OPERATOR_CONTAINS.</td>
</tr>
<tr>
<td>critical_operator</td>
<td>The operator for the comparing the actual value with the critical threshold. See &quot;Relational Operators&quot;.</td>
</tr>
<tr>
<td>critical_value</td>
<td>The critical threshold value. This is NULL if not set. A list of values may be specified for OPERATOR_CONTAINS.</td>
</tr>
<tr>
<td>observation_period</td>
<td>The period at which the metric values are computed and verified against the threshold setting. The valid range is 1 to 60 minutes.</td>
</tr>
<tr>
<td>consecutive_occurrences</td>
<td>The number of observation periods the metric value should violate the threshold value before the alert is issued.</td>
</tr>
<tr>
<td>instance_name</td>
<td>The name of the instance for which the threshold is set. This is NULL for database-wide alerts.</td>
</tr>
<tr>
<td>object_type</td>
<td>See &quot;Object Types&quot;.</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object. This is NULL for SYSTEM.</td>
</tr>
</tbody>
</table>

Usage Notes

Note that this subprogram does not check if the value of the instance_name parameter is meaningful or valid. Passing a name that does not identify a valid instance will result in a threshold that is not used by any by any instance although the threshold setting will be visible in the DBA_THRESHOLDS view. The exception is the lower-case string 'database_wide' which is semantically equivalent to passing NULL for the instance name, the latter being the preferred usage.
145

DBMS_SERVICE

The DBMS_SERVICE package lets you create, delete, activate, and deactivate services for a single instance.

The chapter contains the following topics:

- Overview
- Security Model
- Constants
- Operating Procedures
- Exceptions
- Summary of DBMS_SERVICE Subprograms

See Also:

Oracle Real Application Clusters Administration and Deployment Guide for administering services in Oracle Real Application Clusters.

145.1 DBMS_SERVICE Overview

DBMS_SERVICE supports the workload management of high availability, quality of service, job scheduling, and other planned operations in the RDBMS for the purposes of workload measurement, management, prioritization, and XA and distributed transaction management.

Oracle Real Application Clusters (RAC) can manage service names across instances as administered through SRVCTL. The DBMS_SERVICE package allows the creation, deletion, starting, and stopping of services in a single instance. Additionally, it provides the ability to disconnect all sessions that connect to an instance with a service name.

See Also:

For more information about Oracle Real Application Clusters, Oracle Real Application Clusters Administration and Deployment Guide.
145.2 DBMS_SERVICE Security Model

The DBMS_SERVICE package has certain security requirements.

Privileges

The client using this package must have the ALTER SYSTEM execution privilege and the V$SESSION table read privilege.

Schemas

This package must be installed under SYS schema.

Roles

The EXECUTE privilege of the package is granted to the DBA role only.

145.3 DBMS_SERVICE Constants

The DBMS_SERVICE package provides constants that can be used for specifying parameter values.

- Constants used in calling arguments are described in Table 145-1
- Constants used in connection balancing goal arguments are described in Table 145-2
- Constants used in TAF failover attribute arguments are described in Table 145-3

Table 145-1  Constants Used in Calling Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOAL_NONE</td>
<td>NUMBER</td>
<td>0</td>
<td>Disables Load Balancing Advisory</td>
</tr>
<tr>
<td>GOAL_SERVICE_TIME</td>
<td>NUMBER</td>
<td>1</td>
<td>Load Balancing Advisory is based on elapsed time for work done in the service plus available bandwidth to the service</td>
</tr>
<tr>
<td>GOAL_THROUGHPUT</td>
<td>NUMBER</td>
<td>2</td>
<td>Load Balancing Advisory is based on the rate that work is completed in the service plus available bandwidth to the service</td>
</tr>
</tbody>
</table>

Table 145-2  Constants Used in Connection Balancing Goal Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB_GOAL_SHORT</td>
<td>NUMBER</td>
<td>1</td>
<td>Connection load balancing uses Load Balancing Advisory, when Load Balancing Advisory is enabled (either goal_service_time or goal_throughput). When GOAL=None (no load balancing advisory), connection load balancing uses an abridged advice based on CPU utilization.</td>
</tr>
</tbody>
</table>
### Table 145-2 (Cont.) Constants Used in Connection Balancing Goal Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB_GOAL_LONG</td>
<td>NUMBER</td>
<td>2</td>
<td>Balances the number of connections for each instance using session count for each service. This setting is recommended for applications with long connections such as forms. This setting can be used with Load Balancing Advisory when the connection pool is sized to accommodate gravitation within the pool itself (without adding or removing connections). The latter is the most efficient design.</td>
</tr>
</tbody>
</table>

### Table 145-3 Constants Used in High Availability Attribute Arguments for FAN, Application Continuity, Transaction Guard and TAF

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAILOVER_METHOD_NON</td>
<td>VARCHAR2</td>
<td>0</td>
<td>Server side TAF is not enabled for this service</td>
</tr>
<tr>
<td>FAILOVER_METHOD_BASIC</td>
<td>VARCHAR2</td>
<td>1</td>
<td>Server side TAF method is BASIC. BASIC is the only value currently supported. This means that a new connection is established at failure time.</td>
</tr>
<tr>
<td>FAILOVER_TYPE_NONE</td>
<td>VARCHAR</td>
<td></td>
<td>Server side TAF failover type is NONE. At failure time, if the failover type is SESSION, TAF reconnects to a surviving node and re-establish a vanilla database session. Customizations (for example, ALTER SESSION) must be re-executed in a failover callback.</td>
</tr>
<tr>
<td>FAILOVER_TYPE_SESSION</td>
<td>VARCHAR</td>
<td></td>
<td>Server side TAF failover type is SESSION. At failure time, TAF reconnects to a surviving node and re-establish a vanilla database session. Customizations (for example, ALTER SESSION) must be re-executed in a failover callback.</td>
</tr>
<tr>
<td>FAILOVER_TYPE_SELECT</td>
<td>VARCHAR</td>
<td></td>
<td>Server side TAF failover type is SELECT.</td>
</tr>
<tr>
<td>FAILOVER_RETRIES</td>
<td>NUMBER</td>
<td></td>
<td>Number of connection attempts when failover occurs. Specifies the number of times for Application Continuity and TAF to attempt the reconnect and re-authenticate pair. The value must be an integer greater than 0. The default in Oracle Database 12c Release 1 (12.1) for Application Continuity is 30.</td>
</tr>
<tr>
<td>FAILOVER_RESTORE_NONE</td>
<td>VARCHAR2</td>
<td>NONE</td>
<td>The initial state is not restored before replaying for Application Continuity and TAF. This is recommended for OCI applications that use Application Continuity and build their own state in the request. For example, SQLPLUS.</td>
</tr>
<tr>
<td>FAILOVER_RESTORE_BASIC</td>
<td>VARCHAR2</td>
<td>LEVEL1</td>
<td>This is the recommended value for Java and ODP.NET applications using Application Continuity. The initial states that the user knows are restored automatically before replaying. If the user needs additional states, a callback must be registered.</td>
</tr>
</tbody>
</table>
Table 145-3  (Cont.) Constants Used in High Availability Attribute Arguments for FAN, Application Continuity, Transaction Guard and TAF

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAILOVER_DELAY</td>
<td>NUMBER</td>
<td>Number of seconds delay between each connection attempt. This is the delay that Application Continuity and TAF waits if a reconnect and re-authentication fails. The value must be an integer greater than 0. The default in Oracle Database 12c Release 1 (12.1) is 10s when using Application Continuity. Using FAILOVER_DELAY the failover can be delayed until the service is next available. This can work well in conjunction with a planned outage that may make a service temporarily unavailable (such as for several minutes).</td>
<td></td>
</tr>
<tr>
<td>STOP_OPTION_NONE</td>
<td>VARCHAR</td>
<td>Sessions are not disconnected.</td>
<td></td>
</tr>
<tr>
<td>STOP_OPTION_IMMEDIATE</td>
<td>VARCHAR</td>
<td>Sessions are disconnected immediately after the drain_timeout expires.</td>
<td></td>
</tr>
<tr>
<td>STOP_OPTION_TRANSACTIONAL</td>
<td></td>
<td>Sessions are disconnected after the transactions during the drain_timeout. The sessions disconnect immediately when drain_timeout expires.</td>
<td></td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>NUMBER</td>
<td>For Application Continuity, this parameter specifies whether the session state that is not transactional is changed by the application during request execution. A value of DYNAMIC is recommended for all applications. If you are in any doubt, or the application can be customized, you must use DYNAMIC.</td>
<td></td>
</tr>
</tbody>
</table>

Usage Notes

- If a TAF callback has been registered, then the failover retries and failover delay are ignored. If an error occurs, TAF continues to re-attempt the connect and authentication as long as the callback returns a value of OCI_FO_RETRY. Any delay must be coded into the callback logic.
- Server side TAF settings override client-side counterparts that might be configured in TNS connect descriptors. If TAF is not configured on the client side, then at a minimum, the failover type must be set to enable TAF. If the failover type is set on the server side, then the failover method defaults to BASIC. Delay and retries are optional and may be specified independently.

145.4 DBMS_SERVICE Operating Procedures

You cannot use the following procedures with Oracle Real Applications Clusterware, Oracle Restart, and Oracle Global Data Services.

- CREATE_SERVICE Procedure
- DELETE_SERVICE Procedure
- MODIFY_SERVICE Procedure
– START_SERVICE Procedure
– STOP_SERVICE Procedure

• With Oracle Database 12c release 1, you are advised to use the parameter interface in all service-related subprograms.
• If you wish to use DBMS_SERVICE on a pluggable database (PDB) in a single instance, you must connect to that PDB first.

145.5 DBMS_SERVICE Exceptions

This table lists the exceptions raised by the DBMS_SERVICE package.

<table>
<thead>
<tr>
<th>Table 145-4</th>
<th>DBMS_SERVICE Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>Error Code</td>
</tr>
<tr>
<td>NULL_SERVICE_NAME</td>
<td>44301</td>
</tr>
<tr>
<td>NULL_NETWORK_NAME</td>
<td>44302</td>
</tr>
<tr>
<td>SERVICE_EXISTS</td>
<td>44303</td>
</tr>
<tr>
<td>SERVICE_DOES_NOT_EXIST</td>
<td>44304</td>
</tr>
<tr>
<td>SERVICE_IN_USE</td>
<td>44305</td>
</tr>
<tr>
<td>SERVICE_NAME_TOO_LONG</td>
<td>44306</td>
</tr>
<tr>
<td>NETWORK_PREFIX_TOO_LONG</td>
<td>44307</td>
</tr>
<tr>
<td>NOT_INITIALIZED</td>
<td>44308</td>
</tr>
<tr>
<td>GENERAL_FAILURE</td>
<td>44309</td>
</tr>
<tr>
<td>MAX_SERVICES_EXCEEDED</td>
<td>44310</td>
</tr>
<tr>
<td>SERVICE_NOT_RUNNING</td>
<td>44311</td>
</tr>
<tr>
<td>DATABASE_CLOSED</td>
<td>44312</td>
</tr>
<tr>
<td>INVALIDINSTANCE</td>
<td>44313</td>
</tr>
<tr>
<td>NETWORK_EXISTS</td>
<td>44314</td>
</tr>
<tr>
<td>NULL_ATTRIBUTES</td>
<td>44315</td>
</tr>
<tr>
<td>INVALID_ARGUMENT</td>
<td>44316</td>
</tr>
<tr>
<td>DATABASE_READONLY</td>
<td>44317</td>
</tr>
<tr>
<td>MAX_SN_LENGTH</td>
<td>44318</td>
</tr>
<tr>
<td>ERR_AQ_SERVICE</td>
<td>44319</td>
</tr>
<tr>
<td>ERR_GLB_SERVICE</td>
<td>44320</td>
</tr>
<tr>
<td>ERR_INVALID_PDB_NAME</td>
<td>44771</td>
</tr>
</tbody>
</table>
## Table 145-4 (Cont.) DBMS_SERVICE Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_CRS_API</td>
<td>44772</td>
<td>Cluster ready services (CRS) operation failed</td>
</tr>
<tr>
<td>ERR_PDB_CLOSED</td>
<td>44773</td>
<td>Cannot perform requested service operation</td>
</tr>
<tr>
<td>ERR_PDB_INVALID</td>
<td>44774</td>
<td>Pluggable database attribute cannot be changed</td>
</tr>
<tr>
<td>ERR_PDB_NAME</td>
<td>44775</td>
<td>Pluggable database service cannot be created</td>
</tr>
<tr>
<td>ERR_PDB_EXP</td>
<td>44776</td>
<td>Pluggable database service cannot be deleted</td>
</tr>
<tr>
<td>ERR_PDB_FAIL</td>
<td>44777</td>
<td>Pluggable database service cannot be started</td>
</tr>
</tbody>
</table>

## 145.6 Summary of DBMS_SERVICE Subprograms

This table lists the DBMS_SERVICE subprograms and briefly describes them.

### Table 145-5 DBMS_SERVICE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_SERVICE Procedure</td>
<td>Creates service</td>
</tr>
<tr>
<td>DELETE_SERVICE Procedure</td>
<td>Deletes service</td>
</tr>
<tr>
<td>DISCONNECT_SESSION Procedure</td>
<td>Disconnects sessions running under this service</td>
</tr>
<tr>
<td>MODIFY_SERVICE Procedure</td>
<td>Modifies service</td>
</tr>
<tr>
<td>START_SERVICE Procedure</td>
<td>Activates service</td>
</tr>
<tr>
<td>STOP_SERVICE Procedure</td>
<td>Stops service</td>
</tr>
</tbody>
</table>

## 145.6.1 CREATE_SERVICE Procedure

This procedure creates a service name in the data dictionary. Services are also created in the data dictionary implicitly when you set the service in the `service_name` parameter or by means of the `ALTER SYSTEM SET SERVICE_NAMES` command.

### Note:

You cannot use the second version of subprogram if your services are managed by Oracle Clusterware, Oracle Restart or Oracle Global Data Services. The version with the parameter array interface applies to databases that are not managed by Oracle Clusterware, Oracle Restart or Oracle Global Data Services. New attributes are only available using the parameter interface.
Syntax

DBMS_SERVICE.CREATE_SERVICE(
    service_name               IN VARCHAR2,
    network_name               IN VARCHAR2,
    parameter_array            IN TABLE OF VARCHAR2(100));

This overload is maintained for backward compatibility:

DBMS_SERVICE.CREATE_SERVICE(
    service_name               IN VARCHAR2,
    network_name               IN VARCHAR2,
    goal                       IN NUMBER DEFAULT NULL,
    dtp                        IN BOOLEAN DEFAULT NULL,
    aq_ha_notifications        IN BOOLEAN DEFAULT NULL,
    failover_method            IN VARCHAR2 DEFAULT NULL,
    failover_type              IN VARCHAR2 DEFAULT NULL,
    failover_retries           IN NUMBER DEFAULT NULL,
    failover_delay             IN NUMBER DEFAULT NULL,
    clb_goal                   IN NUMBER DEFAULT NULL,
    edition                    IN VARCHAR2 DEFAULT NULL);

Parameters

Table 145-6 CREATE_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service, limited to 64 characters in the Data Dictionary</td>
</tr>
<tr>
<td>network_name</td>
<td>Network name of the service as used in SQLNet connect descriptors for client connections. This is limited to the NET service_names character set (see Oracle Database Net Services Reference).</td>
</tr>
<tr>
<td>parameter_array</td>
<td>Associative array with name-value pairs of the service attributes. Supported names:</td>
</tr>
<tr>
<td></td>
<td>• goal</td>
</tr>
<tr>
<td></td>
<td>• dtp</td>
</tr>
<tr>
<td></td>
<td>• aq_ha_notifications</td>
</tr>
<tr>
<td></td>
<td>• failover_method</td>
</tr>
<tr>
<td></td>
<td>• failover_type</td>
</tr>
<tr>
<td></td>
<td>• failover_retries</td>
</tr>
<tr>
<td></td>
<td>• failover_restore</td>
</tr>
<tr>
<td></td>
<td>• failover_delay</td>
</tr>
<tr>
<td></td>
<td>• clb_goal</td>
</tr>
<tr>
<td></td>
<td>• edition</td>
</tr>
<tr>
<td></td>
<td>• commit_outcome</td>
</tr>
<tr>
<td></td>
<td>• retention_timeout</td>
</tr>
<tr>
<td></td>
<td>• replay_initiation_timeout</td>
</tr>
<tr>
<td></td>
<td>• session_state_consistency</td>
</tr>
<tr>
<td></td>
<td>• sql_translation_profile</td>
</tr>
<tr>
<td></td>
<td>• drain_timeout</td>
</tr>
<tr>
<td></td>
<td>• stop_option</td>
</tr>
</tbody>
</table>
Table 145-6  (Cont.) CREATE_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal</td>
<td>Workload management goal directive for the service. Valid values:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SERVICE.GOAL_SERVICE_TIME</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SERVICE.GOAL_THROUGHPUT</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SERVICE.GOAL_NONE</td>
</tr>
<tr>
<td>dtp</td>
<td>Declares the service to be for X/Open Distributed Transaction Processing (DTP) or any distributed transaction (especially XA)</td>
</tr>
<tr>
<td>aq_ha_notifications</td>
<td>Determines whether Fast Application Notification (FAN) is enabled for OCI/OCCI/ODP. In Oracle Database12c, FAN uses Oracle Notification Services (ONS). This parameter is still used to enable FAN. FAN is recommended for all High Availability systems, and is on by default for Application Continuity</td>
</tr>
<tr>
<td>failover_method</td>
<td>Failover TYPE for the service for Application Continuity and TAF. If the failover_type is set to TRANSACTION on the service, this automatically sets COMMIT_OUTCOME to TRUE. JDBC Replay Driver uses the FAILOVER_TYPE service attribute setting of TRANSACTION for TRANSACTION failover. OCI uses the older settings of SELECT and SESSION. The server only accepts FAILOVER_METHOD = BASIC with the TRANSACTION setting.</td>
</tr>
<tr>
<td>failover_type</td>
<td>TRANSACTION for Application Continuity. SELECT or BASIC for TAF.</td>
</tr>
<tr>
<td>failover_retries</td>
<td>Number of connection retries for Application Continuity and TAF. Using the failover_retries and failover_delay parameters, the failover can be delayed until the service is next available. This parameter is for connecting. It does not control the number of failovers, which is 3 for each incident for Application Continuity.</td>
</tr>
<tr>
<td>failover_delay</td>
<td>Delay in seconds between connection retries for Application Continuity and TAF. The default is 10 seconds for Application Continuity. Do not use a 0-second delay if the service needs time to failover and register. Long delays are good for planned outages and to failover to Data Guard. Short delays work well with RAC when the service is already available.</td>
</tr>
<tr>
<td>edition</td>
<td>If this argument has a non-NULL value, this provides the initial session edition for subsequent database connections using this service that do not specify an edition. If no value is specified, this argument has no effect.</td>
</tr>
<tr>
<td></td>
<td>During service creation or modification, no validation is performed on this parameter.</td>
</tr>
<tr>
<td></td>
<td>At connection time, if the connecting user does not have USE privilege on the edition, or the edition does not exist, this raises the error ORA-38802 (edition does not exist).</td>
</tr>
<tr>
<td>drain_timeout</td>
<td>If this parameter is defined, all sessions connected to that service are drained by the client drivers and pools using Fast Connection Failover (FCF). The drain_timeout can be set on the service, so stopping and relocating drains for this time by default.</td>
</tr>
</tbody>
</table>
### Table 145-6  (Cont.) CREATE_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop_option</td>
<td>Provides options to terminate a session. The supported values are:</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTIONAL</td>
</tr>
<tr>
<td></td>
<td>- drain_timeout applies to the transactions. After the drain_timeout expire, the sessions are immediately killed.</td>
</tr>
<tr>
<td></td>
<td>- IMMEDIATE</td>
</tr>
<tr>
<td></td>
<td>The sessions are killed after drain_timeout expires.</td>
</tr>
<tr>
<td></td>
<td>- NONE</td>
</tr>
<tr>
<td></td>
<td>Sessions are not terminated.</td>
</tr>
<tr>
<td></td>
<td>However, these parameter values can be overridden using the command line.</td>
</tr>
<tr>
<td>commit_outcome</td>
<td>Determines whether transaction COMMIT outcome is accessible after the COMMIT has executed. While the database guarantees that COMMIT is durable, this ensures that the outcome of the COMMIT is durable. Applications use the feature to probe the status of the commit last executed after an outage, and is available to applications to determine an outcome. Note:</td>
</tr>
<tr>
<td></td>
<td>- Invoking the GET_LTXID_OUTCOME Procedure of the DBMS_APP.Cont package requires that the commit_outcome attribute be set.</td>
</tr>
<tr>
<td></td>
<td>- commit_outcome has no effect on active Data Guard and read-only databases.</td>
</tr>
<tr>
<td></td>
<td>- commit_outcome is only allowed on the database service and on user-defined database services</td>
</tr>
<tr>
<td>retention_timeout</td>
<td>Used in conjunction with commit_outcome, it determines the amount of time (in seconds) that the COMMIT_OUTCOME is retained. Default is 24 hours (86400). Maximum value is 30 days (2592000).</td>
</tr>
<tr>
<td>replay_initiation_timeout</td>
<td>For Application Continuity, replay_initiation_timeout is the difference between the time of original execution of first operation of a request, and the time that the replay is ready to start after a successful reconnect. Replay initiation time is measured from the time that the request was originally submitted until the time that replay has connected and is ready to replay. When replay is expected, keep this value high. Default is 900 seconds.</td>
</tr>
<tr>
<td>session_state_consistency</td>
<td>Describes how nontransactional is changed during a request (values are DYNAMIC or STATIC). This parameter is considered only if failover_type is set to TRANSACTION for Application Continuity. Examples of session state are NLS settings, optimizer preferences, event settings, PL/SQL global variables, temporary tables, advanced queues, LOBs, and result cache. If these values change after the request starts, set to DYNAMIC (default). Almost all applications should use DYNAMIC mode. If you are unsure, use DYNAMIC mode.</td>
</tr>
<tr>
<td>sql_translation_name</td>
<td>Name of SQL translation unit</td>
</tr>
<tr>
<td>clb_goal</td>
<td>Method used for Connection Load Balancing (see Table 145-2)</td>
</tr>
</tbody>
</table>

#### Examples

```sql
DBMS_SERVICE.CREATE_SERVICE('ernie.example.com','ernie.example.com');
DECLARE
```
params dbms_service.svc_parameter_array;
BEGIN
params('FAILOVER_TYPE')            :='TRANSACTION';
params('REPLAY_INITIATION_TIMEOUT'):=1800;
params('RETENTION_TIMEOUT')        :=86400;
params('FAILOVER_DELAY')           :=10;
params('FAILOVER_RETRIES')         :=30;
params('DRAIN_TIMEOUT')            :=60;
params('STOP_OPTION')              :='DBMS_SERVICE.STOP_OPTION_IMMEDIATE';
params('FAILOVER_RESTORE')         :='DBMS_SERVICE.FAILOVER_RESTORE_BASIC';
params('commit_outcome')           :='true';
params('aq_ha_notifications')      :='true';
DBMS_SERVICE.MODIFY_SERVICE('GOLD',params);
END;

145.6.2 DELETE_SERVICE Procedure

This procedure deletes a service from the data dictionary.

Note:
You cannot use this subprogram if your services are managed by Oracle Clusterware, Oracle Restart, or Oracle Global Data Services.

Syntax

DBMS_SERVICE.DELETE_SERVICE(
    service_name   IN VARCHAR2);

Parameters

Table 145-7  DELETE_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service, limited to 64 characters in the Data Dictionary</td>
</tr>
</tbody>
</table>

Examples

DBMS_SERVICE.DELETE_SERVICE('ernie.example.com');

145.6.3 DISCONNECT_SESSION Procedure

This procedure disconnects sessions with the named service at the current instance.

Syntax

DBMS_SERVICE.DISCONNECT_SESSION(
    service_name         IN VARCHAR2,
    disconnect_option    IN NUMBER DEFAULT POST_TRANSACTION;
Parameters

Table 145-8  DISCONNECT_SESSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service, limited to 64 characters in the Data Dictionary</td>
</tr>
<tr>
<td>disconnect_option</td>
<td>The options, package constants, are expressed as NUMBER:</td>
</tr>
<tr>
<td>· POST_TRANSACTION = 0</td>
<td>session disconnects after the current transaction commits or rolls back</td>
</tr>
<tr>
<td>· IMMEDIATE = 1</td>
<td>session disconnects immediately</td>
</tr>
<tr>
<td>· NOREPLAY = 2</td>
<td>session disconnects immediately and be flagged to not be replayed by application continuity, that is IMMEDIATE and NOREPLAY together</td>
</tr>
</tbody>
</table>

Note: IMMEDIATE or POST_TRANSACTION and NOREPLAY is automatically translated as 1 or 0 or 2 respectively. However, passing a string literal (quoted using either the ' or " characters, such as "IMMEDIATE" or 'POST_TRANSACTION' or 'NOREPLAY') raises an error.

Usage Notes

• This procedure can be used in the context of a single instance as well as with Oracle Real Application Clusters.

• This subprogram does not return until all corresponding sessions are disconnected. Therefore, use the DBMS_JOB package or put the SQL session in background if the caller does not want to wait for all corresponding sessions to be disconnected.

Examples

This disconnects sessions with service_name 'ernie.example.com'.

DBMS_SERVICE.DISCONNECT_SESSION('ernie.example.com');

If a service is using application continuity, and you do not want the sessions replayed but simply terminated, use the following:

EXECUTE DBMS_SERVICE.DISCONNECT_SESSION('service name', DBMS_SERVICE.NOREPLAY);

145.6.4 MODIFY_SERVICE Procedure

This procedure modifies an existing service.

Note:

You cannot use the second version of subprogram if your services are managed by Oracle Clusterware, Oracle Restart, or Oracle Global Data Services. The version with the parameter array interface applies to databases that are not managed by Oracle Clusterware, Oracle Restart or Oracle Global Data Services. New attributes are only available using the parameter interface.
Syntax

```sql
DBMS_SERVICE.MODIFY_SERVICE(
    service_name               IN VARCHAR2,
    parameter_array            IN svc_parameter_array);
```

This overload is maintained for backward compatibility:

```sql
DBMS_SERVICE.MODIFY_SERVICE(
    service_name               IN VARCHAR2,
    goal                       IN NUMBER DEFAULT NULL,
    dtp                        IN BOOLEAN DEFAULT NULL,
    aq_ha_notifications        IN BOOLEAN DEFAULT NULL,
    failover_method            IN VARCHAR2 DEFAULT NULL,
    failover_type              IN VARCHAR2 DEFAULT NULL,
    failover_retries           IN NUMBER DEFAULT NULL,
    failover_delay             IN NUMBER DEFAULT NULL,
    clb_goal                   IN NUMBER DEFAULT NULL,
    edition                    IN VARCHAR2 DEFAULT NULL,
    modify_edition             IN BOOLEAN DEFAULT FALSE;
```

Parameters

Table 145-9  MODIFY_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service, limited to 64 characters in the Data Dictionary</td>
</tr>
<tr>
<td>parameter_array</td>
<td>Associative array with name/value pairs of the service attributes. Supported names:</td>
</tr>
<tr>
<td></td>
<td>goal</td>
</tr>
<tr>
<td></td>
<td>dtp</td>
</tr>
<tr>
<td></td>
<td>aq_ha_notifications</td>
</tr>
<tr>
<td></td>
<td>failover_method</td>
</tr>
<tr>
<td></td>
<td>failover_type</td>
</tr>
<tr>
<td></td>
<td>failover_restore</td>
</tr>
<tr>
<td></td>
<td>failover_retries</td>
</tr>
<tr>
<td></td>
<td>failover_delay</td>
</tr>
<tr>
<td></td>
<td>drain_timeout</td>
</tr>
<tr>
<td></td>
<td>stop_option</td>
</tr>
<tr>
<td></td>
<td>edition</td>
</tr>
<tr>
<td></td>
<td>commit_outcome</td>
</tr>
<tr>
<td></td>
<td>retention_timeout</td>
</tr>
<tr>
<td></td>
<td>replay_initiation_timeout</td>
</tr>
<tr>
<td></td>
<td>session_state_consistency</td>
</tr>
<tr>
<td></td>
<td>sql_translation_name</td>
</tr>
<tr>
<td>goal</td>
<td>Workload management goal directive for the service. Valid values:</td>
</tr>
<tr>
<td></td>
<td>DBMS_SERVICE.GOAL_SERVICE_TIME</td>
</tr>
<tr>
<td></td>
<td>DBMS_SERVICE.GOAL_THROUGHPUT</td>
</tr>
<tr>
<td></td>
<td>DBMS_SERVICE.GOAL_NONE</td>
</tr>
<tr>
<td>dtp</td>
<td>Declares the service to be for X/Open Distributed Transaction Processing (DTP) or any distributed transaction (especially XA)</td>
</tr>
</tbody>
</table>
### Table 145-9  (Cont.) MODIFY_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aq_ha_notifications</td>
<td>Determines whether Fast Application Notification (FAN) is enabled for OCI/OCCI/ODP. In Oracle Database 12c, FAN uses Oracle Notification Services (ONS). This parameter is still used to enable FAN. FAN is recommended for all High Availability systems, and is on by default for Application Continuity.</td>
</tr>
<tr>
<td>failover_method</td>
<td>Failover TYPE for the service for Application Continuity and TAF. If the failover_type is set to TRANSACTION on the service, this automatically sets COMMIT_OUTCOME to TRUE. JDBC Replay Driver uses the FAILOVER_TYPE service attribute setting of TRANSACTION for TRANSACTION failover. OCI uses the older settings of SELECT and SESSION. The server only accepts FAILOVER_METHOD = BASIC with the TRANSACTION setting.</td>
</tr>
<tr>
<td>failover_type</td>
<td>Failover TYPE for the service for Application Continuity and TAF.</td>
</tr>
<tr>
<td>failover_restore</td>
<td>For Application Continuity, when the failover_restore parameter is set, the session states are restored before replaying for ODP.NET and Java. Use LEVEL1 for ODP.NET and Java with Application Continuity to restore the initial state. For AC OCI, use NONE for applications that are not STATIC.</td>
</tr>
<tr>
<td>failover_retries</td>
<td>Number of connection retries for Application Continuity and TAF. Using the failover_retries and failover_delay parameters, the failover can be delayed until the service is next available. This parameter is for connecting. It does not control the number of failovers, which is 3 for each incident for Application Continuity.</td>
</tr>
<tr>
<td>failover_delay</td>
<td>Delay in seconds between connection retries for Application Continuity and TAF. The default is 10 seconds for Application Continuity. Do not use a 0-second delay if the service needs time to failover and register. Long delays are good for planned outages and to failover to Data Guard. Short delays work well with Oracle RAC when the service is already available.</td>
</tr>
<tr>
<td>drain_timeout</td>
<td>When this parameter is set, all sessions connected to that service are drained by the client drivers and pools using Fast Connection Failover (FCF). The drain_timeout can be set on the service, to stop and relocate drains for this time by default.</td>
</tr>
<tr>
<td>edition</td>
<td>If this argument has a non-NULL value, this provides the initial session edition for subsequent database connections using this service that do not specify an edition. If no value is specified, this argument has no effect. During service creation or modification, no validation is performed on this parameter. At connection time, if the connecting user does not have USE privilege on the edition, or the edition does not exist, this raises the error ORA-38802 (edition does not exist).</td>
</tr>
</tbody>
</table>
Table 145-9  (Cont.) MODIFY_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit_outcome</td>
<td>Determines whether transaction COMMIT outcome is accessible after the COMMIT has executed. While the database guarantees that COMMIT is durable, this ensures that the outcome of the COMMIT is durable. Applications use the feature to probe the status of the commit last executed after an outage, and is available to applications to determine an outcome. Note:</td>
</tr>
<tr>
<td></td>
<td>* Invoking the GET_LTXID_OUTCOME Procedure of the DBMS_APP_CONT package requires that the commit_outcome attribute be set.</td>
</tr>
<tr>
<td></td>
<td>* commit_outcome has no effect on active Data Guard and read-only databases.</td>
</tr>
<tr>
<td></td>
<td>* commit_outcome is allowed only on user-defined database services</td>
</tr>
<tr>
<td>retention_timeout</td>
<td>Used in conjunction with commit_outcome, it determines the amount of time (in seconds) that the COMMIT_OUTCOME is retained. Default is 24 hours (86400). Maximum value is 30 days (2592000).</td>
</tr>
<tr>
<td>replay_initiation_timeout</td>
<td>For Application Continuity, replay_initiation_timeout is the difference between the time of original execution of first operation of a request, and the time that the replay is ready to start after a successful reconnect. Replay initiation time is measured from the time that the request was originally submitted until the time that replay has connected and is ready to replay. When replay is expected, keep this value high. Default is 900 seconds.</td>
</tr>
<tr>
<td>session_state_consistency</td>
<td>Describes how nontransactional is changed during a request (values are DYNAMIC or STATIC). This parameter is considered only if failover_type is set to TRANSACTION for Application Continuity. Examples of session state are NLS settings, optimizer preferences, event settings, PL/SQL global variables, temporary tables, advanced queues, LOBs, and result cache. If these values change after the request starts, set to DYNAMIC (default). Almost all applications should use DYNAMIC mode. If you are unsure, use DYNAMIC mode.</td>
</tr>
<tr>
<td>sql_translation_name</td>
<td>Name of SQL translation unit</td>
</tr>
<tr>
<td>modify_edition</td>
<td>If TRUE, the edition service attribute is updated to use the edition argument value. If FALSE or NULL, the edition attribute is not updated.</td>
</tr>
<tr>
<td>clb_goal</td>
<td>Method used for Connection Load Balancing (see Table 145-2)</td>
</tr>
</tbody>
</table>

Usage Notes

- If you are using Clustered Managed Services with Oracle Clusterware, or using Oracle Restart with your single instance database, you must modify services using the srvctl command rather than DBMS_SERVICE. When the service is started by Oracle Clusterware or Oracle Restart, the service is modified in the database to match the resource defined to either Oracle Clusterware or Oracle Restart. Any changes made with DBMS_SERVICE are lost unless they are also made with the corresponding srvctl command. Starting with 11.2.0.2, service attribute
modifications take effect immediately when the service is started or modified by srvctl.

• Although users can modify the edition attribute while the service is up and running, it may not be safe to do so. Users must proceed with caution because this causes new connections to be connected at the new edition, while the existing connection is not affected. This can cause mid-tier operations to connect to the wrong edition.

145.6.5 START_SERVICE Procedure

This procedure starts a service. In Oracle RAC, implementing this option acts on the instance specified.

Note:

You cannot use this subprogram if your services are managed by Oracle Clusterware, Oracle Restart or Oracle Global Data Services.

Syntax

DBMS_SERVICE.START_SERVICE(
    service_name  IN VARCHAR2,
    instance_name IN VARCHAR2);

Parameters

Table 145-10 START_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service limited to 64 characters in the Data Dictionary</td>
</tr>
<tr>
<td>instance_name</td>
<td>Name of the instance where the service must be activated (optional). NULL results in starting of the service on the local instance. In single instance, this can only be the current instance or NULL. Specify DBMS_SERVICE.ALL_INSTANCES to start the service on all configured instances.</td>
</tr>
</tbody>
</table>

Examples

DBMS_SERVICE.START_SERVICE('ernie.example.com');

145.6.6 STOP_SERVICE Procedure

This procedure stops a service.

Note:

You cannot use this subprogram if your services are managed by Oracle Clusterware, Oracle Restart or Oracle Global Data Services.
Syntax

DBMS_SERVICE.STOP_SERVICE(
    service_name   IN VARCHAR2,
    instance_name  IN VARCHAR2;
)

Parameters

Table 145-11 STOP_SERVICE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>Name of the service limited to 64 characters in the Data Dictionary</td>
</tr>
<tr>
<td>instance_name</td>
<td>Name of the instance where the service must be stopped (optional). NULL results in stopping of the service locally. In single instance, this can only be the current instance or NULL. The default in Oracle RAC and exclusive case is NULL. Specify DBMS_SERVICE.ALL_INSTANCES to stop the service on all configured instances.</td>
</tr>
<tr>
<td>drain_timeout</td>
<td>The time in seconds for the session to drain.</td>
</tr>
<tr>
<td>stop_option</td>
<td>To specify how sessions are stopped with draining. The possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td>• IMMEDIATE: sessions are aborted immediately after the time specified in drain_timeout.</td>
</tr>
<tr>
<td></td>
<td>• TRANSACTIONAL: applies for transactions. After the transaction expires, the sessions are immediately terminated.</td>
</tr>
<tr>
<td></td>
<td>• NONE: sessions are not terminated.</td>
</tr>
<tr>
<td></td>
<td>These values can be overridden on the command line using SRVCTL.</td>
</tr>
<tr>
<td>noreplay</td>
<td>Disconnected sessions are not recovered by Application Continuity.</td>
</tr>
</tbody>
</table>

Examples

DBMS_SERVICE.STOP_SERVICE('ernie.example.com');
This package provides access to SQL ALTER SESSION and SET ROLE statements, and other session information, from PL/SQL. You can use DBMS_SESSION to set preferences and security levels.

This chapter contains the following topics:

- Security Model
- Operational Notes
- Data Structures
- Summary of DBMS_SESSION Subprograms

### 146.1 DBMS_SESSION Security Model

This package runs with the privileges of the calling user, rather than the package owner SYS.

### 146.2 DBMS_SESSION Operational Notes

You should not attempt to turn close_cached_open_cursors on or off.

### 146.3 DBMS_SESSION Data Structures

The DBMS_SESSION package defines TABLE types.

**Table Types**

- INTEGER_ARRAY Table Type
- LNAME_ARRAY Table Type

### 146.3.1 DBMS_SERVICE INTEGER_ARRAY Table Type

INTEGER_ARRAY is a table type of BINARY_INTEGER.

**Syntax**

```
TYPE integer_array IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

### 146.3.2 DBMS_SERVICE LNAME_ARRAY Table Type

LNAME_ARRAY is a table type of VARCHAR2.

**Syntax**

```
TYPE lname_array IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```
### 146.4 Summary of DBMS_SESSION Subprograms

This table lists the DBMS_SESSION subprograms in alphabetical order and briefly describes them.

**Table 146-1  DBMS_SESSION Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR_ALL_CONTEXT Procedure</td>
<td>Clears all context information</td>
</tr>
<tr>
<td>CLEARCONTEXT Procedure</td>
<td>Clears the context</td>
</tr>
<tr>
<td>CLEAR_IDENTIFIER Procedure</td>
<td>Clears the identifier</td>
</tr>
<tr>
<td>CLOSE_DATABASE_LINK Procedure</td>
<td>Closes database link</td>
</tr>
<tr>
<td>FREE_UNUSED_USER_MEMORY Procedure</td>
<td>Lets you reclaim unused memory after performing operations requiring large amounts of memory</td>
</tr>
<tr>
<td>GET_PACKAGE_MEMORY_UTILIZATION Procedure</td>
<td>Describes static package memory usage</td>
</tr>
<tr>
<td>IS_ROLE_ENABLED Function</td>
<td>Determines if the named role is enabled for the session.</td>
</tr>
<tr>
<td>IS_SESSION_ALIVE Function</td>
<td>Determines if the specified session is active</td>
</tr>
<tr>
<td>LIST_CONTEXT Procedures</td>
<td>Returns a list of active namespace and context for the current session</td>
</tr>
<tr>
<td>MODIFY_PACKAGE_STATE Procedure</td>
<td>Used to perform various actions (as specified by the action_flags parameter) on the session state of all PL/SQL program units active in the session</td>
</tr>
<tr>
<td>RESET_PACKAGE Procedure</td>
<td>De-instantiates all packages in the session</td>
</tr>
<tr>
<td>SESSION_TRACE_DISABLE Procedure</td>
<td>Resets the session-level SQL trace for the session from which it was called.</td>
</tr>
<tr>
<td>SESSION_TRACE_ENABLE Procedure</td>
<td>Enables session-level SQL trace for the invoking session</td>
</tr>
<tr>
<td>SET_CONTEXT Procedure</td>
<td>Sets or resets the value of a context attribute</td>
</tr>
<tr>
<td>SET_EDITION_DEFERRED Procedure</td>
<td>Requests a switch to the specified edition</td>
</tr>
<tr>
<td>SET_IDENTIFIER Procedure</td>
<td>Sets the identifier</td>
</tr>
<tr>
<td>SET_NLS Procedure</td>
<td>Sets Globalization Support (NLS)</td>
</tr>
<tr>
<td>SET_ROLE Procedure</td>
<td>Sets role</td>
</tr>
<tr>
<td>SET_SQL_TRACE Procedure</td>
<td>Turns tracing on or off</td>
</tr>
<tr>
<td>SLEEP Procedure</td>
<td>Suspends the session for a specified period of time.</td>
</tr>
<tr>
<td>SWITCH_CURRENT_CONSUMER_GROUP Procedure</td>
<td>Facilitates changing the current resource consumer group of a user's current session</td>
</tr>
<tr>
<td>UNIQUE_SESSION_ID Function</td>
<td>Returns an identifier that is unique for all sessions currently connected to this database</td>
</tr>
</tbody>
</table>

### 146.4.1 ADD_SQL_CONNECTION_TEST Procedure

This procedure adds a new connection test that is used during draining sessions before planned maintenance begins. Use this procedure when the SQL connection...
test is not covered by standard tests. The test is enabled when added. If the optional service name qualifier is provided, the test only applies only to that service name.

Syntax

DBMS_SESSION.ADD_SQL_CONNECTION_TEST (
    connection_test    IN VARCHAR2
    service_name       IN VARCHAR2   DEFAULT NULL);

Parameters

Table 146-2  ADD_SQL_CONNECTION_TEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION_TEST</td>
<td>The SQL text used to test and drain connections.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>Optional service name qualifier.</td>
</tr>
</tbody>
</table>

Usage Notes

The ADD_SQL_CONNECTION_TEST Procedure adds a connection test for the purpose of draining sessions before planned maintenance begins. The connection test is used by the application to test connections that are marked for draining. Sessions are set for draining at stop and relocate operations for services or PDBs. When set the RDBMS closes the connection while draining so the application sees no errors during planned maintenance. You can enter as many CONNECTION TESTs as needed. They are used only during planned maintenance. The tests apply to all RAC instances.

Check online documentation for latest updates on service qualifier availability.

Added connection can be viewed by querying the view DBA_CONNECTION_TESTS.

This procedure is owned by SYS and is granted to users for execution at CDB$ROOT or PDB levels, or when not multitenant, at dictionary level.

146.4.2 CLEAR_ALL_CONTEXT Procedure

This procedure clears application context information in the specified namespace.

Syntax

DBMS_SESSION.CLEAR_ALL_CONTEXT
    namespace   VARCHAR2);

Parameters

Table 146-3  CLEAR_ALL_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>The namespace where the application context information is to be cleared. Required.</td>
</tr>
</tbody>
</table>

Usage Notes

• This procedure must be invoked directly or indirectly by the trusted package.
• Any changes in context value are reflected immediately and subsequent calls to access the value through `SYS_CONTEXT` return the most recent value.

### 146.4.3 CLEAR_CONTEXT Procedure

This procedure clears application context in the specified namespace.

**Syntax**
```
DBMS_SESSION.CLEAR_CONTEXT
namespace VARCHAR2,
client_identifier VARCHAR2,
attribute VARCHAR2);
```

**Parameters**

**Table 146-4  CLEAR_CONTEXT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>Namespace in which the application context is to be cleared. Required. For a session-local context, <code>namespace</code> must be specified. If <code>namespace</code> is defined as Session Local Context, then <code>client_identifier</code> is optional since it is only associated with a globally accessed context. For a globally accessed context, <code>namespace</code> must be specified. NULL is a valid value for <code>client_identifier</code> because a session with no identifier set can see a context that looks like the (namespace, attribute, value, username, null) set using <code>SET_CONTEXT</code>.</td>
</tr>
<tr>
<td>client_identifier</td>
<td>Applies to a global context and is optional for other types of contexts; 64-byte maximum</td>
</tr>
<tr>
<td>attribute</td>
<td>Specific attribute in the namespace to be cleared. Optional. the default is NULL. If you specify <code>attribute</code> as NULL, then (namespace, attribute, value) for that namespace are cleared from the session. If <code>attribute</code> is not specified, then all context information that has the <code>namespace</code> and <code>client_identifier</code> arguments is cleared.</td>
</tr>
</tbody>
</table>

**Usage Notes**

• This procedure must be invoked directly or indirectly by the trusted package.

• Any changes in context value are reflected immediately and subsequent calls to access the value through `SYS_CONTEXT` return the most recent value.

### 146.4.4 CLEAR_IDENTIFIER Procedure

This procedure removes the `set_client_id` in the session.

**Syntax**
```
DBMS_SESSION.CLEAR_IDENTIFIER;
```
Usage Notes

This procedure is executable by public.

146.4.5 CLOSE_DATABASE_LINK Procedure

This procedure closes an open database link. It is equivalent to the following SQL statement:

```
ALTER SESSION CLOSE DATABASE LINK <name>
```

Syntax

```
DBMS_SESSION.CLOSE_DATABASE_LINK ( 
    dblink VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblink</td>
<td>Name of the database link to close</td>
</tr>
</tbody>
</table>

146.4.6 DELETE_SQL_CONNECTION_TEST Procedure

This procedure deletes a connection test that is no longer needed for planned draining. Removing a test applies immediately to all RAC instances where the PDB is open.

Syntax

```
DBMS_SESSION.DELETE_SQL_CONNECTION_TEST ( 
    connection_test          IN VARCHAR2 
    service_name             IN VARCHAR2   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION_TEST</td>
<td>The SQL text used to test and drain connections.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>Optional service name qualifier. If the optional SERVICE_NAME qualifier is provided, only the test for that service name is deleted.</td>
</tr>
</tbody>
</table>

Usage Notes

If you are not certain if a test should be deleted, you can disable the test using DISABLE_CONNECTION_TEST Procedure. Only custom SQL tests can be deleted. Predefined tests cannot be deleted. Check for latest updates on service qualifier availability.

This procedure is owned by SYS at CDB$ROOT or PDB level, or SYS for when not multitenant.

Connection tests and their status can be checked by querying the view DBA_CONNECTION_TESTS.
146.4.7 DISABLE_CONNECTION_TEST Procedure

This procedure disables usage of a connection test during draining of sessions. Disabling a test applies immediately to all RAC instances where the PDB is open.

Syntax

```
DBMS_SESSION.DISABLE_CONNECTION_TEST (  
    connection_test_type     IN VARCHAR2,  
    connection_test          IN VARCHAR2,  
    service_name             IN VARCHAR2   DEFAULT NULL);
```

Parameters

Table 146-7  DISABLE_CONNECTION_TEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION_TEST_TYPE</td>
<td>The permitted values are:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SESSION.SQL_TEST</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SESSION.PING_TEST</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SESSION.ENDREQUEST_TEST</td>
</tr>
<tr>
<td>CONNECTION_TEST</td>
<td>The SQL text used to test and drain connections.</td>
</tr>
<tr>
<td></td>
<td>This parameter is allowed only if the value of CONNECTION_TEST_TYPE is SQL_TEST.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>Optional service name qualifier. If the optional service name qualifier is provided, only the test for that service name is enabled. A disable at service name level takes precedence over an enable at PDB level. That is the PDB can be enabled, and the service disabled.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is owned by SYS and is granted to users for execution at CDB$ROOT or PDB levels, or when not multitenant, at dictionary level.

Connection tests and their status can be checked by querying the view DBA_CONNECTION_TESTS.

146.4.8 ENABLE_CONNECTION_TEST Procedure

This procedure enables usage of a connection test for draining database sessions before planned maintenance. Enabling a test applies immediately to all RAC instances where the PDB is open.

Syntax

```
DBMS_SESSION.ENABLE_CONNECTION_TEST (  
    connection_test_type     IN VARCHAR2,  
    connection_test          IN VARCHAR2,  
    service_name             IN VARCHAR2   DEFAULT NULL);
```
Parameters

Table 146-8  ENABLE_CONNECTION_TEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| CONNECTION_TEST_TYPE | The connection type used when managing connection tests for draining before planned maintenance. See ADD, DELETE, ENABLE, DISABLE procedures for connection tests. The permitted values are:  
  · DBMS_SESSION.SQL_TEST  
  · DBMS_SESSION.PING_TEST  
  · DBMS_SESSION.ENDREQUEST_TEST  |
| CONNECTION_TEST   | The SQL text used to test and drain connections at the RDBMS before planned maintenance starts. This parameter is allowed only if the value of CONNECTION_TEST_TYPE is SQL_TEST. |
| SERVICE_NAME      | Optional service name qualifier. If the optional service name qualifier is provided, only the test for that service name is enabled. An enable at service name level overrides any higher-level disables. That is, the PDB can be disabled, and the service enabled. |

Usage Notes

- This procedure is owned by SYS and is granted to users for execution at CDB$ROOT or PDB levels, or when not multitenant, at dictionary level.
- ENABLE_CONNECTION_TEST enables a connection test for draining sessions during planned maintenance. The enable operation applies to all RAC instances where the PDB is open. It persists across database restarts.
- This procedure is owned by SYS and is granted to users for execution at CDB$ROOT or PDB levels, or when not multitenant, at dictionary level.

146.4.9 FREE_UNUSED_USER_MEMORY Procedure

This procedure reclaims unused memory after performing operations requiring large amounts of memory (more than 100K).

Examples of operations that use large amounts of memory include:

- Large sorting where entire sort_area_size is used and sort_area_size is hundreds of KB.
- Compiling large PL/SQL packages, procedures, or functions.
- Storing hundreds of KB of data within PL/SQL indexed tables.

You can monitor user memory by tracking the statistics "session UGA memory" and "session PGA memory" in the v$sesstat or v$statname fixed views. Monitoring these statistics also shows how much memory this procedure has freed.
This procedure should only be used in cases where memory is at a premium. It should be used infrequently and judiciously.

Syntax

DBMS_SESSION.FREE_UNUSED_USER_MEMORY;

Return Values

The behavior of this procedure depends upon the configuration of the server operating on behalf of the client:

- **Dedicated server**: This returns unused PGA memory and session memory to the operating system. Session memory is allocated from the PGA in this configuration.

- **Shared server**: This returns unused session memory to the shared_pool. Session memory is allocated from the shared_pool in this configuration.

Usage Notes

In order to free memory using this procedure, the memory must not be in use. After an operation allocates memory, only the same type of operation can reuse the allocated memory. For example, after memory is allocated for sort, even if the sort is complete and the memory is no longer in use, only another sort can reuse the sort-allocated memory. For both sort and compilation, after the operation is complete, the memory is no longer in use, and the user can call this procedure to free the unused memory.

An indexed table implicitly allocates memory to store values assigned to the indexed table's elements. Thus, the more elements in an indexed table, the more memory the RDBMS allocates to the indexed table. As long as there are elements within the indexed table, the memory associated with an indexed table is in use.

The scope of indexed tables determines how long their memory is in use. Indexed tables declared globally are indexed tables declared in packages or package bodies. They allocate memory from session memory. For an indexed table declared globally, the memory remains in use for the lifetime of a user's login (lifetime of a user's session), and is freed after the user disconnects from ORACLE.

Indexed tables declared locally are indexed tables declared within functions, procedures, or anonymous blocks. These indexed tables allocate memory from PGA memory. For an indexed table declared locally, the memory remains in use for as long as the user is still running the procedure, function, or anonymous block in which the indexed table is declared. After the procedure, function, or anonymous block is finished running, the memory is then available for other locally declared indexed tables to use (in other words, the memory is no longer in use).

Assigning an uninitialized, "empty" indexed table to an existing index table is a method to explicitly re-initialize the indexed table and the memory associated with the indexed table. After this operation, the memory associated with the indexed table is no longer in use, making it available to be freed by calling this procedure. This method is particularly useful on indexed tables declared globally which can grow during the
lifetime of a user's session, as long as the user no longer needs the contents of the indexed table.

The memory rules associated with an indexed table's scope still apply; this method and this procedure, however, allow users to intervene and to explicitly free the memory associated with an indexed table.

Examples

The following PL/SQL illustrates the method and the use of procedure FREE_UNUSED_USER_MEMORY.

```
CREATE PACKAGE foobar
    type number_idx_tbl is table of number indexed by binary_integer;

store1_table  number_idx_tbl;     --  PL/SQL indexed table
store2_table  number_idx_tbl;     --  PL/SQL indexed table
store3_table  number_idx_tbl;     --  PL/SQL indexed table
...
END;            --  end of foobar

DECLARE
    ...
    empty_table   number_idx_tbl;     --  uninitialized ("empty") version
BEGIN
    FOR i in 1..1000000 loop
        store1_table(i) := i;           --  load data
    END LOOP;
    ...
    store1_table := empty_table;      --  "truncate" the indexed table
    ...
    dbms_session.free_unused_user_memory;  -- give memory back to system
    store1_table(1) := 100;           --  index tables still declared;
    store2_table(2) := 200;           --  but truncated.
...
END;
```

146.4.10 GET_PACKAGE_MEMORY_UTILIZATION Procedure

This procedure describes static package memory usage.

The output collections describe memory usage in each instantiated package. Each package is described by its owner name, package name, used memory amount, and unused allocated memory amount. The amount of unused memory is greater than zero because of memory fragmentation and also because once used free memory chunks initially go to a free list owned by the package memory heap. They are released back to the parent heap only when the FREE_UNUSED_USER_MEMORY Procedure is invoked.

Syntax

```
DBMS_SESSION.GET_PACKAGE_MEMORY_UTILIZATION (  
    owner_names     OUT NOCOPY LNAME_ARRAY,    
    unit_names      OUT NOCOPY LNAME_ARRAY,    
    unit_types      OUT NOCOPY INTEGER_ARRAY,  
    used_amounts    OUT NOCOPY INTEGER_ARRAY,  
    free_amounts    OUT NOCOPY INTEGER_ARRAY);  
```
Parameters

Table 146-9  GET_PACKAGE_MEMORY_UTILIZATION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>Owner of package</td>
</tr>
<tr>
<td>unit_name</td>
<td>Name of package</td>
</tr>
<tr>
<td>unit_types</td>
<td>Value of the type# columns of the dictionary table obj$</td>
</tr>
<tr>
<td>used_amounts</td>
<td>Amount of allocated memory specified in bytes</td>
</tr>
<tr>
<td>free_amounts</td>
<td>Amount of available memory specified in bytes</td>
</tr>
</tbody>
</table>

146.4.11 IS_ROLE_ENABLED Function

This function determines if the named role is enabled for this session.

Syntax

```sql
DBMS_SESSION.IS_ROLE_ENABLED (rolename VARCHAR2) RETURN BOOLEAN;
```

Parameters

Table 146-10  IS_ROLE_ENABLED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolename</td>
<td>Name of the role</td>
</tr>
</tbody>
</table>

Return Values

Table 146-11  IS_ROLE_ENABLED Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_role_enabled</td>
<td>TRUE or FALSE, depending on whether the role is enabled</td>
</tr>
</tbody>
</table>

146.4.12 IS_SESSION_ALIVE Function

This function determines if the specified session is active.

Syntax

```sql
DBMS_SESSION.IS_SESSION_ALIVE (uniqueid VARCHAR2) RETURN BOOLEAN;
```
Parameters

Table 146-12  IS_SESSION_ALIVE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniqueid</td>
<td>Unique ID of the session: This is the same one as returned by UNIQUE_SESSION_ID.</td>
</tr>
</tbody>
</table>

Return Values

Table 146-13  IS_SESSION_ALIVE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_session_alive</td>
<td>TRUE or FALSE, depending on whether the session is active</td>
</tr>
</tbody>
</table>

146.4.13 LIST_CONTEXT Procedures

This procedure returns a list of active namespaces and contexts for the current session.

Syntax

```plaintext
TYPE AppCtxRecTyp IS RECORD (
    namespace VARCHAR2(30),
    attribute VARCHAR2(30),
    value VARCHAR2(256));

TYPE AppCtxTabTyp IS TABLE OF AppCtxRecTyp INDEX BY BINARY_INTEGER;

DBMS_SESSION.LIST_CONTEXT (
    list OUT AppCtxTabTyp,
    size OUT NUMBER);
```

Parameters

Table 146-14  LIST_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Buffer to store a list of application context set in the current session</td>
</tr>
</tbody>
</table>

Return Values

Table 146-15  LIST_CONTEXT Procedure Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list of (namespace, attribute, values) set in current session</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of entries in the buffer returned</td>
</tr>
</tbody>
</table>
Usage Notes

The context information in the list appears as a series of `<namespace> <attribute> <value>`. Because `list` is a table type variable, its size is dynamically adjusted to the size of returned list.

146.4.14 MODIFY_PACKAGE_STATE Procedure

This procedure is used to perform various actions (as specified by the `action_flags` parameter) on the session state of all PL/SQL program units active in the session. This takes effect after the PL/SQL call that made the current invocation finishes running.

The procedure uses the `DBMS_SESSION` constants listed in Table 146-17.

Syntax

```sql
DBMS_SESSION.MODIFY_PACKAGE_STATE(
    action_flags IN PLS_INTEGER);
```
Parameters

Table 146-16 MODIFY_PACKAGE_STATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action_flags</td>
<td>Bit flags that determine the action taken on PL/SQL program units:</td>
</tr>
<tr>
<td></td>
<td>DBMS_SESSION.FREE_ALL_RESOURCES (or 1)—frees all memory associated with each of the previously run PL/SQL programs from the session. Clears the current values of any package globals and closes cached cursors. On subsequent use, the PL/SQL program units are reinstantiated and package globals are reinitialized. Invoking MODIFY_PACKAGE_STATE with the DBMS_SESSION.FREE_ALL_RESOURCES parameter provides functionality identical to the DBMS_SESSION.RESET_PACKAGE() interface.</td>
</tr>
<tr>
<td></td>
<td>DBMS_SESSION.REINITIALIZE (or 2)—reinitializes packages without actually being freed and recreated from scratch. Instead the package memory is reused. In terms of program semantics, the DBMS_SESSION.REINITIALIZE flag is similar to the DBMS_SESSION.FREE_ALL_RESOURCES flag in that both have the effect of reinitializing all packages. However, DBMS_SESSION.REINITIALIZE should exhibit better performance than the DBMS_SESSION.FREE_ALL_RESOURCES option because:</td>
</tr>
<tr>
<td></td>
<td>· Packages are reinitialized without actually being freed and recreated from scratch. Instead the package memory gets reused.</td>
</tr>
<tr>
<td></td>
<td>· Any open cursors are closed, semantically speaking. However, the cursor resource is not actually freed. It is simply returned to the PL/SQL cursor cache. The cursor cache is not flushed. Hence, cursors corresponding to frequently accessed static SQL in PL/SQL remains cached in the PL/SQL cursor cache and the application does not incur the overhead of opening, parsing, and closing a new cursor for those statements on subsequent use.</td>
</tr>
<tr>
<td></td>
<td>· The session memory for PL/SQL modules without global state (such as types, stored-procedures) are not freed and recreated.</td>
</tr>
</tbody>
</table>

Usage Notes

See the parameter descriptions in Table 146-18 for the differences between the flags and why DBMS_SESSION.REINITIALIZE exhibits better performance than DBMS_SESSION.FREE_ALL_RESOURCES.

Table 146-17 Action_flags Constants for MODIFY_PACKAGE_STATE

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREE_ALL_RESOURCES</td>
<td>PLS_INTEGER:= 1</td>
</tr>
<tr>
<td>REINITIALIZE</td>
<td>PLS_INTEGER:= 2</td>
</tr>
</tbody>
</table>

- Reinitialization refers to the process of resetting all package variables to their initial values and running the initialization block (if any) in the package bodies. Consider the package:
package P is
    n number;
    m number := P2.foo;
    d date := SYSDATE;
    cursor c is select * from emp;
    procedure bar;
end P;
/
package body P is
    v varchar2(20) := 'hello';
    procedure bar is
        begin
            ...
        end;
    procedure init_pkg is
        begin
            ....
        end;
        begin
            -- initialization block
            init_pkg;
            ...
        end P;
/

For the package P, reinitialization involves:

- **Setting** P.n to NULL
- **Invoking function** P2.foo and setting P.m to the value returned from P2.foo
- **Setting** P.d to the return value of SYSDATE built-in
- **Closing cursor** P.c if it was previously opened
- **Setting** P.v to 'hello'
- **Running the initialization block in the package body**
- **The reinitialization for a package is done only if the package is actually referenced subsequently. Furthermore, the packages are reinitialized in the order in which they are referenced subsequently.**
- **When using** FREE_ALL_RESOURCES or REINITIALIZE, **make sure that resetting package variable values does not affect the application.**
- **Because** DBMS_SESSION.REINITIALIZE does not actually cause all the package state to be freed, in some situations, the application could use significantly more session memory than if the FREE_ALL_RESOURCES flag or the RESET_PACKAGE procedure had been used. For instance, after performing DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE), if the application does not refer to many of the packages that were previously referenced, then the session memory for those packages remains until the end of the session (or until DBMS_SESSION.RESET_PACKAGE is called).
- **Because the client-side PL/SQL code cannot reference remote package variables or constants, you must explicitly use the values of the constants. For example, DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE) does not compile on the client because it uses the constant DBMS_SESSION.REINITIALIZE.**
Instead, use `DBMS_SESSION.MODIFY_PACKAGE_STATE(2)` on the client, because the argument is explicitly provided.

Examples

This example illustrates the use of `DBMS_SESSION.MODIFY_PACKAGE_STATE`. Consider a package `P` with some global state (a cursor `c` and a number `cnt`). When the package is first initialized, the package variable `cnt` is 0 and the cursor `c` is CLOSED. Then, in the session, change the value of `cnt` to 111 and also execute an OPEN operation on the cursor. If you call `print_status` to display the state of the package, you see that `cnt` is 111 and that the cursor is OPEN. Next, call `DBMS_SESSION.MODIFY_PACKAGE_STATE`. If you print the status of the package `P` again using `print_status`, you see that `cnt` is 0 again and the cursor is CLOSED. If the call to `DBMS_SESSION.MODIFY_PACKAGE_STATE` had not been made, then the second call to `print_status` would have printed 111 and OPEN.

```sql
create or replace package P is
  cnt    number := 0;
  cursor c is select * from emp;
  procedure print_status;
end P;
/
show errors;

create or replace package body P is
  procedure print_status is
  begin
    dbms_output.put_line('P.cnt = ' || cnt);
    if c%ISOPEN then
      dbms_output.put_line('P.c is OPEN');
    else
      dbms_output.put_line('P.c is CLOSED');
    end if;
  end;
end P;
/
show errors;

SQL> set serveroutput on;
SQL> begin
  2   P.cnt := 111;
  3   open p.c;
  4   P.print_status;
  5  end;
  6  /
P.cnt = 111
P.c is OPEN

PL/SQL procedure successfully completed.

SQL> begin
  2   dbms_session.modify_package_state(dbms_session.reinitialize);
  3  end;
  4  /
PL/SQL procedure successfully completed.

SQL> set serveroutput on;
SQL>
SQL> begin
  2   P.print_status;
```
PL/SQL procedure successfully completed.

### 146.4.15 SESSION_TRACE_DISABLE Procedure

This procedure resets the session-level SQL trace for the session from which it was called. Client ID and service/module/action traces are not affected.

**Syntax**

```sql
DBMS_SESSION.SESSION_TRACE_DISABLE;
```

### 146.4.16 SESSION_TRACE_ENABLE Procedure

This procedure enables session-level SQL trace for the invoking session. Invoking this procedure results in SQL tracing of every SQL statement issued by the session.

**Syntax**

```sql
DBMS_SESSION.SESSION_TRACE_ENABLE(
    waits  IN   BOOLEAN DEFAULT TRUE,
    binds  IN   BOOLEAN DEFAULT FALSE,
    plan_stat IN   VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>waits</td>
<td>Specifies if wait information is to be traced</td>
</tr>
<tr>
<td>binds</td>
<td>Specifies if bind information is to be traced</td>
</tr>
<tr>
<td>plan_stat</td>
<td>Frequency at which we dump row source statistics. Value should be 'NEVER','FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.</td>
</tr>
</tbody>
</table>

### 146.4.17 RESET_PACKAGE Procedure

This procedure de-instantiates all packages in this session. It frees the package state.

**Note:**

See "SESSION_TRACE_ENABLE Procedure". The MODIFY_PACKAGE_STATE interface, introduced in Oracle9i, provides an equivalent of the RESET_PACKAGE capability. It is an efficient, lighter-weight variant for reinitializing the state of all PL/SQL packages in the session.
Memory used for caching the execution state is associated with all PL/SQL functions, procedures, and packages that were run in a session.

For packages, this collection of memory holds the current values of package variables and controls the cache of cursors opened by the respective PL/SQL programs. A call to **RESET_PACKAGE** frees the memory associated with each of the previously run PL/SQL programs from the session, and, consequently, clears the current values of any package globals and closes any cached cursors.

**RESET_PACKAGE** can also be used to reliably restart a failed program in a session. If a program containing package variables fails, then it is hard to determine which variables need to be reinitialized. **RESET_PACKAGE** guarantees that all package variables are reset to their initial values.

**Syntax**

```
DBMS_SESSION.RESET_PACKAGE;
```

**Usage Notes**

Because the amount of memory consumed by all executed PL/SQL can become large, you might use **RESET_PACKAGE** to trim down the session memory footprint at certain points in your database application. However, make sure that resetting package variable values does not affect the application. Also, remember that later execution of programs that have lost their cached memory and cursors will perform slower, because they need to re-create the freed memory and cursors.

**RESET_PACKAGE** does not free the memory, cursors, and package variables immediately when called.

---

**Note:**

**RESET_PACKAGE** only frees the memory, cursors, and package variables after the PL/SQL call that made the invocation finishes running.

---

For example, PL/SQL procedure **P1** calls PL/SQL procedure **P2**, and **P2** calls **RESET_PACKAGE**. The **RESET_PACKAGE** effects do not occur until procedure **P1** finishes execution (the PL/SQL call ends).

**Examples**

This SQL*Plus script runs a large program with many PL/SQL program units that may or may not use global variables, but it doesn't need them beyond this execution:

```
EXECUTE large_plsql_program1;
```

To free up PL/SQL cached session memory:

```
EXECUTE DBMS_SESSION.RESET_PACKAGE;
```

To run another large program:

```
EXECUTE large_plsql_program2;
```
146.4.18 SET_CONTEXT Procedure

This procedure sets the context, of which there are four types: session local, globally initialized, externally initialized, and globally accessed.

Of its five parameters, only the first three are required; the final two parameters are optional, used only in globally accessed contexts. Further parameter information appears in the parameter table and the usage notes.

Syntax

```
DBMS_SESSION.SET_CONTEXT (  
    namespace VARCHAR2,  
    attribute VARCHAR2,  
    value VARCHAR2,  
    username VARCHAR2,  
    client_id VARCHAR2 );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>Namespace of the application context to be set, limited to 30 bytes</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute of the application context to be set, limited to 30 bytes</td>
</tr>
<tr>
<td>value</td>
<td>Value of the application context to be set, limited to 4 kilobytes.</td>
</tr>
<tr>
<td>username</td>
<td>Database username attribute of the application context. Default: NULL</td>
</tr>
<tr>
<td>client_id</td>
<td>Application-specific client_id attribute of the application context (64-byte maximum). Default: NULL</td>
</tr>
</tbody>
</table>

Usage Notes

- The first three parameters are required for all types of context.
- The `username` parameter must be a valid SQL identifier.
- The `client_id` parameter must be a string of at most 64 bytes. It is case-sensitive and must match the argument provided for `set_identifier`.
- If the namespace parameter is a global context namespace, then the `username` parameter is matched against the current database user name in the session, and the `client_id` parameter is matched against the current `client_id` in the session. If these parameters are not set, NULL is assumed, enabling any user to see the context values.
- This procedure must be invoked directly or indirectly by the trusted package.
- The caller of SET_CONTEXT must be in the calling stack of a procedure that has been associated to the context namespace through a `CREATE CONTEXT` statement. The checking of the calling stack does not cross a DBMS boundary.
- No limit applies to the number of attributes that can be set in a namespace. An attribute retains its value during the user’s session unless it is reset by the user.
If the value of the parameter in the namespace has been set, \texttt{SET_CONTEXT} overwrites this value.

Any changes in context value are reflected immediately and subsequent calls to access the value through \texttt{SYS_CONTEXT} return the most recent value.

\textbf{See Also:}

- Oracle Database Security Guide for more information about
  - “Setting the username and client ID”
  - "Example: Creating a Global Application Context that Uses a Client Session ID"

### 146.4.19 SET_EDITION_DEFERRED Procedure

This procedure requests a switch to the specified edition. The switch takes effect at the end of the current client call.

\textbf{Syntax}

\begin{verbatim}
DBMS_SESSION.SET_EDITION_DEFERRED (    edition IN VARCHAR2);
\end{verbatim}

\textbf{Parameters}

\textbf{Table 146-20} \textit{SET_EDITION_DEFERRED Procedure Parameters}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>edition</td>
<td>Name of the edition to which to switch. The contents of the string are processed as a SQL identifier; double quotation marks must surround the remainder of the string if special characters or lower case characters are present in the edition's actual name and, if double quotation marks are not used, the contents are set in uppercase. The caller must have USE privilege on the named edition.</td>
</tr>
</tbody>
</table>

### 146.4.20 SET_IDENTIFIER Procedure

This procedure sets the client ID in the session.

\textbf{Syntax}

\begin{verbatim}
DBMS_SESSION.SET_IDENTIFIER (    client_id VARCHAR2);
\end{verbatim}
### Parameters

#### Table 146-21  SET_IDENTIFIER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>Case-sensitive application-specific identifier of the current database session. The maximum number of bytes for this parameter is 64 bytes. If the input exceeds 64 bytes, the additional bytes are truncated.</td>
</tr>
</tbody>
</table>

#### Usage Notes

- **SET_IDENTIFIER** sets the session's client id to the given value. This value can be used to identify sessions in `v$session` by means of `v$session.client_identifier`. It can also be used to identify sessions by means of `sys_context('USERENV','CLIENT_IDENTIFIER')`.
- This procedure is executable by **PUBLIC**.

### 146.4.21 SET_NLS Procedure

This procedure sets up your Globalization Support (NLS). It is equivalent to the SQL statement: `ALTER SESSION SET <nls_parameter> = <value>`.

#### Syntax

```sql
DBMS_SESSION.SET_NLS (  
  param VARCHAR2,  
  value VARCHAR2);  
```

#### Parameters

#### Table 146-22  SET_NLS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param</td>
<td>Globalization Support parameter. The parameter name must begin with 'NLS'.</td>
</tr>
<tr>
<td>value</td>
<td>Parameter value. If the parameter is a text literal, then it needs embedded single-quotes. For example, &quot;set_nls ('nls_date_format','&quot;DD-MON-YY&quot;)&quot;.</td>
</tr>
</tbody>
</table>

### 146.4.22 SET_ROLE Procedure

This procedure enables and disables roles. It is equivalent to the `SET ROLE` SQL statement.

#### Syntax

```sql
DBMS_SESSION.SET_ROLE (  
  role_cmd VARCHAR2);  
```
Parameters

Table 146-23  SET_ROLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_cmd</td>
<td>Text is appended to &quot;set role&quot; and then run as SQL</td>
</tr>
</tbody>
</table>

Usage Notes

Note that the procedure creates a new transaction if it is not invoked from within an existing transaction.

146.4.23 SET_SQL_TRACE Procedure

This procedure turns tracing on or off. It is equivalent to the SQL statement `ALTER SESSION SET SQL_TRACE ....`

Syntax

```
DBMS_SESSION.SET_SQL_TRACE (sql_trace boolean);
```

Parameters

Table 146-24  SET_SQL_TRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_trace</td>
<td>TRUE turns tracing on, FALSE turns tracing off</td>
</tr>
</tbody>
</table>

146.4.24 SLEEP Procedure

This procedure suspends the session for a specified period of time.

Syntax

```
DBMS_SESSION.SLEEP (seconds  IN NUMBER);
```

Parameters

Table 146-25  SLEEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds</td>
<td>Amount of time, in seconds, to suspend the session. The smallest increment can be entered in hundredths of a second; for example, 1.95 is a legal time value.</td>
</tr>
</tbody>
</table>
146.4.25 SWITCH_CURRENT_CONSUMER_GROUP Procedure

This procedure changes the current resource consumer group of a user's current session.

This lets you switch to a consumer group if you have the switch privilege for that particular group. If the caller is another procedure, then this enables the user to switch to a consumer group for which the owner of that procedure has switch privilege.

Syntax

```sql
DBMS_SESSION.switch_current_consumer_group (  
    new_consumer_group     IN  VARCHAR2,  
    old_consumer_group     OUT VARCHAR2,  
    initial_group_on_error IN  BOOLEAN);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_consumer_group</td>
<td>Name of consumer group to which you want to switch</td>
</tr>
<tr>
<td>old_consumer_group</td>
<td>Name of the consumer group from which you just switched out</td>
</tr>
<tr>
<td>initial_group_on_error</td>
<td>If TRUE, then sets the current consumer group of the caller to his/her initial consumer group in the event of an error</td>
</tr>
</tbody>
</table>

Return Values

This procedure outputs the old consumer group of the user in the parameter `old_consumer_group`.

**Note:**

- The `old_consumer_group` parameter returns the name of old consumer group only if it were set explicitly. That is, you might get NULL if the old consumer group was set by some mapping rules.
- You can switch back to the old consumer group later using the value returned in `old_consumer_group`.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>29368</td>
<td>Non-existent consumer group</td>
</tr>
<tr>
<td>1031</td>
<td>Insufficient privileges</td>
</tr>
<tr>
<td>29396</td>
<td>Cannot switch to OTHER_GROUPS consumer group</td>
</tr>
</tbody>
</table>
**Usage Notes**

The owner of a procedure must have privileges on the group from which a user was switched (`old_consumer_group`) in order to switch them back. There is one exception: The procedure can always switch the user back to his/her initial consumer group (skipping the privilege check).

By setting `initial_group_on_error` to `TRUE`, `SWITCH_CURRENT_CONSUMER_GROUP` puts the current session into the default group, if it can't put it into the group designated by `new_consumer_group`. The error associated with the attempt to move a session into `new_consumer_group` is raised, even though the current consumer group has been changed to the initial consumer group.

**Examples**

```sql
CREATE OR REPLACE PROCEDURE high_priority_task is
    old_group varchar2(30);
    prev_group varchar2(30);
    curr_user varchar2(30);
BEGIN
    -- switch invoker to privileged consumer group. If we fail to do so, an
    -- error is thrown, but the consumer group does not change
    -- because 'initial_group_on_error' is set to FALSE
    dbms_session.switch_current_consumer_group('tkrogrp1', old_group, FALSE);
    -- set up exception handler (in the event of an error, we do not want to
    -- return to caller while leaving the session still in the privileged
    -- group)
    BEGIN
        -- perform some operations while under privileged group
        EXCEPTION
            WHEN OTHERS THEN
                -- It is possible that the procedure owner does not have privileges
                -- on old_group. 'initial_group_on_error' is set to TRUE to make sure
                -- that the user is moved out of the privileged group in such a
                -- situation
                dbms_session.switch_current_consumer_group(old_group, prev_group, TRUE);
                RAISE;
    END;
    -- we've succeeded. Now switch to old_group, or if cannot do so, switch
    -- to caller's initial consumer group
    dbms_session.switch_current_consumer_group(old_group, prev_group, TRUE);
END high_priority_task;
/```
146.4.26 UNIQUE_SESSION_ID Function

This function returns an identifier that is unique for all sessions currently connected to this database. Multiple calls to this function during the same session always return the same result.

Syntax

```sql
DBMS_SESSION.UNIQUE_SESSION_ID
    RETURN VARCHAR2;
```

Pragmas

`pragma restrict_references(unique_session_id,WNDS,RNDS,WNPS);`

Return Values

**Table 146-28**  **UNIQUE_SESSION_ID Function Return Values**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique_session_id</td>
<td>Returns up to 24 bytes</td>
</tr>
</tbody>
</table>
The DBMS_SFW_ACL_ADMIN package provides interfaces for administering and managing access control policies for the "database service firewall" feature. Each policy is represented by an access control list (ACL) containing hosts that are allowed access to a specific database service. Local listeners and server processes validate all inbound client connections against the ACL.

This chapter contains the following topics:

- DBMS_SFW_ACL_ADMIN Security Model
- DBMS_SFW_ACL_ADMIN Operational Notes
- DBMS_SFW_ACL_ADMIN Examples
- Summary of DBMS_SFW_ACL_ADMIN Subprograms

147.1 DBMS_SFW_ACL_ADMIN Security Model

This package is owned by the DBSFWUSER schema. The procedures in this package can be run only by the DBSFWUSER user.

147.2 DBMS_SFW_ACL_ADMIN Operational Notes

These operation notes apply to DBMS_SFW_ACL_ADMIN.

- An ACL contains entries, which are called "ace", for "access control entries".
- You add entries to an ACL using the IP_ADD_ACE Procedure and IP_ADD_PDB_ACE Procedure. After calling these procedures, you call the COMMIT_ACL Procedure to send the updated ACL to the listeners. Similarly, if you remove entries from an ACL using the IP_REMOVE_ACE Procedure, IP_REMOVE_ACL Procedure, IP_REMOVE_PDB_ACE Procedure, or IP_REMOVE_PDB_ACL Procedure, you need to call the COMMIT_ACL Procedure to update the ACL on the listeners.
- Access control must be enabled using the new FIREWALL endpoint attribute and the new LOCAL_REGISTRATION_ADDRESS_listenerName parameter. The configuration can be done manually in listener.ora or through the Server Control Utility (SRVCTL). Refer to the Oracle Database Net Services Reference and the Oracle Real Application Clusters Administration and Deployment Guide for configuration instructions.
- You can query the IP_ACL table to show the committed ACLs. But to see the ACLs that have been sent to the local listeners, you have to query the V$IP_ACL or GV$IP_ACL view. The IP_ACL table can contain ACLs that are not in [G]V$IP_ACL because the database services for those ACLs were not running at the time of the commit. When the services are running, you can call the COMMIT_ACL procedure again to send the committed ACLs in the IP_ACL table to the local listeners.

In an Oracle RAC environment, GV$IP_ACL can be used to query ACLs across the database cluster, and V$IP_ACL to query ACLs in the connected instance.
147.3 DBMS_SFW_ACL_ADMIN Examples

These three examples show how DBMS_SFW_ACL_ADMIN can be used to administer and manage access control policies.

The following example adds three access control entries to the ACL and commits them.

```sql
## Connect to DBSFWUSER
SQL> connect dbsfwuser/<password>
Connected.

## Create an ACL for database service SVC1
SQL> exec dbms_sfw_acl_admin.ip_add_ace('svc1','192.168.12.1');
PL/SQL procedure successfully completed.

SQL> exec dbms_sfw_acl_admin.ip_add_ace('svc1','192.168.12.2');
PL/SQL procedure successfully completed.

SQL> exec dbms_sfw_acl_admin.ip_add_ace('svc1','test02.example.com');
PL/SQL procedure successfully completed.

## Commit the ACLs to the DB ACL table.
## This sends the ACLs for running services to ALL local Listeners
SQL> exec dbms_sfw_acl_admin.commit_acl;
PL/SQL procedure successfully completed.
```

The following example retrieves the ACLs committed from the previous example.

```sql
SQL> select * from ip_acl;
SERVICE_NAME        HOST
-------------------   -------------------
"SVC1.EXAMPLE.COM"   192.168.12.1
"SVC1.EXAMPLE.COM"   192.168.12.2
"SVC1.EXAMPLE.COM"   TEST02.EXAMPLE.COM
```

## View ACLs sent to the local Listeners
## NOTE: ACLs are sent ONLY to running services
```sql
SQL> select * from v$ip_acl;
SERVICE_NAME    HOST      CON_ID
-------------------  -------------------  ----
SVC1.EXAMPLE.COM  192.168.12.1     1
SVC1.EXAMPLE.COM  192.168.12.2     1
SVC1.EXAMPLE.COM  TEST02.EXAMPLE.COM  1
```

The following example adds access control entries for pluggable database "PDB1" using various host formats.

```sql
SQL> exec dbms_sfw_acl_admin.ip_add_pdb_ace('pdb1','192.168.12.3');
PL/SQL procedure successfully completed.

SQL> exec dbms_sfw_acl_admin.ip_add_pdb_ace('pdb1','192.168.12.0/23');
PL/SQL procedure successfully completed.

SQL> exec dbms_sfw_acl_admin.ip_add_pdb_ace('pdb1','192.168.12.*');
PL/SQL procedure successfully completed.

SQL> exec dbms_sfw_acl_admin.commit_acl;
PL/SQL procedure successfully completed.
```
147.4 Summary of DBMS_SFW_ACL_ADMIN Subprograms

This table lists the DBMS_SFW_ACL_ADMIN subprograms and briefly describes them.

Table 147-1  DBMS_SFW_ACL_ADMIN Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMIT_ACL Procedure</td>
<td>Commits changes to the ACL tables, and propagates the changes to the local listeners for database instances.</td>
</tr>
<tr>
<td>IP_ADD_ACE Procedure</td>
<td>Adds an access control entry to the ACL for a database service.</td>
</tr>
<tr>
<td>IP_ADD_PDB_ACE Procedure</td>
<td>Adds an access control entry to the ACL for all the database services in a pluggable database (PDB).</td>
</tr>
<tr>
<td>IP_REMOVE_ACE Procedure</td>
<td>Removes an entry from the ACL for a database service.</td>
</tr>
<tr>
<td>IP_REMOVE_ACL Procedure</td>
<td>Removes all entries from the ACL for a database service.</td>
</tr>
<tr>
<td>IP_REMOVE_PDB_ACE Procedure</td>
<td>Removes an access control entry from the ACL for all the database services in a pluggable database (PDB).</td>
</tr>
<tr>
<td>IP_REMOVE_PDB_ACL Procedure</td>
<td>Removes all entries from the ACL for all the database services in a pluggable database (PDB).</td>
</tr>
</tbody>
</table>

147.4.1 COMMIT_ACL Procedure

This procedure commits changes to the ACL tables. It also propagates the changes to the local listeners for database instances.

If you have changed access entries for database services, but the database services were not running at the time when you called the COMMIT_ACL procedure, then those changes will be committed to the ACL tables, but they will not be sent to the local listener. To send the entries to the listener, start up the database services, and call the COMMIT_ACL procedure again.

This procedure returns when the operation has completed successfully.

Syntax

```
DBMS_SFW_ACL_ADMIN.COMMIT_ACL;
```

Parameters

None

147.4.2 IP_ADD_ACE Procedure

This procedure adds an access control entry to the ACL for a database service.

Syntax

```
DBMS_SFW_ACL_ADMIN.IP_ADD_ACE (  
    p_service_name  IN  VARCHAR2,  
    p_host          IN  VARCHAR2);  
```
Parameters

Table 147-2  IP_ADD_ACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_service_name</td>
<td>The name of the database service for which you want to add an access control entry.</td>
</tr>
<tr>
<td>p_host</td>
<td>The host of the client that is allowed access to the service. This value can be a host name, an IPv4 address, or an IPv6 address. Wildcard &quot;***&quot; for IPv4 and CIDR format are also allowed.</td>
</tr>
</tbody>
</table>

147.4.3 IP_ADD_PDB_ACE Procedure

This procedure adds an access control entry to the ACL for all the database services in a pluggable database (PDB).

Syntax

```
DBMS_SFW_ACL_ADMIN.IP_ADD_PDB_ACE (
  p_pdb_name  IN  VARCHAR2,
  p_host      IN  VARCHAR2);
```

Parameters

Table 147-3  IP_ADD_PDB_ACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_pdb_name</td>
<td>The name of the PDB.</td>
</tr>
<tr>
<td>p_host</td>
<td>The host of the client that is allowed access to the database services in the PDB. This value can be a host name, an IPv4 address, or an IPv6 address. Wildcard &quot;***&quot; for IPv4 and CIDR format are also allowed.</td>
</tr>
</tbody>
</table>

147.4.4 IP_REMOVE_ACE Procedure

This procedure removes an entry from the ACL for a database service.

Syntax

```
DBMS_SFW_ACL_ADMIN.IP_REMOVE_ACE (
  p_service_name  IN  VARCHAR2,
  p_host          IN  VARCHAR2);
```

Parameters

Table 147-4  IP_REMOVE_ACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_service_name</td>
<td>The name of the database service from which you want to remove an access control entry.</td>
</tr>
</tbody>
</table>
Table 147-4  (Cont.) IP_REMOVE_ACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_host</td>
<td>The host that you want to remove from the ACL. This value can be a host name, an IPv4 address, or an IPv6 address. Wildcard *** for IPv4 and CIDR format are also allowed. This has to match the existing value exactly. You can query the IP_ACL table to get the list of entries for a database service.</td>
</tr>
</tbody>
</table>

147.4.5 IP_REMOVE_ACL Procedure

This procedure removes all entries from the ACL for a database service.

Syntax

```sql
DBMS_SFW_ACL_ADMIN.IP_REMOVE_ACL (p_service_name  IN  VARCHAR2);
```

Parameters

Table 147-5  IP_REMOVE_ACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_service_name</td>
<td>The name of the database service whose ACL you want to clear.</td>
</tr>
</tbody>
</table>

147.4.6 IP_REMOVE_PDB_ACE Procedure

This procedure removes an access control entry from the ACL for all the database services in the specified pluggable database (PDB).

Syntax

```sql
DBMS_SFW_ACL_ADMIN.IP_REMOVE_PDB_ACE (p_pdb_name  IN  VARCHAR2,
p_host      IN  VARCHAR2);
```

Parameters

Table 147-6  IP_REMOVE_PDB_ACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_pdb_name</td>
<td>The name of the PDB.</td>
</tr>
<tr>
<td>p_host</td>
<td>The host that you want to remove from the ACL. This value can be a host name, an IPv4 address, or an IPv6 address. Wildcard *** for IPv4 and CIDR format are also allowed. This has to match the existing value exactly. You can query the IP_ACL table to get the list of entries for a database service.</td>
</tr>
</tbody>
</table>
147.4.7 IP_REMOVE_PDB_ACL Procedure

This procedure removes all entries from the ACL for all the database services in the specified pluggable database (PDB).

Syntax

```
DBMS_SFW_ACL_ADMIN.IP_REMOVE_PDB_ACL (p_pdb_name IN VARCHAR2);
```

Parameters

**Table 147-7  IP_REMOVE_PDB_ACL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_pdb_name</td>
<td>The name of the PDB.</td>
</tr>
</tbody>
</table>
The **DBMS_SODA** package is a PL/SQL package implementing Simple Oracle Document Access (SODA). SODA allows you to use the Oracle Database as a NoSQL document store. The core abstraction provided by SODA is that of document collections. The **DBMS_SODA** package allows you to create, list, and delete document collections from PL/SQL, and to perform CRUD (create, replace, update, delete) operations on documents. All DDL functions are encapsulated within this package.

This chapter contains the following topics:

- **DBMS_SODA Security Model**
- **Summary of DBMS_SODA Subprograms**

### 148.1 DBMS_SODA Security Model

This package is available to users with the **SODA_APP** role.

All SODA types (packages and types) are **SYS types**. **PUBLIC** is granted **EXECUTE** privilege on the **DBMS_SODA** described in this chapter.

### 148.2 Summary of DBMS_SODA Subprograms

This table lists the **DBMS_SODA** subprograms in alphabetical order and briefly describes them.

#### Table 148-1  DBMS_SODA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CREATE_COLLECTION Function</strong></td>
<td>Creates a collection using the collection name and metadata.</td>
</tr>
<tr>
<td><strong>DROP_COLLECTION Function</strong></td>
<td>Drops an existing collection from the user’s schema. This also removes all the documents in the collection.</td>
</tr>
<tr>
<td><strong>LIST_COLLECTION_NAMES Function</strong></td>
<td>Lists the collection names in the user’s schema as a table of <strong>NVARCHAR2</strong>.</td>
</tr>
<tr>
<td><strong>OPEN_COLLECTION Function</strong></td>
<td>Opens an existing collection.</td>
</tr>
</tbody>
</table>
148.2.1 CREATE_COLLECTION Function

Creates a collection using the collection name and metadata. Uses the settings specified in the metadata and auto-assigns the ones that are not, and returns the collection object. If the metadata argument is omitted or set to NULL, a collection is created with default metadata. The returned collection is open for read and/or write operations. If a collection already exists, the function just opens and returns the collection object.

Syntax

```sql
DBMS_SODA.CREATE_COLLECTION (
    collection_Name      IN NVARCHAR2,
    metadata             IN VARCHAR2 DEFAULT NULL,
    create_Mode          IN PLS_INTEGER DEFAULT CREATE_MODE_DDL)
RETURN SODA_Collection_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection_Name</td>
<td>The name of the collection. The value of collection_Name is case-sensitive.</td>
</tr>
<tr>
<td>metadata</td>
<td>The metadata of the collection in VARCHAR2 format.</td>
</tr>
<tr>
<td>create_Mode</td>
<td>Valid values are:</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SODA.CREATE_MODE_DDL (default). First attempts to create a new table for the collection. If table exists already, tries to map that existing table to the collection. Minimal checking is performed to ensure the table shape matches supplied collection metadata (if not, then an error is returned).</td>
</tr>
<tr>
<td></td>
<td>- DBMS_SODA.CREATE_MODE_MAP. Tries to map an existing table to the collection. Minimal checking is performed to ensure the table shape matches supplied collection metadata (if not, then an error is returned).</td>
</tr>
</tbody>
</table>

Return Values

The function returns a Soda_Collection_T object representing the collection.

Exceptions

- Descriptor Error—if the input descriptor is invalid
- Error—if an error occurs while creating the collection
148.2.2 DROP_COLLECTION Function

Drops an existing collection from the user’s schema. This also removes all the documents in
the collection.

Syntax

```
DBMS_SODA.DROP_COLLECTION (  
    collection_Name      IN NVARCHAR2)  
RETURN NUMBER;
```

Parameters

**Table 148-3  DROP_COLLECTION Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection_Name</td>
<td>The name of the collection. The value of collection_Name is case-sensitive.</td>
</tr>
</tbody>
</table>

Return values

This function returns the following values:

- 1—if the collection was dropped successfully
- 0—if the collection does not exist

Exceptions

If an error occurs while dropping the collection, for example, due to uncommitted writes to the
collection or privilege issues.

148.2.3 LIST_COLLECTION_NAMES Function

This function returns a list of collection names in the user’s schema as a table of NVARCHAR2.

Syntax

```
DBMS_SODA.LIST_COLLECTION_NAMES ()
RETURN SODA_CollName_List_T;
```
Return Values

This function returns a list of collection names as a table of NVARCHAR2(255). The collection list is empty if there are no collections in the schema.

Exceptions

Error—if an error occurs while listing the collection names.

See Also:

Oracle Database SODA for PL/SQL Developer's Guide

148.2.4 OPEN_COLLECTION Function

Opens an existing collection for read and/or write operations.

Syntax

```
DBMS_SODA.OPEN_COLLECTION (
    collection_Name IN NVARCHAR2)
RETURN SODA_Collection_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection_Name</td>
<td>The name of the collection. The value of collection_Name is case-sensitive.</td>
</tr>
</tbody>
</table>

Return Values

This function returns the following values:

- a collection object which is open
- NULL, if the collection does not exist

Exceptions

Error—if an error occurs while creating the collection

See Also:

- Opening an Existing Document Collection with SODA for PL/SQL
- Checking Whether a Given Collection Exists with SODA for PL/SQL
The **DBMS_SHARED_POOL** package provides access to the shared pool, which is the shared memory area where cursors and PL/SQL objects are stored. **DBMS_SHARED_POOL** enables you to display the sizes of objects in the shared pool, and mark them for keeping or not-keeping in order to reduce memory fragmentation.

This chapter contains the following topics:

- Overview
- Operational Notes
- Summary of DBMS_SHARED_POOL Subprograms

### 149.1 DBMS_SHARED_POOL Overview

The procedures provided here may be useful when loading large PL/SQL objects. When large PL/SQL objects are loaded, users response time is affected because of the large number of smaller objects that need to be aged out from the shared pool to make room (due to memory fragmentation). In some cases, there may be insufficient memory to load the large objects.

**DBMS_SHARED_POOL** is also useful for frequently executed triggers. You may want to keep compiled triggers on frequently used tables in the shared pool.

Additionally, **DBMS_SHARED_POOL** supports sequences. Sequence numbers are lost when a sequence is aged out of the shared pool. **DBMS_SHARED_POOL** is useful for keeping sequences in the shared pool and thus preventing the loss of sequence numbers.

### 149.2 DBMS_SHARED_POOL Operational Notes

To create **DBMS_SHARED_POOL**, run the DBMSPOOL.SQL script. The PRVTPOOL.PLB script is automatically executed after DBMSPOOL.SQL runs. These scripts are not run by as part of standard database creation.

### 149.3 Summary of DBMS_SHARED_POOL Subprograms

This table lists the **DBMS_SHARED_POOL** subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORTED_REQUEST_THRESHOLDLD Procedure</td>
<td>Sets the aborted request threshold for the shared pool</td>
</tr>
<tr>
<td>KEEP Procedure</td>
<td>Keeps an object in the shared pool</td>
</tr>
<tr>
<td>MARKHOT Procedure</td>
<td>Marks a library cache object as a hot object</td>
</tr>
</tbody>
</table>
Table 149-1 (Cont.) DBMS_SHARED_POOL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PURGE Procedure</td>
<td>Purges the named object or specified heap(s) of the object</td>
</tr>
<tr>
<td>SIZES Procedure</td>
<td>Shows objects in the shared pool that are larger than the specified size</td>
</tr>
<tr>
<td>UNKEEP Procedure</td>
<td>Unkeeps the named object</td>
</tr>
<tr>
<td>UNMARKHOT Procedure</td>
<td>Unmarks a library cache object as a hot object</td>
</tr>
</tbody>
</table>

149.3.1 ABORTED_REQUEST_THRESHOLD Procedure

This procedure sets the aborted request threshold for the shared pool.

Syntax

```
DBMS_SHARED_POOL.ABORTED_REQUEST_THRESHOLD (threshold_size NUMBER);
```

Parameters

Table 149-2 ABORTED_REQUEST_THRESHOLD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold_size</td>
<td>Size, in bytes, of a request which does not try to free unpinned (not &quot;unkeep-ed&quot;) memory within the shared pool. The range of threshold_size is 5000 to ~2 GB inclusive.</td>
</tr>
</tbody>
</table>

Exceptions

An exception is raised if the threshold is not in the valid range.

Usage Notes

Usually, if a request cannot be satisfied on the free list, then the RDBMS tries to reclaim memory by freeing objects from the LRU list and checking periodically to see if the request can be fulfilled. After finishing this step, the RDBMS has performed a near equivalent of an 'ALTER SYSTEM FLUSH SHARED_POOL'.

Because this impacts all users on the system, this procedure "localizes" the impact to the process failing to find a piece of shared pool memory of size greater than threshold_size. This user gets the 'out of memory' error without attempting to search the LRU list.

149.3.2 KEEP Procedure

This procedure keeps an object in the shared pool. Once an object has been kept in the shared pool, it is not subject to aging out of the pool. This may be useful for...
frequently used large objects. When large objects are brought into the shared pool, several objects may need to be aged out to create a contiguous area large enough.

Syntax

```
DBMS_SHARED_POOL.KEEP (  
    name         VARCHAR2,  
    flag         CHAR DEFAULT 'P');
```

```
DBMS_SHARED_POOL.KEEP (  
    schema         VARCHAR2,  
    objname        VARCHAR2,  
    namespace      NUMBER,  
    heaps          NUMBER,  
    edition_name   VARCHAR2 DEFAULT NULL);
```

```
DBMS_SHARED_POOL.KEEP (  
    hash           VARCHAR2,  
    namespace      NUMBER,  
    heaps          NUMBER);
```

Parameters

Table 149-3 KEEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the object to keep.</td>
</tr>
<tr>
<td>flag</td>
<td>A character string indicating what kind of object to keep the name identifies. The string is not case sensitive. This parameter is optional.</td>
</tr>
<tr>
<td></td>
<td>If the parameter is not specified, the package assumes that the first parameter is the name of a package/procedure/function and will resolve the name.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'P' or 'p' to fully specify that the input is the name of a package/procedure/function.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'Q' or 'q' to specify that the input is the name of a sequence.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'R' or 'r' to specify that the input is the name of a trigger.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'T' or 't' to specify that the input is the name of a type.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'JS' or 'js' to specify that the input is the name of a java source.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'JC' or 'jc' to specify that the input is the name of a java class.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'JD' or 'jd' to specify that the input is the name of a java shared data.</td>
</tr>
<tr>
<td></td>
<td>• Set to 'C' or 'c' to specify that the input is the name of a cursor.</td>
</tr>
<tr>
<td>schema</td>
<td>The user name or the schema to which the object belongs to.</td>
</tr>
<tr>
<td>objname</td>
<td>The name of the object to keep.</td>
</tr>
<tr>
<td>namespace</td>
<td>A number indicating the library cache namespace in which the object has to be searched for.</td>
</tr>
<tr>
<td>heaps</td>
<td>The heaps to keep. For example, if heap 0 and heap 6 are to be kept.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition that the target object resides in. This parameter is optional.</td>
</tr>
<tr>
<td>hash</td>
<td>A 16-byte hash value for the object.</td>
</tr>
</tbody>
</table>
Exceptions

An exception is raised if the named object is not found.

Usage Notes

There are two kinds of objects:

- PL/SQL objects, triggers, sequences, types, and Java objects, which are specified by name.
- SQL cursor objects which are specified by a two-part number (indicating a location in the shared pool).

For example:

```sql
DBMS_SHARED_POOL.KEEP('scott.hispackage')
```

This keeps package `HISPACKAGE`, owned by `SCOTT`. The names for PL/SQL objects follow SQL rules for naming objects (for example, delimited identifiers and multibyte names are allowed). A cursor can be kept by `DBMS_SHARED_POOL.KEEP('0034CDFF, 20348871','C')`, `0034CDFF` being the `ADDRESS` and `20348871` the `HASH_VALUE`. Note that the complete hexadecimal address must be in the first 8 characters.

149.3.3 MARKHOT Procedure

This procedure marks a library cache object as a hot object.

Syntax

```sql
DBMS_SHARED_POOL.MARKHOT (
    schema VARCHAR2,
    objname VARCHAR2,
    namespace NUMBER DEFAULT 1,
    global BOOLEAN DEFAULT TRUE,
    edition_name VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_SHARED_POOL.MARKHOT (
    hash VARCHAR2,
    namespace NUMBER DEFAULT 1,
    global BOOLEAN DEFAULT TRUE);
```

Parameters

Table 149-4  MARKHOT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>User name or the schema to which the object belongs</td>
</tr>
<tr>
<td>objname</td>
<td>Name of the object</td>
</tr>
<tr>
<td>namespace</td>
<td>Number indicating the library cache namespace in which the object is to be searched. Views, such as <code>USER_OBJECTS</code> and <code>DBA_OBJECTS</code>, reflect the namespace as a number column, as do most dictionary tables such as <code>OBJ$</code>.</td>
</tr>
<tr>
<td>global</td>
<td>If <code>TRUE</code> (default), mark the object hot on all Oracle RAC instances</td>
</tr>
<tr>
<td>hash</td>
<td>16-byte hash value for the object</td>
</tr>
</tbody>
</table>
### Table 149-4 (Cont.) MARKHOT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>edition_name</td>
<td>Denotes the name of the edition that the target object resides in. This parameter is optional.</td>
</tr>
</tbody>
</table>

#### Exceptions

ORA-06502: An exception is raised if the named object cannot be found due to incorrect input.

ORA-04043: An exception is raised if the named object cannot be found (bad namespace, or hash input).

#### Usage Notes

If a package or type's specification is marked hot or unhot, then the corresponding package or type body will be implicitly marked as hot or unhot.

### 149.3.4 PURGE Procedure

This procedure purges the named object or specified heaps of the object.

#### Syntax

```sql
DBMS_SHARED_POOL.PURGE (
   name VARCHAR2,
   flag CHAR DEFAULT 'P',
   heaps NUMBER DEFAULT 1);

DBMS_SHARED_POOL.PURGE (
   schema VARCHAR2,
   objname VARCHAR2,
   namespace NUMBER,
   heaps NUMBER,
   edition_name VARCHAR2 DEFAULT NULL);

DBMS_SHARED_POOL.PURGE (
   hash VARCHAR2,
   namespace NUMBER,
   heaps NUMBER);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the object to purge. The value for this identifier is the concatenation of the address and hash_value columns from the v$sqlareas view. This is displayed by the SIZES procedure. Currently, TABLE and VIEW objects may not be purged.</td>
</tr>
</tbody>
</table>
Table 149-5  (Cont.) PURGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>(Optional) If this is not specified, then the package assumes that the first parameter is the name of a package/procedure/function and resolves the name. Set to 'P' or 'p' to fully specify that the input is the name of a package/procedure/function. Set to 'T' or 't' to specify that the input is the name of a type. Set to 'R' or 'r' to specify that the input is the name of a trigger. Set to 'Q' or 'q' to specify that the input is the name of a sequence. In case the first argument is a cursor address and hash-value, the parameter should be set to any character except 'P' or 'p' or 'Q' or 'q' or 'R' or 'r' or 'T' or 't'.</td>
</tr>
<tr>
<td>heaps</td>
<td>Heaps to be purged. For example, if heap 0 and heap 6 are to be purged: 1&lt;&lt;0</td>
</tr>
<tr>
<td>schema</td>
<td>User name or the schema to which the object belongs</td>
</tr>
<tr>
<td>objname</td>
<td>Name of the object to purge</td>
</tr>
<tr>
<td>namespace</td>
<td>Parameter is a number indicating the library cache namespace in which the object is to be searched</td>
</tr>
<tr>
<td>hash</td>
<td>16-byte hash value for the object</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition that the target object resides in. This parameter is optional.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-6570: An exception is raised if the named object cannot be found
ORA-6570: An object cannot be purged it marked as permanently kept

Usage Notes

All objects supported by the KEEP Procedure are supported for PURGE.

149.3.5 SIZES Procedure

This procedure shows objects in the shared_pool that are larger than the specified size. The name of the object is also given, which can be used as an argument to either the KEEP or UNKEEP calls.

Syntax

DBMS_SHARED_POOL.SIZES (
   minsize NUMBER);


c<i>149-6</c>
Parameters

Table 149-6  SIZES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minsize</td>
<td>Size, in kilobytes, over which an object must be occupying in the shared pool, in order for it to be displayed.</td>
</tr>
</tbody>
</table>

Usage Notes

Issue the SQLDBA or SQLPLUS 'SET SERVEROUTPUT ON SIZE XXXXX' command prior to using this procedure so that the results are displayed.

149.3.6 UNKEEP Procedure

This procedure unkeeps the named object.

Syntax

```sql
DBMS_SHARED_POOL.UNKEEP (
    name        VARCHAR2,
    flag        CHAR DEFAULT 'P');

DBMS_SHARED_POOL.UNKEEP (
    schema      VARCHAR2,
    objname     VARCHAR2,
    namespace   NUMBER,
    edition_name VARCHAR2 DEFAULT NULL);

DBMS_SHARED_POOL.UNKEEP (
    hash        VARCHAR2,
    namespace   NUMBER);
```

⚠️ WARNING:

This procedure may not be supported in the future if automatic mechanisms are implemented to make this unnecessary.

Parameters

Table 149-7  UNKEEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the object to unkeep.</td>
</tr>
<tr>
<td>flag</td>
<td>A character string indicating what kind of object to keep the name identifies. The string is not case sensitive. This parameter is optional. If the parameter is not specified, the package assumes that the first parameter is the name of a package/procedure/function and will resolve the name.</td>
</tr>
</tbody>
</table>
Table 149-7 (Cont.) UNKEEP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>The user name or the schema to which the object belongs to.</td>
</tr>
<tr>
<td>objname</td>
<td>The name of the object to unkeep.</td>
</tr>
<tr>
<td>namespace</td>
<td>A number that indicates the library cache namespace in which the object has to be searched for.</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition that the target object resides in. This parameter is optional.</td>
</tr>
<tr>
<td>hash</td>
<td>A 16-byte hash value for the object.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-06502: An exception is raised if the named object cannot be found

149.3.7 UNMARKHOT Procedure

This procedure unmarks a library cache object as a hot object.

Syntax

```sql
DBMS_SHARED_POOL.UNMARKHOT (  
    schema      VARCHAR2,  
    objname     VARCHAR2,  
    namespace   NUMBER DEFAULT 1,  
    global      BOOLEAN DEFAULT TRUE,  
    edition_name VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_SHARED_POOL.UNMARKHOT (  
    hash         VARCHAR2,  
    namespace   NUMBER DEFAULT 1,  
    global      BOOLEAN DEFAULT TRUE);
```

Parameters

Table 149-8 UNMARKHOT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>User name or the schema to which the object belongs</td>
</tr>
<tr>
<td>objname</td>
<td>Name of the object</td>
</tr>
<tr>
<td>namespace</td>
<td>Number indicating the library cache namespace in which the object is to be searched</td>
</tr>
<tr>
<td>global</td>
<td>If TRUE, unmark the object hot on all Oracle RAC instances. The default value of this parameter is TRUE.</td>
</tr>
<tr>
<td>hash</td>
<td>A 16-byte hash value for the object</td>
</tr>
<tr>
<td>edition_name</td>
<td>Denotes the name of the edition that the target object resides in. This parameter is optional.</td>
</tr>
</tbody>
</table>
Exceptions

ORA-06502: An exception is raised if the named object cannot be found due to incorrect input.

ORA-04043: An exception is raised if the named object cannot be found (bad namespace, or hash input, or non-existent object)

Usage Notes

If a package or type's specification is marked hot or unhot, then the corresponding package or type body will be implicitly marked as hot or unhot.
The DBMS_SPACE package enables you to analyze segment growth and space requirements.

This chapter contains the following topics:

- Security Model
- Data Structures
- Summary of DBMS_SPACE Subprograms

150.1 DBMS_SPACE Security Model

This package runs with SYS privileges. The execution privilege is granted to PUBLIC. Subprograms in this package run under the caller security. The user must have ANALYZE privilege on the object.

150.2 DBMS_SPACE Data Structures

The DBMS_SPACE package defines an OBJECT type, a RECORD type, and a TABLE type.

OBJECT Types
CREATE_TABLE_COST_COLINFO Object Type

RECORD Types
ASA_RECO_ROW Record Type

TABLE Types
ASA_RECO_ROW_TB Table Type

150.2.1 DBMS_SPACE CREATE_TABLE_COST_COLINFO Object Type

This type describes the datatype and size of a column in the table.

Syntax

```plsql
TYPE create_table_cost_colinfo IS OBJECT(
    col_type VARCHAR(200),
    col_size NUMBER)
```
Attributes

Table 150-1  CREATE_TABLE_COST_COLINFO Object Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_type</td>
<td>Column type</td>
</tr>
<tr>
<td>col_size</td>
<td>Column size</td>
</tr>
</tbody>
</table>

150.2.2 DBMS_SPACE ASA_RECO_ROW Record Type

This type contains the column type of individual columns returned by the ASA_RECOMMENDATIONS Function.

Syntax

```sql
TYPE asa_reco_row IS RECORD (  
tablespace_name       VARCHAR2(30),  
segment_owner         VARCHAR2(30),  
segment_name          VARCHAR2(30),  
segment_type          VARCHAR2(18),  
partition_name        VARCHAR2(30),  
allocated_space       NUMBER,      
used_space            NUMBER,      
reclaimable_space     NUMBER,      
chain_rowexcess       NUMBER,      
recommendations       VARCHAR2(1000),  
c1                    VARCHAR2(1000),  
c2                    VARCHAR2(1000),  
c3                    VARCHAR2(1000),  
task_id               NUMBER,      
msg_id                NUMBER);  
```

Attributes

Table 150-2  ASA_RECO_ROW Attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of the tablespace containing the object</td>
</tr>
<tr>
<td>segment_owner</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>segment_name</td>
<td>Name of the object</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of the segment 'TABLE','INDEX' and so on</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition</td>
</tr>
<tr>
<td>allocated_space</td>
<td>Space allocated to the segment</td>
</tr>
<tr>
<td>used_space</td>
<td>Space actually used by the segment</td>
</tr>
<tr>
<td>reclaimable_space</td>
<td>Reclaimable free space in the segment</td>
</tr>
<tr>
<td>chain_rowexcess</td>
<td>Percentage of excess chain row pieces that can be eliminated</td>
</tr>
<tr>
<td>recommendations</td>
<td>Recommendation or finding for this segment</td>
</tr>
<tr>
<td>c1</td>
<td>Command associated with the recommendation</td>
</tr>
<tr>
<td>c2</td>
<td>Command associated with the recommendation</td>
</tr>
</tbody>
</table>
Table 150-2  (Cont.) ASA_RECO_ROW Attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3</td>
<td>Command associated with the recommendation</td>
</tr>
<tr>
<td>task_id</td>
<td>Advisor Task that processed this segment</td>
</tr>
<tr>
<td>mesg_id</td>
<td>Message ID corresponding to the recommendation</td>
</tr>
</tbody>
</table>

Related Topics

- **DBMS_SPACE ASA_RECOMMENDATIONS Function**
  This function returns recommendations using the stored results of the auto segment advisor. This function returns results from the latest run on any given object.

150.2.3 DBMS_SPACE ASA_RECO_ROW_TB Table Type

The type `asa_reco_row_tb` is a table of `asa_reco_row`.

Syntax

```
TYPE asa_reco_row_tb IS TABLE OF asa_reco_row;
```

150.3 Summary of DBMS_SPACE Subprograms

This table lists the **DBMS_SPACE** subprograms and briefly describes them.

Table 150-3  **DBMS_SPACE Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASA_RECOMMENDATIONS Function</td>
<td>Returns recommendations/findings of segment advisor run automatically by the system or manually invoked by the user</td>
</tr>
<tr>
<td>CREATE_INDEX_COST Procedure</td>
<td>Determines the cost of creating an index on an existing table</td>
</tr>
<tr>
<td>CREATE_TABLE_COST Procedures</td>
<td>Determines the size of the table given various attributes</td>
</tr>
<tr>
<td>FREE_BLOCKS Procedure</td>
<td>Returns information about free blocks in an object (table, index, or cluster)</td>
</tr>
<tr>
<td>ISDATAFILEDROPPABLE_NAME Procedure</td>
<td>Checks whether a datafile is droppable</td>
</tr>
<tr>
<td>OBJECT_DEPENDENT_SEGMENTS Function</td>
<td>Returns the list of segments that are associated with the object</td>
</tr>
<tr>
<td>OBJECT_GROWTH_TREND Function</td>
<td>A table function where each row describes the space usage of the object at a specific point in time</td>
</tr>
<tr>
<td>SPACE_USAGE Procedures</td>
<td>Returns information about free blocks in an auto segment space managed segment</td>
</tr>
<tr>
<td>UNUSED_SPACE Procedure</td>
<td>Returns information about unused space in an object (table, index, or cluster)</td>
</tr>
</tbody>
</table>
150.3.1 DBMS_SPACE ASA_RECOMMENDATIONS Function

This function returns recommendations using the stored results of the auto segment advisor. This function returns results from the latest run on any given object.

Syntax

```
DBMS_SPACE.ASA_RECOMMENDATIONS (
  all_runs        IN    VARCHAR2 DEFAULT := TRUE,
  show_manual     IN    VARCHAR2 DEFAULT := TRUE,
  show_findings   IN    VARCHAR2 DEFAULT := FALSE)
RETURN ASA_RECO_ROW_TB PIPELINED;
```

Parameters

Table 150-4 ASA_RECOMMENDATIONS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_runs</td>
<td>Returns the results of all the auto advisor runs or only the results of the latest run. The valid values are TRUE and FALSE. The default value is TRUE.</td>
</tr>
<tr>
<td></td>
<td>If TRUE, returns recommendations/findings for all runs of auto segment advisor. If FALSE, returns the results of the LATEST run only. LATEST does not make sense for manual invocation of segment advisor. This is applicable only for auto advisor.</td>
</tr>
<tr>
<td></td>
<td>show_manual This parameter is used to indicate if the results of manual jobs should be included.</td>
</tr>
<tr>
<td></td>
<td>If TRUE, results of manual tasks are shown. If FALSE, results of manual tasks are not shown. Specifying manual=true does not negate the specification of auto advisor tasks.</td>
</tr>
<tr>
<td></td>
<td>However, the all_runs settings may override manual. If all_runs is FALSE, implying we only want to see the latest of auto advisor job, then manual may not be specified as true.</td>
</tr>
<tr>
<td></td>
<td>The valid values are TRUE and FALSE. The default value is TRUE.</td>
</tr>
<tr>
<td>show_findings</td>
<td>Shows only the findings instead of the recommendations. The valid values are TRUE and FALSE. The default value is FALSE.</td>
</tr>
</tbody>
</table>

Table 150-5 Parameter Usage

<table>
<thead>
<tr>
<th>all_runs</th>
<th>show_manual</th>
<th>show_findings</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>All findings from auto advisor and manual tasks.</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>All recommendations from auto advisor and manual tasks.</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>All findings from auto advisor tasks.</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>All recommendations from all auto advisor tasks.</td>
</tr>
</tbody>
</table>
### Table 150-5  (Cont.) Parameter Usage

<table>
<thead>
<tr>
<th>all_runs</th>
<th>show_manual</th>
<th>show_findings</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>N/A</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>N/A</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>Findings for the latest auto advisor task.</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>Recommendations from the latest auto advisor task.</td>
</tr>
</tbody>
</table>

#### 150.3.2 CREATE_INDEX_COST Procedure

This procedure determines the cost of creating an index on an existing table. The input is the DDL statement that will be used to create the index. The procedure will output the storage required to create the index.

**Syntax**

```sql
DBMS_SPACE.CREATE_INDEX_COST (
    ddl         IN    VARCHAR2,
    used_bytes  OUT   NUMBER,
    alloc_bytes OUT   NUMBER,
    plan_table  IN    VARCHAR2 DEFAULT NULL);
```

**Pragmas**

`pragma restrict_references(create_index_cost,WNDS);`

**Parameters**

**Table 150-6  CREATE_INDEX_COST Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddl</td>
<td>The create index DDL statement</td>
</tr>
<tr>
<td>used_bytes</td>
<td>The number of bytes representing the actual index data</td>
</tr>
<tr>
<td>alloc_bytes</td>
<td>Size of the index when created in the tablespace</td>
</tr>
<tr>
<td>plan_table</td>
<td>Which plan table to use, default NULL</td>
</tr>
</tbody>
</table>

**Usage Notes**

- The table on which the index is created must already exist.
- The computation of the index size depends on statistics gathered on the segment.
- It is imperative that the table must have been analyzed recently.
- In the absence of correct statistics, the results may be inaccurate, although the procedure will not raise any errors.
150.3.3 CREATE_TABLE_COST Procedures

This procedure is used in capacity planning to determine the size of the table given various attributes. The size of the object can vary widely based on the tablespace storage attributes, tablespace block size, and so on. There are two overloads of this procedure.

- The first version takes the column information of the table as argument and outputs the table size.
- The second version takes the average row size of the table as argument and outputs the table size.

This procedure can be used on tablespace of dictionary managed and locally managed extent management as well as manual and auto segment space management.

Syntax

```sql
DBMS_SPACE.CREATE_TABLE_COST (  
    tablespace_name    IN VARCHAR2,  
    avg_row_size       IN NUMBER,  
    row_count          IN NUMBER,  
    pct_free           IN NUMBER,  
    used_bytes         OUT NUMBER,  
    alloc_bytes        OUT NUMBER);

DBMS_SPACE.CREATE_TABLE_COST (  
    tablespace_name    IN VARCHAR2,  
    colinfos           IN CREATE_TABLE_COST_COLUMNS,  
    row_count          IN NUMBER,  
    pct_free           IN NUMBER,  
    used_bytes         OUT NUMBER,  
    alloc_bytes        OUT NUMBER);

CREATE TYPE create_table_cost_colinfo IS OBJECT (  
    COL_TYPE   VARCHAR(200),  
    COL_SIZE   NUMBER);
```

Parameters

Table 150-7  CREATE_TABLE_COST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>The tablespace in which the object will be created. The default is SYSTEM tablespace.</td>
</tr>
<tr>
<td>avg_row_size</td>
<td>The anticipated average row size in the table</td>
</tr>
<tr>
<td>colinfos</td>
<td>The description of the columns</td>
</tr>
<tr>
<td>row_count</td>
<td>The anticipated number of rows in the table</td>
</tr>
<tr>
<td>pct_free</td>
<td>The percentage of free space in each block for future expansion of existing rows due to updates</td>
</tr>
<tr>
<td>used_bytes</td>
<td>The space used by user data</td>
</tr>
<tr>
<td>alloc_bytes</td>
<td>The size of the object taking into account the tablespace extent characteristics</td>
</tr>
</tbody>
</table>
Usage Notes

- The `used_bytes` represent the actual bytes used by the data. This includes the overhead due to the block metadata, pctfree etc.
- The `alloc_bytes` represent the size of the table when it is created in the tablespace. This takes into account, the size of the extents in the tablespace and tablespace extent management properties.

Examples

```sql
-- review the parameters
SELECT argument_name, data_type, type_owner, type_name
FROM all_arguments
WHERE object_name = 'CREATE_TABLE_COST'
AND overload = 2

-- examine the input parameter type
SELECT text
FROM dba_source
WHERE name = 'CREATE_TABLE_COST_COLUMNS';

-- drill down further into the input parameter type
SELECT text
FROM dba_source
WHERE name = 'create_table_cost_colinfo';

set serveroutput on
DECLARE
    ub NUMBER;
    ab NUMBER;
    cl sys.create_table_cost_columns;
BEGIN
    cl := sys.create_table_cost_columns( sys.create_table_cost_colinfo('NUMBER',10),
                                          sys.create_table_cost_colinfo('VARCHAR2',30),
                                          sys.create_table_cost_colinfo('VARCHAR2',30),
                                          sys.create_table_cost_colinfo('DATE',NULL));
    DBMS_SPACE.CREATE_TABLE_COST('SYSTEM',cl,100000,0,ub,ab);
    DBMS_OUTPUT.PUT_LINE('Used Bytes: ' || TO_CHAR(ub));
    DBMS_OUTPUT.PUT_LINE('Alloc Bytes: ' || TO_CHAR(ab));
END;
/
```

150.3.4 FREE_BLOCKS Procedure

This procedure returns information about free blocks in an object (table, index, or cluster).

See SPACE_USAGE Procedures for returning free block information in an auto segment space managed segment.

Syntax

```
DBMS_SPACE.FREE_BLOCKS (segment_owner IN VARCHAR2,
                          segment_name  IN VARCHAR2,
                          segment_type  IN VARCHAR2,
```

Summary of DBMS_SPACE Subprograms
freelist_group_id IN NUMBER,
free_blks OUT NUMBER,
scan_limit IN NUMBER DEFAULT NULL,
partition_name IN VARCHAR2 DEFAULT NULL);

Pragmas
pragma restrict_references(free_blocks,WNDS);

Parameters

Table 150-8  FREE_BLOCKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>Schema name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER):</td>
</tr>
<tr>
<td>freelist_group_id</td>
<td>Freelist group (instance) whose free list size is to be computed</td>
</tr>
<tr>
<td>free_blks</td>
<td>Returns count of free blocks for the specified group</td>
</tr>
<tr>
<td>scan_limit</td>
<td>Maximum number of free list blocks to read (optional). Use a scan limit of X you are interested only in the question, “Do I have X blocks on the free list?”</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition name of the segment to be analyzed. This is only used for partitioned tables. The name of subpartition should be used when partitioning is composite.</td>
</tr>
</tbody>
</table>

Examples

The following uses the CLUS cluster in SCOTT schema with 4 freelist groups. It returns the number of blocks in freelist group 3 in CLUS.

```
DBMS_SPACE.FREE_BLOCKS('SCOTT', 'CLUS', 'CLUSTER', 3, :free_blocks);
```

Note:

An error is raised if scan_limit is not a positive number.
150.3.5 ISDATAFILEDROPPABLE_NAME Procedure

This procedure checks whether a datafile is droppable. This procedure may be called before actually dropping the file.

Syntax

```
DBMS_SPACE.ISDATAFILEDROPPABLE_NAME (    filename     IN     VARCHAR2,    value        OUT    NUMBER);
```

Pragmas

```
pragma restrict_references(free_blocks,WNDS);
```

Parameters

Table 150-9  ISDATAFILEDROPPABLE_NAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name of the file</td>
</tr>
<tr>
<td>value</td>
<td>Values: 0 if the file is not droppable, 1 if the file is droppable.</td>
</tr>
</tbody>
</table>

Examples

```
DECLARE   fname    VARCHAR2(100);   retval   NUMBER;BEGIN   SELECT file_name   INTO fname   FROM dba_data_files   WHERE file_name like '%empty%';DBMS_SPACE.ISDATAFILEDROPPABLE_NAME(fname, retval);DBMS_OUTPUT.PUT_LINE(retval);END;/
```

150.3.6 OBJECT_DEPENDENT_SEGMENTS Function

This table function, given an object, returns the list of segments that are associated with the object.

Syntax

```
DBMS_SPACE.OBJECT_DEPENDENT_SEGMENTS(    objowner    IN     VARCHAR2,    objname     IN     VARCHAR2,    partname    IN     VARCHAR2,    objtype     IN     NUMBER)    RETURN dependent_segments_table PIPELINED;
```

Parameters

Table 150-10  OBJECT_DEPENDENT_SEGMENTS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objowner</td>
<td>The schema containing the object</td>
</tr>
<tr>
<td>objname</td>
<td>The name of the object</td>
</tr>
<tr>
<td>partname</td>
<td>The name of the partition</td>
</tr>
</tbody>
</table>
Table 150-10  (Cont.) OBJECT_DEPENDENT_SEGMENTS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objtype</td>
<td>Type of the object:</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_TABLE constant positive := 1;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_NESTED_TABLE constant positive := 2;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_INDEX constant positive := 3;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_CLUSTER constant positive := 4;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_TABLE_PARTITION constant positive := 7;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_INDEX_PARTITION constant positive := 8;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_TABLE_SUBPARTITION constant positive := 9;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_INDEX_SUBPARTITION constant positive := 10;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_MV constant positive := 13;</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_TYPE_MVLOG constant positive := 14;</td>
</tr>
</tbody>
</table>

Return Values

The content of one row of a dependent_segments_table:

```sql
TYPE object_dependent_segment IS RECORD (  
  segment_owner      VARCHAR2(100),  
  segment_name       VARCHAR2(100),  
  segment_type       VARCHAR2(100),  
  tablespace_name    VARCHAR2(100),  
  partition_name     VARCHAR2(100),  
  lob_column_name    VARCHAR2(100));
```

Table 150-11  OBJECT_DEPENDENT_SEGMENT Type Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>The schema containing the segment</td>
</tr>
<tr>
<td>segment_name</td>
<td>The name of the segment</td>
</tr>
<tr>
<td>segment_type</td>
<td>The type of the segment, such as table, index or LOB</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>The name of the tablespace</td>
</tr>
<tr>
<td>partition_name</td>
<td>The name of the partition, if any</td>
</tr>
<tr>
<td>lob_column_name</td>
<td>The name of the LOB column, if any</td>
</tr>
</tbody>
</table>

150.3.7 OBJECT_GROWTH_TREND Function

This is a table function. The output is one or more rows where each row describes the space usage of the object at a specific point in time.

Either the space usage totals will be retrieved from Automatic Workload Repository Facilities (AWRF), or the current space usage will be computed and combined with space usage deltas retrieved from AWRF.
Syntax

```
DBMS_SPACE.OBJECT_GROWTH_TREND (
    object_owner           IN    VARCHAR2,
    object_name            IN    VARCHAR2,
    object_type            IN    VARCHAR2,
    partition_name         IN    VARCHAR2 DEFAULT NULL,
    start_time             IN    TIMESTAMP DEFAULT NULL,
    end_time               IN    TIMESTAMP DEFAULT NULL,
    interval               IN    DSINTERVAL_UNCONSTRAINED DEFAULT NULL,
    skip_interpolated      IN    VARCHAR2 DEFAULT 'FALSE',
    timeout_seconds        IN    NUMBER DEFAULT NULL,
    single_datapoint_flag  IN    VARCHAR2 DEFAULT 'TRUE')
RETURN object_growth_trend_table PIPELINED;
```

Parameters

**Table 150-12  OBJECT_GROWTH_TREND Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_owner</td>
<td>The schema containing the object</td>
</tr>
<tr>
<td>object_name</td>
<td>The name of the object</td>
</tr>
<tr>
<td>object_type</td>
<td>The type of the object</td>
</tr>
<tr>
<td>partition_name</td>
<td>The name of the partition</td>
</tr>
<tr>
<td>start_time</td>
<td>Statistics generated after this time will be used in generating the growth trend</td>
</tr>
<tr>
<td>end_time</td>
<td>Statistics generated until this time will be used in generating the growth trend</td>
</tr>
<tr>
<td>interval</td>
<td>The interval at which to sample</td>
</tr>
<tr>
<td>skip_interpolated</td>
<td>Whether interpolation of missing values should be skipped</td>
</tr>
<tr>
<td>timeout_seconds</td>
<td>The time-out value for the function in seconds</td>
</tr>
<tr>
<td>single_data_point_flag</td>
<td>Whether in the absence of statistics the segment should be sampled</td>
</tr>
</tbody>
</table>

Return Values

The object_growth_trend_row and object_growth_trend_table are used by the OBJECT_GROWTH_TREND table function to describe its output.

```
TYPE object_growth_trend_row IS RECORD(
    timepoint      TIMESTAMP,
    space_usage    NUMBER,
    space_alloc    NUMBER,
    quality        VARCHAR(20));
```

**Table 150-13  OBJECT_GROWTH_TREND_ROW Type Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timepoint</td>
<td>The time at which the statistic was recorded</td>
</tr>
<tr>
<td>space_usage</td>
<td>The space used by data</td>
</tr>
<tr>
<td>space_alloc</td>
<td>The size of the segment including overhead and unused space</td>
</tr>
</tbody>
</table>
Table 150-13  (Cont.) OBJECT_GROWTH_TREND_ROW Type Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quality</td>
<td>The quality of result: &quot;GOOD&quot;, &quot;INTERPOLATED&quot;, &quot;PROJECTION&quot;</td>
</tr>
</tbody>
</table>

TYPE object_growth_trend_table IS TABLE OF object_growth_trend_row;

150.3.8 SPACE_USAGE Procedures

This procedure has two variations to show space usage.

The first form of the procedure shows the space usage of data blocks under the segment High Water Mark. You can calculate usage for LOBs, LOB PARTITIONS and LOB SUBPARTITIONS. This procedure can only be used on tablespaces that are created with auto segment space management. The bitmap blocks, segment header, and extent map blocks are not accounted for by this procedure. Note that this overload cannot be used on SECUREFILE LOBS.

Note:

For LOB segments, the number of blocks that is returned from full_blocks and unformatted_blocks is actually the number of chunks for the LOB segment.

The second form of the procedure returns information about SECUREFILE LOB space usage. It will return the amount of space in blocks being used by all the SECUREFILE LOBs in the LOB segment. The procedure displays the space actively used by the LOB column, freed space that has retention expired, and freed space that has retention unexpired. Note that this overload can be used only on SECUREFILE LOBS.

Syntax

```sql
DBMS_SPACE.SPACE_USAGE(
    segment_owner            IN VARCHAR2,
    segment_name             IN VARCHAR2,
    segment_type             IN VARCHAR2,
    unformatted_blocks       OUT NUMBER,
    unformatted_bytes        OUT NUMBER,
    fs1_blocks               OUT NUMBER,
    fs1_bytes                OUT NUMBER,
    fs2_blocks               OUT NUMBER,
    fs2_bytes                OUT NUMBER,
    fs3_blocks               OUT NUMBER,
    fs3_bytes                OUT NUMBER,
    fs4_blocks               OUT NUMBER,
    fs4_bytes                OUT NUMBER,
    full_blocks              OUT NUMBER,
    full_bytes               OUT NUMBER,
    partition_name           IN VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_SPACE.SPACE_USAGE(
    segment_owner            IN VARCHAR2,
```

Chapter 150

Summary of DBMS_SPACE Subprograms
Parameters

### Table 150-14  SPACE_USAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>Schema name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_name</td>
<td>Name of the segment to be analyzed</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of the segment to be analyzed (TABLE, INDEX,或CLUSTER):</td>
</tr>
<tr>
<td></td>
<td>- TABLE</td>
</tr>
<tr>
<td></td>
<td>- TABLE PARTITION</td>
</tr>
<tr>
<td></td>
<td>- TABLE SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>- INDEX</td>
</tr>
<tr>
<td></td>
<td>- INDEX PARTITION</td>
</tr>
<tr>
<td></td>
<td>- INDEX SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>- CLUSTER</td>
</tr>
<tr>
<td></td>
<td>- LOB</td>
</tr>
<tr>
<td></td>
<td>- LOB PARTITION</td>
</tr>
<tr>
<td></td>
<td>- LOB SUBPARTITION</td>
</tr>
<tr>
<td>unformatted_blocks</td>
<td>For LOB segments, the number of blocks that is returned from unformatted_blocks is actually the number of chunks for the LOB segment.</td>
</tr>
<tr>
<td>unformatted_bytes</td>
<td>Total number of bytes unformatted</td>
</tr>
<tr>
<td>fs1_blocks</td>
<td>Number of blocks having at least 0 to 25% free space</td>
</tr>
<tr>
<td>fs1_bytes</td>
<td>Number of bytes having at least 0 to 25% free space</td>
</tr>
<tr>
<td>fs2_blocks</td>
<td>Number of blocks having at least 25 to 50% free space</td>
</tr>
<tr>
<td>fs2_bytes</td>
<td>Number of bytes having at least 25 to 50% free space</td>
</tr>
<tr>
<td>fs3_blocks</td>
<td>Number of blocks having at least 50 to 75% free space</td>
</tr>
<tr>
<td>fs3_bytes</td>
<td>Number of bytes having at least 50 to 75% free space</td>
</tr>
<tr>
<td>fs4_blocks</td>
<td>Number of blocks having at least 75 to 100% free space</td>
</tr>
<tr>
<td>fs4_bytes</td>
<td>Number of bytes having at least 75 to 100% free space</td>
</tr>
<tr>
<td>full_blocks</td>
<td>The number of blocks that is returned from full_blocks is actually the number of chunks for the LOB segment</td>
</tr>
<tr>
<td>full_bytes</td>
<td>Total number of bytes full in the segment</td>
</tr>
<tr>
<td>segment_size_blocks</td>
<td>Number of blocks allocated to the segment</td>
</tr>
<tr>
<td>segment_size_bytes</td>
<td>Number of bytes allocated to the segment</td>
</tr>
</tbody>
</table>
Table 150-14  (Cont.) SPACE_USAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>used_blocks</td>
<td>Number blocks allocated to the LOB that contains active data</td>
</tr>
<tr>
<td>used_bytes</td>
<td>Number bytes allocated to the LOB that contains active data</td>
</tr>
<tr>
<td>expired_blocks</td>
<td>Number of expired blocks used by the LOB to keep version data</td>
</tr>
<tr>
<td>expired_bytes</td>
<td>Number of expired bytes used by the LOB to keep version data</td>
</tr>
<tr>
<td>unexpired_blocks</td>
<td>Number of unexpired blocks used by the LOB to keep version data</td>
</tr>
<tr>
<td>unexpired_bytes</td>
<td>Number of unexpired bytes used by the LOB to keep version data</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition (NULL if not a partition)</td>
</tr>
</tbody>
</table>

Examples

```sql
variable unf number;
variable unfb number;
variable fs1 number;
variable fs1b number;
variable fs2 number;
variable fs2b number;
variable fs3 number;
variable fs3b number;
variable fs4 number;
variable fs4b number;
variable full number;
variable fullb number;

begin
  dbms_space.space_usage('U1','T',
    'TABLE',
    :unf, :unfb,
    :fs1, :fs1b,
    :fs2, :fs2b,
    :fs3, :fs3b,
    :fs4, :fs4b,
    :full, :fullb);
end;
/
print unf ;
print unfb ;
print fs4 ;
print fs4b;
print fs3 ;
print fs3b;
print fs2 ;
print fs2b;
print fs1 ;
print fs1b;
print full;
print fullb;
```
150.3.9 UNUSED_SPACE Procedure

This procedure returns information about unused space in an object (table, index, or cluster).

Syntax

```sql
DBMS_SPACE.UNUSED_SPACE (
    segment_owner              IN  VARCHAR2,
    segment_name               IN  VARCHAR2,
    segment_type               IN  VARCHAR2,
    total_blocks               OUT NUMBER,
    total_bytes                OUT NUMBER,
    unused_blocks              OUT NUMBER,
    unused_bytes               OUT NUMBER,
    last_used_extent_file_id   OUT NUMBER,
    last_used_extent_block_id  OUT NUMBER,
    last_used_block            OUT NUMBER,
    partition_name             IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 150-15  UNUSED_SPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>Schema name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_name</td>
<td>Segment name of the segment to be analyzed</td>
</tr>
<tr>
<td>segment_type</td>
<td>Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER):</td>
</tr>
<tr>
<td></td>
<td>• TABLE</td>
</tr>
<tr>
<td></td>
<td>• TABLE_PARTITION</td>
</tr>
<tr>
<td></td>
<td>• TABLE SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>• INDEX</td>
</tr>
<tr>
<td></td>
<td>• INDEX_PARTITION</td>
</tr>
<tr>
<td></td>
<td>• INDEX SUBPARTITION</td>
</tr>
<tr>
<td></td>
<td>• CLUSTER</td>
</tr>
<tr>
<td></td>
<td>• LOB</td>
</tr>
<tr>
<td></td>
<td>• LOB PARTITION</td>
</tr>
<tr>
<td></td>
<td>• LOB SUBPARTITION</td>
</tr>
<tr>
<td>total_blocks</td>
<td>Returns total number of blocks in the segment</td>
</tr>
<tr>
<td>total_bytes</td>
<td>Returns total number of blocks in the segment, in bytes</td>
</tr>
<tr>
<td>unused_blocks</td>
<td>Returns number of blocks which are not used</td>
</tr>
<tr>
<td>unused_bytes</td>
<td>Returns, in bytes, number of blocks which are not used</td>
</tr>
<tr>
<td>last_used_extent_file_id</td>
<td>Returns the file ID of the last extent which contains data</td>
</tr>
<tr>
<td>last_used_extent_block_id</td>
<td>Returns the starting block ID of the last extent which contains data</td>
</tr>
<tr>
<td>last_used_block</td>
<td>Returns the last block within this extent which contains data</td>
</tr>
<tr>
<td>partition_name</td>
<td>Partition name of the segment to be analyzed. This is only used for</td>
</tr>
<tr>
<td></td>
<td>partitioned tables; the name of subpartition should be used when partitioning</td>
</tr>
<tr>
<td></td>
<td>is compose.</td>
</tr>
</tbody>
</table>

Chapter 150
Summary of DBMS_SPACE Subprograms
Examples

The following declares the necessary bind variables and executes.

```sql
DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks, :total_bytes, :unused_blocks, :unused_bytes, :lastextf, :last_extb, :lastusedblock);
```
The **DBMS_SPACE_ADMIN** package provides functionality for locally managed tablespaces.

This chapter contains the following topics:

- Security Model
- Constants
- Operational Notes
- Summary of DBMS_SPACE_ADMIN Subprograms

See Also:

*Oracle Database Administrator’s Guide* for an example and description of using **DBMS_SPACE_ADMIN**.

### 151.1 DBMS_SPACE_ADMIN Security Model

This package runs with **SYS** privileges; therefore, any user who has privilege to execute the package can manipulate the bitmaps.

### 151.2 DBMS_SPACE_ADMIN Constants

The **DBMS_SPACE_ADMIN** package provides constants that can be used for specifying parameter values.

**Table 151-1**  DBMS_SPACE_ADMIN Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGMENT_VERIFY_EXTENTS</td>
<td>POSITIVE</td>
<td>1</td>
<td>Verifies that the space owned by segment is appropriately reflected in the bitmap as used</td>
</tr>
<tr>
<td>SEGMENT_VERIFY_EXTENTS</td>
<td>POSITIVE</td>
<td>2</td>
<td>Verifies that the space owned by segment is appropriately reflected in the bitmap as used and that no other segment claims any of this space to be used by it</td>
</tr>
<tr>
<td>SEGMENT_MARK_CORRUPT</td>
<td>POSITIVE</td>
<td>3</td>
<td>Marks a temporary segment as corrupt whereby facilitating its elimination from the dictionary (without space reclamation)</td>
</tr>
<tr>
<td>Constant</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SEGMENT_MARK_VALID</td>
<td>POSITIVE</td>
<td>4</td>
<td>Marks a corrupt temporary segment as valid. It is useful when the corruption in the segment extent map or elsewhere has been resolved and the segment can be dropped normally.</td>
</tr>
<tr>
<td>SEGMENT_DUMP_EXTENT_MAP</td>
<td>POSITIVE</td>
<td>5</td>
<td>Dumps the extent map for a given segment.</td>
</tr>
<tr>
<td>TABLESPACE_VERIFY_BITMAP</td>
<td>POSITIVE</td>
<td>6</td>
<td>Verifies the bitmap of the tablespace with extent maps of the segments in that tablespace to make sure everything is consistent.</td>
</tr>
<tr>
<td>TABLESPACE_EXTENT_MAKE_FREE</td>
<td>POSITIVE</td>
<td>7</td>
<td>Marks the block range (extent) as free in the bitmaps.</td>
</tr>
<tr>
<td>TABLESPACE_EXTENT_MAKE_USED</td>
<td>POSITIVE</td>
<td>8</td>
<td>Marks the block range (extent) as used in the bitmaps.</td>
</tr>
<tr>
<td>SEGMENT_VERIFY_BASIC</td>
<td>POSITIVE</td>
<td>9</td>
<td>Performs the basic metadata checks.</td>
</tr>
<tr>
<td>SEGMENT_VERIFY_DEEP</td>
<td>POSITIVE</td>
<td>10</td>
<td>Performs deep verification.</td>
</tr>
<tr>
<td>SEGMENT_VERIFY_SPECIFIC</td>
<td>POSITIVE</td>
<td>11</td>
<td>Performs a specific check for the segment.</td>
</tr>
<tr>
<td>HWM_CHECK</td>
<td>POSITIVE</td>
<td>12</td>
<td>Checks high water mark (HWM).</td>
</tr>
<tr>
<td>BMB_CHECK</td>
<td>POSITIVE</td>
<td>13</td>
<td>Checks integrity among L1, L2 and L3 BMBs (Bit Map Blocks).</td>
</tr>
<tr>
<td>SEG_DICT CHECK</td>
<td>POSITIVE</td>
<td>14</td>
<td>Checks consistency of segment header with corresponding SEG entry.</td>
</tr>
<tr>
<td>EXTENT_TS_BITMAP_CHECK</td>
<td>POSITIVE</td>
<td>15</td>
<td>Checks whether the tablespace bitmaps corresponding to the extent map are marked used.</td>
</tr>
<tr>
<td>DB_BACKPOINTER_CHECK</td>
<td>POSITIVE</td>
<td>16</td>
<td>Checks whether the L1 BMBs, L2 BMBs, L3 BMBs and data blocks point to the same parent segment.</td>
</tr>
<tr>
<td>EXTENT_SEGMENT_BITMAP_CHECK</td>
<td>POSITIVE</td>
<td>17</td>
<td>Checks whether the bitmap blocks are consistent with the extent map.</td>
</tr>
<tr>
<td>BITMAPS_CHECK</td>
<td>POSITIVE</td>
<td>18</td>
<td>Checks from the datablocks that the bitmap states representing the blocks are consistent.</td>
</tr>
<tr>
<td>TS_VERIFY_BITMAPS</td>
<td>POSITIVE</td>
<td>19</td>
<td>Checks whether the tablespace bitmaps are consistent with the extents belonging to that tablespace.</td>
</tr>
<tr>
<td>TS_VERIFY_DEEP</td>
<td>POSITIVE</td>
<td>20</td>
<td>Performs TS_VERIFY_BITMAPS and TS_VERIFY_SEGMENTS with DEEP option.</td>
</tr>
<tr>
<td>TS_VERIFY_SEGMENTS</td>
<td>POSITIVE</td>
<td>21</td>
<td>Performs ASSM_SEGMENT_VERIFY on all segments in the tablespace, taking either the BASIC or the DEEP option.</td>
</tr>
</tbody>
</table>
Table 151-1  (Cont.) DBMS_SPACE_ADMIN Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGMENTS_DUMP_BITMAP_S</td>
<td>POSITIVE</td>
<td>27</td>
<td>Dumps only bitmap block summaries</td>
</tr>
</tbody>
</table>

151.3 DBMS_SPACE_ADMIN Operational Notes

Before migrating the SYSTEM tablespace, certain conditions must be met. These conditions are enforced by the TABLESPACE_MIGRATE_TO_LOCAL procedure, except for the cold backup.

- The database must have a default temporary tablespace that is not SYSTEM.
- Dictionary-managed tablespaces cannot have any rollback segments.
- A locally managed tablespace must have at least one online rollback segment. If you are using automatic undo management, then an undo tablespace must be online.
- All tablespaces—except the tablespace containing the rollback segment or the undo tablespace—must be read-only.
- You must have a cold backup of the database.
- The system must be in restricted mode.

151.4 Summary of DBMS_SPACE_ADMIN Subprograms

This table lists the DBMS_SPACE_ADMIN subprograms and briefly describes them.

Table 151-2  DBMS_SPACE_ADMIN Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSM_SEGMENT_VERIFY Procedure</td>
<td>Verifies segments created in ASSM (Automatic Segment-Space Management) tablespaces</td>
</tr>
<tr>
<td>ASSM_TABLESPACE_VERIFY Procedure</td>
<td>Verifies ASSM tablespaces</td>
</tr>
<tr>
<td>DROP_EMPTY_SEGMENTS Procedure</td>
<td>Drops segments from empty tables or table fragments and dependent objects</td>
</tr>
<tr>
<td>GET_SEGADV_ATTRIB Procedure</td>
<td>returns the values of attributes of DBMS_SPACE_ADMIN package</td>
</tr>
<tr>
<td>MATERIALIZE_DEFERRED_SEGMENTS Procedure</td>
<td>Materializes segments for tables and table fragments with deferred segment creation and their dependent objects</td>
</tr>
<tr>
<td>SEGMENT_CORRUPT Procedure</td>
<td>Marks the segment corrupt or valid so that appropriate error recovery can be done</td>
</tr>
<tr>
<td>SEGMENT_DROP_CORRUPT Procedure</td>
<td>Drops a segment currently marked corrupt (without reclaiming space)</td>
</tr>
<tr>
<td>SEGMENT_DUMP Procedure</td>
<td>Dumps the segment header and extent maps of a given segment</td>
</tr>
<tr>
<td>SEGMENT_VERIFY Procedure</td>
<td>Verifies the consistency of the extent map of the segment</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SET_SEGADV_ATTRIB Procedure</td>
<td>Sets/changes the values of attributes of DBSM_SPACE_ADMIN package</td>
</tr>
<tr>
<td>TABLESPACE_FIX_BITMAPS Procedure</td>
<td>Marks the appropriate block range (extent) as free or used in bitmap</td>
</tr>
<tr>
<td>TABLESPACE_FIX_SEGMENTSTATES Procedure</td>
<td>Fixes the state of the segments in a tablespace in which migration was aborted</td>
</tr>
<tr>
<td>TABLESPACE_MIGRATE_FROM_LOCAL Procedure</td>
<td>Migrates a locally managed tablespace to dictionary-managed tablespace</td>
</tr>
<tr>
<td>TABLESPACE_MIGRATE_TO_LOCAL Procedure</td>
<td>Migrates a tablespace from dictionary-managed format to locally managed format</td>
</tr>
<tr>
<td>TABLESPACE_REBUILD_BITMAPS Procedure</td>
<td>Rebuilds the appropriate bitmaps</td>
</tr>
<tr>
<td>TABLESPACE_REBUILD_QUOTAS Procedure</td>
<td>Rebuilds quotas for given tablespace</td>
</tr>
<tr>
<td>TABLESPACE_RELOCATE_BITMAPS Procedure</td>
<td>Relocates the bitmaps to the destination specified</td>
</tr>
<tr>
<td>TABLESPACE_VERIFY Procedure</td>
<td>Verifies that the bitmaps and extent maps for the segments in the tablespace are synchronized</td>
</tr>
</tbody>
</table>

### 151.4.1 ASSM_SEGMENT_VERIFY Procedure

Given a segment definition, the procedure verifies the basic consistency of the space metadata blocks as well as consistency between space metadata and segment data blocks. This procedure verifies segments created in Automatic Segment Space Management (ASSM) tablespaces.

There is however a difference between basic verification and deep verification:

- Basic verification involves consistency checks of space metadata, such as integrity among level 1, level 2, level 3 bitmap blocks, consistency of segment extent map and level 1 bitmap ranges.
- Deep verification involves consistency checks between datablocks and space metadata blocks such as whether the datablocks point correctly to the parent level 1 bitmap blocks, and whether the freeness states in the datablocks are consistent with the freeness states of bits in level 1 bitmap blocks corresponding to the datablocks.

**Syntax**

```
DBMS_SPACE_ADMIN.ASSM_SEGMENT_VERIFY (segment_owner IN VARCHAR2, segment_name IN VARCHAR2, segment_type IN VARCHAR2, partition_name IN VARCHAR2, verify_option IN POSITIVE DEFAULT SEGMENT_VERIFY_BASIC, attrib IN POSITIVE DEFAULT NULL);
```
### Parameters

#### Table 151-3 ASSM_SEGMENT_VERIFY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment_owner</td>
<td>Schema that owns the segment</td>
</tr>
<tr>
<td>segment_name</td>
<td>Name of the segment to be verified</td>
</tr>
<tr>
<td>segment_type</td>
<td>Segment namespace is one of TABLE, TABLE PARTITION, TABLE SUBPARTITION, INDEX, INDEX PARTITION, INDEX SUBPARTITION, LOB, LOB PARTITION, LOB SUBPARTITION, CLUSTER</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition or subpartition</td>
</tr>
<tr>
<td>verify_option</td>
<td>One of the following options:</td>
</tr>
<tr>
<td>attrib</td>
<td>When option SEGMENT_VERIFY_SPECIFIC is specified as option, attrib can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>• HWM_CHECK := 12. Checks whether high water mark information is accurate</td>
</tr>
<tr>
<td></td>
<td>• BMB_CHECK := 13. Checks whether space bitmap blocks have correct backpointers to the segment header</td>
</tr>
<tr>
<td></td>
<td>• SEG_DICT_CHECK := 14. Checks whether dictionary information for segment is accurate</td>
</tr>
<tr>
<td></td>
<td>• EXTENT_TS_BITMAP_CHECK := 15. Checks whether extent maps are consistent with file level bitmaps</td>
</tr>
<tr>
<td></td>
<td>• DB_BACKPOINTER_CHECK := 16. Checks whether datablocks have correct backpointers to the space metadata blocks</td>
</tr>
<tr>
<td></td>
<td>• EXTENT_SEGMENT_BITMAP_CHECK := 17. Checks whether extent map in the segment matches the bitmaps in the segment</td>
</tr>
<tr>
<td></td>
<td>• BITMAPS_CHECK := 18. Checks whether space bitmap blocks are accurate</td>
</tr>
</tbody>
</table>

### Usage Notes

- **Using this procedure requires** `SYSDBA` **privileges.**
- **You can determine the relative file # and header block #** (`header_relative_file` and `header_block` parameters) by querying `DBA_SEGMENTS`.
- **This procedure outputs a dump file named** `sid_ora_process_ID.trc` **to the location specified in the `USER_DUMP_DEST` initialization parameter.**
151.4.2 ASSM_TABLESPACE_VERIFY Procedure

This procedure verifies all the segments created in an ASSM tablespace. The verification for each segment performs basic consistency checks of the space metadata blocks as well as consistency checks between space metadata and segment data blocks.

Syntax

DBMS_SPACE_ADMIN.ASSM_TABLESPACE_VERIFY (  
tablespace_name   IN VARCHAR2,  
ts_option         IN POSITIVE,  
segment_option    IN POSITIVE DEFAULT NULL);

Parameters

Table 151-4  ASSM_TABLESPACE_VERIFY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of the tablespace to verify. The tablespace must be an ASSM tablespace.</td>
</tr>
</tbody>
</table>
| ts_option    | • TS_VERIFY_BITMAPS := 19. The bitmaps are verified against the extents. This detects bits that are marked used or free wrongly and detects multiple allocation of extents. The file metadata is validated against file$ and control file.  
  • TS_VERIFY_DEEP := 20. This option is used to verify the file bitmaps as well perform checks on all the segments.  
  • TS_VERIFY_SEGMENTS := 21. This option is used to invoke SEGMENT_VERIFY on all the segments in the tablespace. Optionally you can write a script that queries all the segments in the tablespace and invoke SEGMENT_VERIFY. |
| segment_option | When TS_VERIFY_SEGMENTS is specified, segment_option can be one of the following:  
  • SEGMENT_VERIFY_BASIC := 9  
  • SEGMENT_VERIFY_DEEP := 10  
The value of segment_option is NULL when TS_VERIFY_DEEP or TS_VERIFY_BITMAPS is specified. |

Usage Notes

• Using this procedure requires SYSDBA privileges.
• This procedure outputs a dump file named sid_ora_process_ID.trc to the location specified in the USER_DUMP_DEST initialization parameter.
151.4.3 DROP_EMPTY_SEGMENTS Procedure

This procedure drops segments from empty tables or table fragments and dependent objects.

Syntax

```sql
DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS (  
  schema_name       IN     VARCHAR2   DEFAULT NULL,  
  table_name        IN     VARCHAR2   DEFAULT NULL,  
  partition_name    IN     VARCHAR2   DEFAULT NULL);
```

Parameters

Table 151-5  DROP_EMPTY_SEGMENTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of schema</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of table</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of partition</td>
</tr>
</tbody>
</table>

Usage Notes

Given a schema name, this procedure scans all tables in the schema. For each table, if the table or any of its fragments are found to be empty, and the table satisfies certain criteria (restrictions being the same as those described in "Restrictions on Deferred Segment Creation"), then the empty table fragment and associated index segments are dropped along with the corresponding LOB data and index segments. A subsequent insert creates segments with the same properties.

Optionally:

- **No** `schema_name` is specified, in which case tables belonging to all schemas are scanned
- **Both** `schema_name` and `table_name` are specified to perform the operation on a specified table
- All three arguments are supplied, restricting the operation to the partition and its dependent objects

151.4.4 GET_SEGADV_ATTRIB Procedure

This procedure returns the values of attributes of `DBMS_SPACE_ADMIN` package.

Syntax

```sql
DBMS_SPACE_ADMIN.GET_SEGADV_ATTRIB(  
  attribute  IN  NUMBER,  
  value      OUT  NUMBER);
```
Parameters

Table 151-6  GET_SEGADV_ATTRIB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>Supported attributes:</td>
</tr>
<tr>
<td></td>
<td>· COMP_ADVISOR — Provides an option to enable or disable Compression Advisor for Automatic Segment Advisor. By default Compression Advisor is enabled for Automatic Segment Advisor.</td>
</tr>
<tr>
<td></td>
<td>· COMP_LOB — Provides an option to enable or disable Compression Advisor for the tables with LOB columns while Automatic Segment Advisor is running. By default Compression Advisor is enabled for tables with LOB columns.</td>
</tr>
<tr>
<td>value</td>
<td>Supported values:</td>
</tr>
<tr>
<td></td>
<td>· ATTR_ENABLE : 1</td>
</tr>
<tr>
<td></td>
<td>· ATTR_DISABLE : 0</td>
</tr>
</tbody>
</table>

151.4.5 MATERIALIZE_DEFERRED_SEGMENTS Procedure

This procedure materializes segments for tables and table fragments with deferred segment creation and their dependent objects.

Syntax

```
DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS (  
schema_name IN VARCHAR2 DEFAULT NULL,  
table_name IN VARCHAR2 DEFAULT NULL,  
partition_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 151-7  MATERIALIZE_DEFERRED_SEGMENTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of schema</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of table</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of partition</td>
</tr>
</tbody>
</table>

Usage Notes

Given a schema name, this procedure scans all tables in the schema. For each table, if the deferred or delayed segment property is set for the table or any of its fragments, then a new segment is created for those fragments and their dependent objects.

Optionally:

- No schema_name is specified, in which case tables belonging to all schemas are scanned.
Both `schema_name` and `table_name` are specified to perform the operation on a specified table.

All three arguments are supplied, restricting the operation to the partition and its dependent objects.

### 151.4.6 SEGMENT_CORRUPT Procedure

This procedure marks the segment corrupt or valid so that appropriate error recovery can be performed.

It cannot be used on the `SYSTEM` tablespace.

#### Syntax

```sql
DBMS_SPACE_ADMIN.SEGMENT_CORRUPT (    tablespace_name         IN    VARCHAR2,
  header_relative_file    IN    POSITIVE,
  header_block            IN    NUMBER,
  corrupt_option          IN    POSITIVE  DEFAULT SEGMENT_MARK_CORRUPT);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tablespace_name</code></td>
<td>Name of tablespace in which segment resides</td>
</tr>
<tr>
<td><code>header_relative_file</code></td>
<td>Relative file number of segment header</td>
</tr>
<tr>
<td><code>header_block</code></td>
<td>Block number of segment header</td>
</tr>
<tr>
<td><code>corrupt_option</code></td>
<td><code>SEGMENT_MARK_CORRUPT</code> (default) or <code>SEGMENT_MARK_VALID</code></td>
</tr>
</tbody>
</table>

#### Usage Notes

You can determine the relative file number and block number (`header_relative_file` and `header_block` parameter) of the segment header block by querying `DBA_SEGMENTS`.

#### Examples

The following example marks the segment as corrupt:

```sql
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, DBMS_SPACE_ADMIN.SEGMENT_MARK_CORRUPT);
```

Alternately, the next example marks a corrupt segment valid:

```sql
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, DBMS_SPACE_ADMIN.SEGMENT_MARK_VALID);
```

### 151.4.7 SEGMENT_DROP_CORRUPT Procedure

This procedure drops a segment currently marked corrupt (without reclaiming space).

For this to work, the segment must be marked `temporary`. To mark a corrupt segment as temporary, issue a `DROP` command on the segment.
Syntax

DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT (
    tablespace_name    IN    VARCHAR2,
    header_relative_file IN    POSITIVE,
    header_block       IN    NUMBER);

Parameters

Table 151-9  SEGMENT_DROP_CORRUPT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace in which segment resides</td>
</tr>
<tr>
<td>header_relative_file</td>
<td>Relative file number of segment header</td>
</tr>
<tr>
<td>header_block</td>
<td>Block number of segment header</td>
</tr>
</tbody>
</table>

Usage Notes

• The space for the segment is not released, and it must be fixed by using the TABLESPACE_FIX_BITMAPS Procedure or the TABLESPACE_REBUILD_BITMAPS Procedure.

• The procedure cannot be used on the SYSTEM tablespace.

• You can determine the relative file number and block number (header_relative_file and header_block parameter) of the segment header block by querying DBA_SEGMENTS.

Examples

EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT('USERS', 4, 33);

151.4.8 SEGMENT_DUMP Procedure

This procedure dumps the segment header and bitmap blocks of a specific segment to the location specified in the USER_DUMP_DEST initialization parameter.

Syntax

DBMS_SPACE_ADMIN.SEGMENT_DUMP (
    tablespace_name    IN    VARCHAR2,
    header_relative_file IN    POSITIVE,
    header_block       IN    NUMBER,
    dump_option        IN    POSITIVE DEFAULT SEGMENT_DUMP_EXTENT_MAP);

Parameters

Table 151-10  SEGMENT_DUMP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace in which segment resides</td>
</tr>
<tr>
<td>header_relative_file</td>
<td>Relative file number of segment header</td>
</tr>
<tr>
<td>header_block</td>
<td>Block number of segment header</td>
</tr>
</tbody>
</table>
Table 151-10  (Cont.) SEGMENT_DUMP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dump_option</td>
<td>One of the following options:</td>
</tr>
<tr>
<td></td>
<td>• SEGMENT_DUMP_EXTENT_MAP</td>
</tr>
<tr>
<td></td>
<td>• SEGMENT_DUMP_BITMAP_SUMMARY</td>
</tr>
</tbody>
</table>

Usage Notes

- You can produce a slightly abbreviated dump, which includes the segment header and bitmap block summaries, without percent-free states of each block if you pass `SEGMENT_DUMP_BITMAP_SUMMARY` as the `dump_option` parameter.
- You can determine the relative file number and block number (header_relative_file and header_block parameter) of the segment header block by querying `DBA_SEGMENTS.HEADER_FILE`. If `HEADER_FILE` is greater than 1023 then use `DBA_DATA_FILES.RELATIVE_FNO`.

Examples

EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('USERS', 4, 33);

151.4.9 SEGMENT_VERIFY Procedure

This procedure checks the consistency of the segment extent map with the tablespace file bitmaps.

Syntax

```sql
DBMS_SPACE_ADMIN.SEGMENT_VERIFY (  
tablespace_name       IN    VARCHAR2,  
header_relative_file  IN    POSITIVE,  
header_block          IN    NUMBER,  
verify_option         IN    POSITIVE DEFAULT SEGMENT_VERIFY_EXTENTS);
```

Parameters

Table 151-11  SEGMENT_VERIFY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace in which segment resides</td>
</tr>
<tr>
<td>header_relative_file</td>
<td>Relative file number of segment header</td>
</tr>
<tr>
<td>header_block</td>
<td>Block number of segment header</td>
</tr>
<tr>
<td>verify_option</td>
<td>What kind of check to do: <code>SEGMENT_VERIFY_EXTENTS</code> or <code>SEGMENT_VERIFY_EXTENTS_GLOBAL</code></td>
</tr>
</tbody>
</table>

Usage Notes

- Anomalies are output as block range, bitmap-block, bitmap-block-range, anomaly-information, in the trace file for all block ranges found to have incorrect space representation. The kinds of problems which would be reported are free space not
considered free, used space considered free, and the same space considered used by multiple segments.

- You can determine the relative file number and block number (header_relative_file and header_block parameter) of the segment header block by querying DBA_SEGMENTS.

Examples

The following example verifies that the segment with segment header at relative file number 4, block number 33, has its extent maps and bitmaps synchronized.

EXECUTE DBMS_SPACE_ADMIN.SEGMENT_VERIFY('USERS', 4, 33, DBMS_SPACE_ADMIN.SEGMENT_VERIFY_EXTENTS);

151.4.10 SET_SEGADV_ATTRIB Procedure

This procedure sets the values of attributes of DBMS_SPACE_ADMIN package.

Syntax

```
DBMS_SPACE_ADMIN.SET_SEGADV_ATTRIB(
    attribute IN NUMBER,
    value      IN NUMBER);
```

Parameters

Table 151-12  SET_SEGADV_ATTRIB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>Supported attributes:</td>
</tr>
<tr>
<td></td>
<td>· COMP_ADVISOR — Provides an option to enable or disable Compression Advisor for Automatic Segment Advisor. By default Compression Advisor is enabled for Automatic Segment Advisor.</td>
</tr>
<tr>
<td></td>
<td>· COMP_LOB — Provides an option to enable or disable Compression Advisor for the tables with LOB columns while Automatic Segment Advisor is running. By default Compression Advisor is enabled for tables with LOB columns.</td>
</tr>
<tr>
<td>value</td>
<td>Supported values:</td>
</tr>
<tr>
<td></td>
<td>· ATTR_ENABLE : 1</td>
</tr>
<tr>
<td></td>
<td>· ATTR_DISABLE : 0</td>
</tr>
</tbody>
</table>

151.4.11 TABLESPACE_FIX_BITMAPS Procedure

This procedure marks the appropriate block range (extent) as free or used in bitmap. It cannot be used on the SYSTEM tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS ( |
    tablespace_name IN VARCHAR2, |
    dbarange_relative_file IN POSITIVE, |
    dbarange_begin_block IN POSITIVE, |
```
Parameters

Table 151-13  TABLESPACE_FIX_BITMAPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace</td>
</tr>
<tr>
<td>dbarange_relative_file</td>
<td>Relative file number of block range (extent)</td>
</tr>
<tr>
<td>dbarange_begin_block</td>
<td>Block number of beginning of extent</td>
</tr>
<tr>
<td>dbarange_end_block</td>
<td>Block number (inclusive) of end of extent</td>
</tr>
<tr>
<td>fix_option</td>
<td>One of the following options:</td>
</tr>
<tr>
<td></td>
<td>· TABLESPACE_EXTENT_MAKE_FREE</td>
</tr>
<tr>
<td></td>
<td>· TABLESPACE_EXTENT_MAKE_USED</td>
</tr>
</tbody>
</table>

Examples

The following example marks bits for 51 blocks for relative file number 4, beginning at block number 33 and ending at 83, as USED in bitmaps.

EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS('USERS', 4, 33, 83, DBMS_SPACE_ADMIN.EXTENT_MAKE_USED);

Alternatively, specifying an option of TABLESPACE_EXTENT_MAKE_FREE marks the bits free in bitmaps. The BEGIN and END blocks must be in extent boundary and be extent multiple; otherwise, an error is raised.

151.4.12 TABLESPACE_FIX_SEGMENT_STATES Procedure

This procedure fixes the state of the segments in a tablespace in which migration was aborted.

During tablespace migration to or from local, the segments are put in a transient state. If migration is aborted, then the segment states are corrected by SMON when event 10906 is set. A database with segments in such a transient state cannot be downgraded. The procedure can be used to fix the state of such segments.

Syntax

DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES (
    tablespace_name IN VARCHAR);

Parameters

Table 151-14  TABLESPACE_FIX_SEGMENT_STATES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of the tablespace whose segments must be fixed</td>
</tr>
</tbody>
</table>
Usage Notes
The tablespace must be kept online and read/write when this procedure is called.

Examples
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES('TS1')

151.4.13 TABLESPACE_MIGRATE_FROM_LOCAL Procedure
This procedure migrates a locally managed tablespace to a dictionary-managed tablespace.

Syntax
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL (
    tablespace_name IN VARCHAR2);

Parameter

Table 151-15  TABLESPACE_MIGRATE_FROM_LOCAL Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace</td>
</tr>
</tbody>
</table>

Usage Notes
The tablespace must be kept online and read/write during migration. Migration of temporary tablespaces and migration of SYSTEM tablespaces are not supported.

Examples
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('USERS');

151.4.14 TABLESPACE_MIGRATE_TO_LOCAL Procedure
This procedure migrates the tablespace from a dictionary-managed format to a locally managed format. Tablespaces migrated to locally managed format are user managed.

Syntax
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL (
    tablespace_name IN VARCHAR2,
    unit_size IN POSITIVE DEFAULT NULL,
    rfno IN POSITIVE DEFAULT NULL);

Parameters

Table 151-16  TABLESPACE_MIGRATE_TO_LOCAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of the tablespace to be migrated</td>
</tr>
</tbody>
</table>
Table 151-16 (Cont.) TABLESPACE_MIGRATE_TO_LOCAL Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit_size</td>
<td>Bitmap unit size (which is the size of the smallest possible chunk of space that can be allocated) in the tablespace specified in number of blocks</td>
</tr>
<tr>
<td>rfno</td>
<td>Relative File Number of the file where the bitmap blocks are placed</td>
</tr>
</tbody>
</table>

Usage Notes

- Before you migrate the SYSTEM tablespace, migrate any dictionary-managed tablespaces that you want to use in read/write mode to locally managed. After the SYSTEM tablespace is migrated, you cannot change dictionary-managed tablespaces to read/write.

See Also:
Oracle Database Administrator's Guide

- The tablespace must be kept online and read/write during migration. Note that temporary tablespaces cannot be migrated.
- Allocation Unit may be specified optionally. The default is calculated by the system based on the highest common divisor of all extents (used or free) for the tablespace. This number is further trimmed based on the MINIMUM EXTENT for the tablespace (5 if MINIMUM EXTENT is not specified). Thus, the calculated value will not be larger than the MINIMUM EXTENT for the tablespace. The last free extent in every file is ignored for GCD calculation. If you specify the unit size, then it must be a factor of the unit_size calculated by the system; otherwise an error message is returned.
- The Relative File Number parameter is used to place the bitmaps in a desired file. If space is not found in the file, then an error is issued. The data file specified must be part of the tablespace being migrated. If the dataflow is not specified, then the system chooses a dataflow in which to place the initial bitmap blocks. If space is not found for the initial bitmaps, then an error is raised.

Examples

To migrate a tablespace 'TS1' in 2KB blocksize with minimum extent size 1MB:

```sql
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('TS1', 512, 2);
```

The bitmaps are placed in file with relative file number 2.

151.4.15 TABLESPACE_REBUILD_BITMAPS Procedure

This procedure rebuilds the appropriate bitmaps. If no bitmap block is specified, then it rebuilds all bitmaps for the given tablespace.

The procedure cannot be used on the SYSTEM tablespace.
Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS (  
tablespace_name         IN    VARCHAR2,
bitmap_relative_file    IN    POSITIVE   DEFAULT NULL,
bitmap_block            IN    POSITIVE   DEFAULT NULL);
```

Parameters

Table 151-17 TABLESPACE_REBUILD_BITMAPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace</td>
</tr>
<tr>
<td>bitmap_relative_file</td>
<td>Relative file number of bitmap block to rebuild</td>
</tr>
<tr>
<td>bitmap_block</td>
<td>Block number of bitmap block to rebuild</td>
</tr>
</tbody>
</table>

Usage Notes

Only full rebuild is supported.

Examples

The following example rebuilds bitmaps for all the files in the USERS tablespace.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('USERS');
```

151.4.16 TABLESPACE_REBUILD_QUOTAS Procedure

This procedure rebuilds quotas for the given tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS (  
tablespace_name         IN    VARCHAR2);
```

Parameters

Table 151-18 TABLESPACE_REBUILD_QUOTAS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace</td>
</tr>
</tbody>
</table>

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS('USERS');
```
151.4.17 TABLESPACE_RELOCATE_BITMAPS Procedure

This procedure relocates the bitmaps to the destination specified.

Syntax

DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS (  
  tablespace_name IN VARCHAR2,  
  filno IN POSITIVE,  
  blkno IN POSITIVE);  

Parameters

Table 151-19  TABLESPACE_RELOCATE_BITMAPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace</td>
</tr>
<tr>
<td>filno</td>
<td>Relative File Number of the destination file</td>
</tr>
<tr>
<td>blkno</td>
<td>Block Number of the destination range</td>
</tr>
</tbody>
</table>

Usage Notes

- Migration of a tablespace from dictionary-managed to locally managed format could result in the creation of SPACE HEADER segment that contains the bitmap blocks. The SPACE HEADER segment is treated as user data. If you explicitly resize a file at or below the space header segment, then an error is issued. Use the TABLESPACE_RELOCATE_BITMAPS command to move the control information to a different destination and then resize the file.
- This procedure cannot be used on the SYSTEM tablespace.
- The tablespace must be kept online and read/write during relocation of bitmaps. This can be done only on migrated locally managed tablespaces.

Examples

EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS('TS1', 3, 4);

Moves the bitmaps to file 3, block 4.

Note:

The source and the destination addresses must not overlap. The destination block number is rounded down to the unit boundary. If there is user data in that location, then an error is raised.
151.4.18 TABLESPACE_VERIFY Procedure

This procedure verifies that the Bitmaps and extent maps for the segments in the tablespace are synchronized.

Syntax

```sql
DBMS_SPACE_ADMIN.TABLESPACE_VERIFY ( 
    tablespace_name         IN    VARCHAR2, 
    verify_option           IN    POSITIVE DEFAULT TABLESPACE_VERIFY_BITMAP);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace</td>
</tr>
<tr>
<td>verify_option</td>
<td>One option is supported: TABLESPACE_VERIFY_BITMAP</td>
</tr>
</tbody>
</table>

Examples

```sql
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_VERIFY('USERS');
```
The `DBMS_SPD` package provides subprograms for managing SQL plan directives (SPD).

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Views**
- **Summary of DBMS_SPD Subprograms**

### See Also:

- [DBMS_STATS](#)
- *Oracle Database SQL Tuning Guide* regarding SQL plan directives

## 152.1 DBMS_SPD Overview

This package provides subprograms for managing SQL plan directives (SPD).

SPD are objects generated automatically by Oracle. For example, if Oracle detects that the single table cardinality estimated made by the optimizer is different from the actual number of rows returned when accessing the table, it will automatically create a directive to perform dynamic statistics for the table. When any SQL statement referencing the table is compiled, the optimizer will perform dynamic statistics for the table to get a more accurate estimate.

## 152.2 DBMS_SPD Security Model

`DBMS_SPD` is an invoker-rights package. The invoker requires `ADMINISTER SQL MANAGEMENT OBJECT` privilege for executing most of the subprograms in this package. Also, the subprograms commit the current transaction (if any), perform the operation, and then commit it again.

## 152.3 DBMS_SPD Views

The DBA view `DBA_SQL_PLAN_DIRECTIVES` shows all the directives created in the system and the view `DBA_SQL_PLAN_DIR_OBJECTS` displays the objects that are included in the directives.

## 152.4 Summary of DBMS_SPD Subprograms

This table lists and briefly describes the `DBMS_SPD` package subprograms.
Table 152-1  DBMS_SPD Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_SQL_PLAN_DIRECTIVE Procedure</td>
<td>Changes different attributes of a SQL plan directive</td>
</tr>
<tr>
<td>CREATE_STGTAB_DIRECTIVE Procedure</td>
<td>Creates a staging table into which to pack (export) SQL plan directives.</td>
</tr>
<tr>
<td>DROP_SQL_PLAN_DIRECTIVE Procedure</td>
<td>Drops a SQL plan directive</td>
</tr>
<tr>
<td>FLUSH_SQL_PLAN_DIRECTIVE Procedure</td>
<td>Allows for manual flushing of the SQL plan directives that are automatically recorded in SGA memory while executing SQL statements.</td>
</tr>
<tr>
<td>GET_PREFS Function</td>
<td>Gets the values for preferences for SQL plan directives</td>
</tr>
<tr>
<td>PACK_STGTAB_DIRECTIVE Function</td>
<td>Packs (exports) SQL plan directives into a staging table.</td>
</tr>
<tr>
<td>SET_PREFS Procedure</td>
<td>Allows the setting of different preferences for SQL plan directives</td>
</tr>
<tr>
<td>UNPACK_STGTAB_DIRECTIVE Function</td>
<td>Unpacks (imports) SQL plan directives from a staging table.</td>
</tr>
</tbody>
</table>

152.4.1 ALTER_SQL_PLAN_DIRECTIVE Procedure

This procedure changes different attributes of a SQL plan directive.

Syntax

```sql
DBMS_SPD.ALTER_SQL_PLAN_DIRECTIVE ( 
    directive_id        IN     NUMBER, 
    attribute_name      IN     VARCHAR2, 
    attribute_value     IN     VARCHAR2);
```

Parameters

Table 152-2  ALTER_SQL_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directive_id</td>
<td>SQL plan directive ID</td>
</tr>
<tr>
<td>attribute_name</td>
<td>• ENABLED</td>
</tr>
<tr>
<td></td>
<td>• AUTO_DROP</td>
</tr>
<tr>
<td>attribute_value</td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td>• ENABLED:</td>
</tr>
<tr>
<td></td>
<td>• If YES directive is enabled and may be used</td>
</tr>
<tr>
<td></td>
<td>• If NO directive is not enabled and will not be used</td>
</tr>
<tr>
<td></td>
<td>• AUTO_DROP:</td>
</tr>
<tr>
<td></td>
<td>• If YES directive will be dropped automatically if not used for SPD_RETENTION_WEEKS. This is the default behavior.</td>
</tr>
<tr>
<td></td>
<td>• If NO directive will not be dropped automatically</td>
</tr>
</tbody>
</table>
Exceptions

- ORA-38171 INSUFFICIENT_PRIVILEGE: The user does not have proper privilege to perform the operation.
- ORA-28104 INVALID_INPUT: The input value is not valid.
- ORA-13158 OBJECT_DOES_NOT_EXIST: The specified object does not exist.

Usage Notes

The ADMINISTER SQL MANAGEMENT OBJECT privilege is required to execute this procedure.

Examples

```sql
BEGIN
  DBMS_SPD.ALTER_SQL_PLAN_DIRECTIVE (12345, 'STATE', 'PERMANENT');
END;
```

152.4.2 CREATE_STGTAB_DIRECTIVE Procedure

This procedure creates a staging table into which to pack (export) SQL plan directives.

Syntax

```sql
DBMS_SPD.CREATE_STGTAB_DIRECTIVE (    
  table_name IN VARCHAR2, 
  table_owner IN VARCHAR2 := USER, 
  tablespace_name IN VARCHAR2 := NULL); 
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of staging table</td>
</tr>
<tr>
<td>table_owner</td>
<td>Name of schema owner of staging table. Default is current schema.</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace. Default NULL means create staging table in the default tablespace:</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-38171 INSUFFICIENT_PRIVILEGE: The user does not have proper privilege to perform the operation.
- ORA-28104 INVALID_INPUT: The input value is not valid.
- ORA-44001 INVALID_SCHEMA: The input schema does not exist.
- ORA-13159 TABLE_ALREADY_EXISTS: The specified table already exists.
- ORA-29304 TABLESPACE_MISSING: The specified tablespace does not exist.

Usage Notes

The ADMINISTER SQL MANAGEMENT OBJECT privilege is required to execute this procedure.
152.4.3 DROP_SQL_PLAN_DIRECTIVE Procedure

This procedure drops a SQL plan directive.

Syntax

```
DBMS_SPD.DROP_SQL_PLAN_DIRECTIVE (    
    directive_id      IN     NUMBER);
```

Parameters

Table 152-4  DROP_SQL_PLAN_DIRECTIVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directive_id</td>
<td>SQL plan directive ID</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-38171 INSUFFICIENT_PRIVILEGE: The user does not have proper privilege to perform the operation.
- ORA-28104 INVALID_INPUT: The input value is not valid.
- ORA-13158 OBJECT DOES NOT_EXIST: The specified object does not exist.

Usage Notes

The ADMINISTER SQL MANAGEMENT OBJECT privilege is required to execute this procedure.

Examples

```
BEGIN    
    DBMS_SPD.DROP_SQL_PLAN_DIRECTIVE (12345);    
END;
```

152.4.4 FLUSH_SQL_PLAN_DIRECTIVE Procedure

This procedure allows for manual flushing of the SQL plan directives that are automatically recorded in SGA memory while executing SQL statements.

The information recorded in the SGA is periodically flushed by an Oracle background process. This procedure provides a way to flush the information manually.

Syntax

```
DBMS_SPD.FLUSH_SQL_PLAN_DIRECTIVE;    
```

Exceptions

ORA-38171 INSUFFICIENT_PRIVILEGE: The user does not have proper privilege to perform the operation.
Usage Notes

The **ADMINISTER SQL MANAGEMENT OBJECT** privilege is required to execute this procedure.

Examples

```
BEGIN
  DBMS_SPD.FLUSH_SQL_PLAN_DIRECTIVE;
END;
```

### 152.4.5 GET_PREFS Function

This function returns the value for the specified preferences for SQL plan directives.

Syntax

```
DBMS_SPD.GET_PREFS (  
    pname     IN   VARCHAR2)  
RETURN VARCHAR2;
```

Parameters

**Table 152-5  ** *GET_PREFS Function Parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>Preference name. The procedure supports the preference SPD_RETENTION_WEEKS.</td>
</tr>
</tbody>
</table>

Return Values

Preference value

Exceptions

- **ORA-38171 INSUFFICIENT_PRIVILEGE**: The user does not have proper privilege to perform the operation.
- **ORA-28104 INVALID_INPUT**: The input value is not valid.

Usage Notes

- **The ADMINISTER SQL MANAGEMENT OBJECT** privilege is required to execute this procedure.
- **SPD_RETENTION_WEEKS** - SQL plan directives are purged if not used for more than the value set for this preference.

Examples

```
SELECT DBMS_SPD.GET_PREFS('SPD_RETENTION_WEEKS') FROM DUAL;
```
152.4.6 PACK_STGTAB_DIRECTIVE Function

This function packs (exports) SQL plan directives into a staging table.

Syntax

```sql
DBMS_SPD.PACK_STGTAB_DIRECTIVE (
    table_name     IN VARCHAR2,
    table_owner    IN VARCHAR2  := USER,
    directive_id   IN NUMBER    := NULL,
    obj_list       IN OBJECTTAB := NULL)
RETURN NUMBER
```

Parameters

Table 152-6  PACK_STGTAB_DIRECTIVE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of staging table</td>
</tr>
<tr>
<td>table_owner</td>
<td>Name of schema owner of staging table. Default is current schema.</td>
</tr>
<tr>
<td>directive_id</td>
<td>SQL plan directive ID. Default NULL means all directives in the system.</td>
</tr>
<tr>
<td>obj_list</td>
<td>Used to filter the directives to be packed based on the objects used in directives. If obj_list is not NULL, a directive is packed only if all the objects in the directive exist in obj_list.</td>
</tr>
</tbody>
</table>

Return Values

Number of SQL plan directives packed.

Exceptions

- ORA-38171 INSUFFICIENT_PRIVILEGE: The user does not have proper privilege to perform the operation.
- ORA-28104 INVALID_INPUT: The input value is not valid.
- ORA-44001 INVALID_SCHEMA: The input schema does not exist.
- ORA-29304 INVALID_STGTAB: The specified staging table is invalid or does not exist.
- ORA-13158 OBJECT_DOES_NOT_EXIST: The specified object does not exist.

Usage Notes

The ADMINISTER SQL MANAGEMENT OBJECT privilege is required to execute this procedure.

Examples

```
-- Pack all directives in the system
SELECT DBMS_SPD.PACK_STGTAB_DIRECTIVE('mydirtab') FROM DUAL;
```

SET SERVEROUTPUT ON;
-- Pack directives relevant to objects in SH schema
DECLARE
    my_list  DBMS_SPD.OBJECTTAB := DBMS_SPD.ObjectTab();
    dir_cnt  NUMBER;
BEGIN
    my_list.extend(1);
    my_list(1).owner := 'SH';           -- schema name
    my_list(1).object_name := NULL;     -- all tables in SH
    my_list(1).object_type := 'TABLE';  -- type of object
    dir_cnt :=
        DBMS_SPD.PACK_STGTAB_DIRECTIVE('mydirtab', obj_list => my_list);
    DBMS_OUTPUT.PUT_LINE('dir_cnt = ' || dir_cnt);
END;

-- Pack directives relevant to tables SALES and CUSTOMERS in SH schema
DECLARE
    my_list  DBMS_SPD.OBJECTTAB := DBMS_SPD.ObjectTab();
    dir_cnt  NUMBER;
BEGIN
    my_list.extend(2);
    -- SALES table
    my_list(1).owner := 'SH';
    my_list(1).object_name := 'SALES';
    my_list(1).object_type := 'TABLE';

    -- CUSTOMERS table
    my_list(2).owner := 'SH';
    my_list(2).object_name := 'CUSTOMERS';
    my_list(2).object_type := 'TABLE';

    dir_cnt :=
        DBMS_SPD.PACK_STGTAB_DIRECTIVE('mydirtab', obj_list => my_list);
    DBMS_OUTPUT.PUT_LINE('dir_cnt = ' || dir_cnt);
END;

152.4.7 SET_PREFS Procedure

This procedure allows the setting of different preferences for SQL plan directives.

Syntax

DBMS_SPD.SET_PREFS (
    pname     IN   VARCHAR2,
    pvalue    IN   VARCHAR2);

Parameters

Table 152-7  SET_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>Preference name. The procedure supports the preference SPD_RETENTION_WEEKS.</td>
</tr>
</tbody>
</table>
Table 152-7  (Cont.) SET_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| pvalue     | Preference value.  
  - SPD_RETENTION_WEEKS: SQL plan directives are purged if not used for more than the value set for this preference. Default is 53 (SPD_RETENTION_WEEKS_DEFAULT) weeks, which means a directive is purged if it has been left unused for little over a year. It can be set to any value greater than or equal to 0. Also value NULL can be passed to set the preference to default. |

Exceptions

- ORA-38171 INSUFFICIENT_PRIVILEGE: The user does not have proper privilege to perform the operation.
- ORA-28104 INVALID_INPUT: The input value is not valid.

Usage Notes

- The ADMINISTER SQL MANAGEMENT OBJECT privilege is required to execute this procedure.
- SPD_RETENTION_WEEKS - SQL plan directives are purged if not used for more than the value set for this preference.

Examples

BEGIN  
  DBMS_SPD.SET_PREFS('SPD_RETENTION_WEEKS', '4');  
END;

152.4.8 UNPACK_STGTAB_DIRECTIVE Function

This procedure unpacks (imports) SQL plan directives from a staging table.

Syntax

DBMS_SPD.UNPACK_STGTAB_DIRECTIVE (  
  table_name         IN VARCHAR2,  
  table_owner        IN VARCHAR2 := USER,  
  directive_id       IN NUMBER := NULL,  
  obj_list           IN OBJECTTAB := NULL)  
RETURN NUMBER

Parameters

Table 152-8  UNPACK_STGTAB_DIRECTIVE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of staging table</td>
</tr>
<tr>
<td>table_owner</td>
<td>Name of schema owner of staging table. Default is current schema.</td>
</tr>
</tbody>
</table>
Table 152-8  (Cont.) UNPACK_STGTAB_DIRECTIVE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directive_id</td>
<td>SQL plan directive ID. Default NULL means all directives in the system.</td>
</tr>
<tr>
<td>obj_list</td>
<td>Used to filter the directives to be unpacked based on the objects used in directives. If obj_list is not NULL, a directive is unpacked only if all the objects in the directive exist in obj_list.</td>
</tr>
</tbody>
</table>

Return Values
Number of SQL plan directives unpacked.

Exceptions
- ORA-38171 INSUFFICIENT_PRIVILEGE: The user does not have proper privilege to perform the operation.
- ORA-28104 INVALID_INPUT: The input value is not valid.
- ORA-44001 INVALID_SCHEMA: The input schema does not exist.
- ORA-29304 INVALID_STGTAB: The specified staging table is invalid or does not exist.
- ORA-13158 OBJECT_DOES_NOT_EXIST: The specified object does not exist.

Usage Notes
The ADMINISTER SQL MANAGEMENT OBJECT privilege is required to execute this procedure.

Examples
-- Unpack all directives in the staging table
SELECT DBMS_SPD.UNPACK_STGTAB_DIRECTIVE('mydirtab') FROM DUAL;

SET SERVEROUTPUT ON;
-- Unpack directives relevant to objects in SH schema
DECLARE
  my_list  DBMS_SPD.OBJECTTAB := DBMS_SPD.ObjectTab();
  dir_cnt  number;
BEGIN
  my_list.extend(1);
  my_list(1).owner := 'SH';           -- schema name
  my_list(1).object_name := null;     -- all tables in SH
  my_list(1).object_type := 'TABLE';  -- type of object
  dir_cnt :=
    DBMS_SPD.UNPACK_STGTAB_DIRECTIVE('mydirtab', obj_list => my_list);
  DBMS_OUTPUT.PUT_LINE('dir_cnt = ' || dir_cnt);
END;

-- Unpack directives relevant to tables SALES and CUSTOMERS in SH schema
DECLARE
  my_list  DBMS_SPD.OBJECTTAB := DBMS_SPD.ObjectTab();
  dir_cnt  NUMBER;
BEGIN
  my_list.extend(2);
  dir_cnt :=
    DBMS_SPD.UNPACK_STGTAB_DIRECTIVE('mydirtab', obj_list => my_list);
  DBMS_OUTPUT.PUT_LINE('dir_cnt = ' || dir_cnt);
END;
-- SALES table
my_list(1).owner := 'SH';
my_list(1).object_name := 'SALES';
my_list(1).object_type := 'TABLE';

-- CUSTOMERS table
my_list(2).owner := 'SH';
my_list(2).object_name := 'CUSTOMERS';
my_list(2).object_type := 'TABLE';

dir_cnt :=
  DBMS_SPD.UNPACK_STGTAB_DIRECTIVE('mydirtab', obj_list => my_list);
DBMS_OUTPUT.PUT_LINE('dir_cnt = ' || dir_cnt);
END;
The `DBMS_SPM` package supports the SQL plan management feature by providing an interface for the DBA or other user to perform controlled manipulation of plan history and SQL plan baselines maintained for various SQL statements.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Examples
- Data Structures
- Summary of DBMS_SPM Subprograms

**See Also:**
For more information about “Using SQL Plan Management” in the *Oracle Database SQL Tuning Guide*

### 153.1 DBMS_SPM Overview

The `DBMS_SPM` package allows the user to manage SQL execution plans using SQL plan management.

SQL plan management prevents performance regressions resulting from sudden changes to the execution plan of a SQL statement by recording and evaluating the execution plans of SQL statements over time, and builds SQL plan baselines composed of a set of existing plans known to be efficient. The SQL plan baselines are then used to preserve performance of corresponding SQL statements, regardless of changes occurring in the system. Common usage scenarios where SQL plan management can improve or preserve SQL performance include:

- A database upgrade that installs a new optimizer version usually results in plan changes for a small percentage of SQL statements, with most of the plan changes resulting in either no performance change or improvement. However, certain plan changes may cause performance regressions. The use of SQL plan baselines significantly minimizes potential performance regressions resulting from a database upgrade.

- Ongoing system and data changes can impact plans for some SQL statements, potentially causing performance regressions. The use of SQL plan baselines helps to minimize performance regressions and stabilize SQL performance.

- Deployment of new application modules means introducing new SQL statements into the system. The application software may use appropriate SQL execution plans developed under a standard test configuration for the new SQL statements. If the system production
configuration differs significantly from the test configuration, SQL plan baselines can be evolved over time to produce better performance.

## 153.2 DBMS_SPM Security Model

The package is owned by SYS. The EXECUTE package privilege is required to execute its procedures. Any user granted the ADMINISTER SQL MANAGEMENT OBJECT privilege is able to execute the DBMS_SPM package.

## 153.3 DBMS_SPM Constants

The DBMS_SPM package provides constants that can be used for specifying parameter values.

These are shown in the following table. These constants are defined as standard input for the `time_limit` parameter of the `EVOLVE_SQL_PLAN_BASELINE` Function.

### Table 153-1 DBMS_SPM Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_LIMIT</td>
<td>INTEGER</td>
<td>2147483647</td>
<td>Oracle determines the appropriate time spent by the <code>EVOLVE_SQL_PLAN_BASELINE</code> Function.</td>
</tr>
<tr>
<td>NO_LIMIT</td>
<td>INTEGER</td>
<td>2147483647 -1</td>
<td>There is no limit to the time spent by the <code>EVOLVE_SQL_PLAN_BASELINE</code> Function.</td>
</tr>
</tbody>
</table>

## 153.4 DBMS_SPM Examples

These examples will help you understand use of DBMS_SPM.

Detailed examples are located under the following topics:

- Migrating Stored Outlines to SQL Plan Baselines
- Migrating Outlines to Utilize SQL Plan Management Features
- Migrating Outlines to Preserve Stored Outline Behavior
- Performing Follow-Up Tasks After Stored Outline Migration

## 153.5 DBMS_SPM Data Structures

The DBMS_SPM package defines a TABLE type.

### Table Types

- NAMELIST Table Type
153.5.1 DBMS_SPM NAMELIST Table Type

This type allows for a list of names as an input parameter.

**Syntax**

```
TYPE name_list IS TABLE OF VARCHAR2(30);
```

153.6 Summary of DBMS_SPM Subprograms

This table lists and briefly describes the DBMS_SPM package subprograms.

**Table 153-2  DBMS_SPM Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT_SQL_PLAN_BASELINE Procedure</td>
<td>Accepts a plan based on the recommendation of an evolve task</td>
</tr>
<tr>
<td>ALTER_SQL_PLAN_BASELINE Function</td>
<td>Changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format</td>
</tr>
<tr>
<td>CANCEL_EVOLVE_TASK Procedure</td>
<td>Cancels a currently executing evolve task</td>
</tr>
<tr>
<td>CONFIGURE Procedure</td>
<td>Sets configuration options for SQL management base, in parameter/value format</td>
</tr>
<tr>
<td>CREATE_EVOLVE_TASK Function</td>
<td>Creates an advisor task and sets its parameters</td>
</tr>
<tr>
<td>CREATE_STMTAB_BASELINE Procedure</td>
<td>Creates a staging table that used for transporting SQL plan baselines from one system to another</td>
</tr>
<tr>
<td>DROP_EVOLVE_TASK Procedure</td>
<td>Drops an evolved task</td>
</tr>
<tr>
<td>DROP_SQL_PLAN_BASELINE Function</td>
<td>Drops a single plan, or all plans associated with a SQL statement</td>
</tr>
<tr>
<td>EVOLVE_SQL_PLAN_BASELINE Function</td>
<td>Evolves SQL plan baselines associated with one or more SQL statements</td>
</tr>
<tr>
<td>EXECUTE_EVOLVE_TASK Function</td>
<td>Executes a previously created evolve task</td>
</tr>
<tr>
<td>IMPLEMENT_EVOLVE_TASK K Function</td>
<td>Implements a plan based on the recommendation of an evolve task</td>
</tr>
<tr>
<td>INTERRUPT_EVOLVE_TASK K Procedure</td>
<td>Interrupts a currently executing evolve task</td>
</tr>
<tr>
<td>LOAD_PLANS_FROM_CURSOR_CACHE Functions</td>
<td>Loads one or more plans present in the cursor cache for a SQL statement</td>
</tr>
<tr>
<td>LOAD_PLANS_FROM_AWR Function</td>
<td>Loads the SQL Management Base (SMB) with SQL plan baselines for a set of SQL statements using the plans from the AWR, and returns the number of plans loaded</td>
</tr>
<tr>
<td>LOAD_PLANS_FROM_SQLSET Function</td>
<td>Loads plans stored in a SQL tuning set (STS) into SQL plan baselines</td>
</tr>
<tr>
<td>MIGRATE_STORED_OUTLINE Functions</td>
<td>Migrates existing stored outlines to SQL plan baselines</td>
</tr>
</tbody>
</table>
Table 153-2  (Cont.) DBMS_SPM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACK_STGTAB_BASELINE</td>
<td>Packs (exports) SQL plan baselines from SQL management base into a staging table</td>
</tr>
<tr>
<td>Procedure</td>
<td>RESET_EVOLVE_TASK Procedure Resets an evolve task to its initial state</td>
</tr>
<tr>
<td></td>
<td>RESUME_EVOLVE_TASK Procedure Resumes a previously interrupted task</td>
</tr>
<tr>
<td></td>
<td>REPORT_AUTO_EVOLVE_TASK Function Displays the results of an automatic evolve task.</td>
</tr>
<tr>
<td></td>
<td>REPORT_EVOLVE_TASK Function Displays the results of an evolved task</td>
</tr>
<tr>
<td></td>
<td>SET_EVOLVE_TASK_PARAMETER Procedure Sets a parameter of an evolve task</td>
</tr>
<tr>
<td></td>
<td>UNPACK_STGTAB_BASELINE Function Unpacks (imports) SQL plan baselines from a staging table into SQL management base</td>
</tr>
</tbody>
</table>

153.6.1 ACCEPT_SQL_PLAN_BASELINE Procedure

The procedure accepts a plan based on the recommendation of an evolve task.

Syntax

```sql
DBMS_SPM.ACCEPT_SQL_PLAN_BASELINE  (
    task_name       IN  VARCHAR2,
    object_id       IN  NUMBER    := NULL,
    task_owner      IN  VARCHAR2  := NULL,
    force           IN  BOOLEAN   := FALSE);
```

Parameters

Table 153-3  ACCEPT_SQL_PLAN_BASELINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to implement</td>
</tr>
<tr>
<td>object_id</td>
<td>Identifier of the advisor framework object that represents a single plan. If NULL, the report is generated for all objects.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Owner of the evolve task. Defaults to the current schema owner.</td>
</tr>
<tr>
<td>force</td>
<td>Accept the plan even if the advisor did not recommend such an action. The default is FALSE requiring acceptance of the plan only if the plan is verified and shows sufficient improvement in benefit.</td>
</tr>
</tbody>
</table>
153.6.2 ALTER_SQL_PLAN_BASELINE Function

This function changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format.

Syntax

```
DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
    sql_handle        IN VARCHAR2 := NULL,
    plan_name         IN VARCHAR2 := NULL,
    attribute_name    IN VARCHAR2,
    attribute_value   IN VARCHAR2)
RETURN PLS_INTEGER;
```

Parameters

Table 153-4 ALTER_SQL_PLAN_BASELINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_handle</td>
<td>SQL statement handle. It identifies plans associated with a SQL statement for an attribute change. If NULL then plan_name must be specified.</td>
</tr>
<tr>
<td>plan_name</td>
<td>Plan name. It identifies a specific plan. Default NULL means set the attribute for all plans associated with a SQL statement identified by sql_handle. If NULL then sql_handle must be specified.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Name of plan attribute to set (see table below).</td>
</tr>
<tr>
<td>attribute_value</td>
<td>Value of plan attribute to use (see table below)</td>
</tr>
</tbody>
</table>

Table 153-5 Names & Values for ALTER_SQL_PLAN_BASELINE Function Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>'YES' means the plan is available for use by the optimizer. It may or may not be used depending on accepted status.</td>
<td>'YES' or 'NO'</td>
</tr>
<tr>
<td>fixed</td>
<td>'YES' means the SQL plan baseline is not evolved over time. A fixed plan takes precedence over a non-fixed plan.</td>
<td>'YES' or 'NO'</td>
</tr>
<tr>
<td>autopurge</td>
<td>'YES' means the plan is purged if it is not used for a time period. 'NO' means it is never purged.</td>
<td>'YES' or 'NO'</td>
</tr>
<tr>
<td>plan_name</td>
<td>Name of the plan</td>
<td>String of up to 30 characters</td>
</tr>
<tr>
<td>description</td>
<td>Plan description.</td>
<td>String of up to 500 bytes</td>
</tr>
</tbody>
</table>

Return Values

The number of plans altered.
### Usage Notes

When a single plan is specified, one of various statuses, or plan name, or description can be altered. When all plans for a SQL statement are specified, one of various statuses, or description can be altered. This function can be called numerous times, each time setting a different plan attribute of same plan(s) or different plan(s).

### 153.6.3 CANCEL_EVOLVE_TASK Procedure

The procedure cancels a currently executing evolve task. All intermediate results are removed from the task.

**Syntax**

```sql
DBMS_SPM.CANCEL_EVOLVE_TASK  (
    task_name        IN  VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to cancel</td>
</tr>
</tbody>
</table>

### 153.6.4 CONFIGURE Procedure

This procedure sets configuration options for the SQL management base and for the maintenance of SQL plan baselines. You can call this function multiple times, setting a different configuration option each time.

**Syntax**

```sql
DBMS_SPM.CONFIGURE  (
    parameter_name    IN VARCHAR2,
    parameter_value   IN VARCHAR2 := NULL,
    allow             IN BOOLEAN := TRUE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter_name</td>
<td>Name of parameter to set (see table below).</td>
</tr>
<tr>
<td>parameter_value</td>
<td>Value of parameter to use (see table below). The maximum length of parameter_value is 1000 characters.</td>
</tr>
<tr>
<td>allow</td>
<td>Whether to include (true) or exclude (false) matching SQL statements and plans for the auto_capture_* parameters. If null, then the procedure ignores the specified parameter.</td>
</tr>
</tbody>
</table>
### Table 153-8  Names and Values for CONFIGURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_capture_action</td>
<td>Action to include (=) or exclude (&lt;&gt;) for SQL plan management automatic capture, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely. The database only uses this filter when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is TRUE.</td>
<td>Action name, for example, R%</td>
<td></td>
</tr>
<tr>
<td>auto_capture_module</td>
<td>Module to include (=) or exclude (&lt;&gt;) for SQL plan management auto capture, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely. The database only uses this filter when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is TRUE.</td>
<td>Module name, for example, LOGGER</td>
<td></td>
</tr>
<tr>
<td>auto_capture_parsing_schema_name</td>
<td>Parsing schema to include (=) or exclude (&lt;&gt;) for SQL plan management auto capture, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely. The database only uses this filter when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is TRUE.</td>
<td>Schema name, for example, HR</td>
<td></td>
</tr>
<tr>
<td>auto_capture_sql_text</td>
<td>Search pattern to apply to SQL text of LIKE or NOT LIKE, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely. The database only uses this filter when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is TRUE.</td>
<td>Text of a SQL statement, for example, SELECT a%</td>
<td></td>
</tr>
<tr>
<td>plan_retention_weeks</td>
<td>Number of weeks to retain unused plans before the database purges them. A null value resets to the default value of 53 weeks, or 1 year plus 1 week. (This retains plans for annually executing queries.) The value of allow is ignored.</td>
<td>5–523</td>
<td>53</td>
</tr>
</tbody>
</table>
Table 153-8  (Cont.) Names and Values for CONFIGURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>space_budget_percent</td>
<td>Maximum percent of SYSAUX space that can be used for SQL management base.</td>
<td>1–50</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>The database issues alerts when this amount is exceeded. A null value resets</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the percentage to the default value of 10%. The value of allow is ignored.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exceptions

Table 153-9  CONFIGURE Exceptions

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-38133</td>
<td>Invalid parameter name</td>
</tr>
<tr>
<td>ORA-38134</td>
<td>Invalid parameter value</td>
</tr>
<tr>
<td>ORA-38150</td>
<td>Not enough space for new filter</td>
</tr>
<tr>
<td>ORA-38151</td>
<td>Module name too long</td>
</tr>
<tr>
<td>ORA-38152</td>
<td>Action name too long</td>
</tr>
<tr>
<td>ORA-38304</td>
<td>Missing or invalid user name</td>
</tr>
</tbody>
</table>

Usage Notes

- **When** parameter_name is auto_capture_sql_text, the parameter_value is an automatic search filter. The filter uses the search pattern of LIKE parameter_name when allow=>true. The filter uses the pattern NOT LIKE parameter_name when allow=>false.

  For all other non-null parameter_name values, the search pattern depends on the allow setting. The parameter uses an equal sign (=) when allow=>true. The parameter uses a not-equal sign (<> when allow=>false.

- You can configure multiple automatic capture parameters of different types. You cannot specify multiple values for the same parameter. Instead, the values specified for a particular parameter are combined. For example, specifying auto_capture_sql_text to be '%TABLE1%', TRUE, and '%TABLE2%', FALSE will result in matching SQL text LIKE '%TABLE1%' and NOT LIKE '%TABLE2%'. The database uses these configuration settings only when the initialization parameter OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is set to TRUE.

- A null value for parameter_value removes the filter for parameter_name entirely. By using parameter_value=>'%' in combination with allow=FALSE, you can filter out all values for a parameter, and then create a separate filter to include only specified values. The DBA_SQL_MANAGEMENT_CONFIG view shows the current filters.

- The default space budget for SQL management base is no more than ten percent of the size of SYSAUX tablespace. The space budget can be set to a maximum of 50%. The default unused plan retention period is one year and one week, which
means a plan will be automatically purged if it has not been used for more than a year. The retention period can be set to a maximum of 523 weeks (i.e. a little over 10 years).

- When the space occupied by SQL management base exceeds the defined space budget limit, a weekly database alert is generated.

Examples
The following example creates a filter for SQL text that is like `SELECT a%`:

EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT', 'select a%', 'TRUE');

The following example filters out the HR parsing schema:

EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_PARSING_SCHEMA_NAME', 'HR', 'FALSE');

The following example removes any existing filters for SQL text:

EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT', NULL, NULL);

The following example removes any LIKE or NOT LIKE filters for the SQL text `select a%`:

EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT', 'select a%', NULL);

The following example creates a filter with the predicate `(action LIKE 'R%') OR (action LIKE '%E_')`:

EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_ACTION', 'R%', 'TRUE');
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_ACTION', '%E_', 'TRUE');

The following example creates a filter with the predicate `NOT(module LIKE 'LOGGER') AND NOT(module LIKE 'UTIL__')`:

EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_MODULE', 'LOGGER', 'FALSE');
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_MODULE', 'UTIL__', 'FALSE');

153.6.5 CREATE_EVOLVE_TASK Function

The function has two overloads, both of which create an advisor task and sets its parameters. This version which takes a SQL handle creates an evolve task in order to evolve one or more plans for a given SQL statement.

Syntax

```sql
DBMS_SPM.CREATE_EVOLVE_TASK (    sql_handle    IN  VARCHAR2  := NULL,
    plan_name     IN  VARCHAR2  := NULL,
    time_limit    IN  NUMBER    := DBMS_SPM.AUTO_LIMIT,
    task_name     IN  VARCHAR2  := NULL,
    description   IN  VARCHAR2  := NULL)    RETURN VARCHAR2;
```
DBMS_SPM.CREATE_EVOLVE_TASK (  
  plan_list IN DBMS_SPM.NAME_LIST,  
  time_limit IN NUMBER := DBMS_SPM.AUTO_LIMIT,  
  task_name IN VARCHAR2 := NULL,  
  description IN VARCHAR2 := NULL)  
RETURN VARCHAR2;

Parameters

Table 153-10  CREATE_EVOLVE_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_handle</td>
<td>Handle of a SQL statement. The default NULL considers all SQL statements with non-accepted plans.</td>
</tr>
<tr>
<td>plan_list</td>
<td>List of plan names. The plans may belong to different SQL statements.</td>
</tr>
<tr>
<td>plan_name</td>
<td>Plan identifier. The default NULL considers all non-accepted plans of the specified SQL handle or all SQL statements if the SQL handle is NULL.</td>
</tr>
<tr>
<td>time_limit</td>
<td>Time limit in number of minutes. The time limit is global and it is used in the following manner. The time limit for first non-accepted plan is equal to the input value. The time limit for the second non-accepted plan is equal to (input value - time spent in first plan verification) and so on. The default DBMS_SPM.AUTO_LIMIT means let the system choose an appropriate time limit based on the number of plan verifications required to be done. The value DBMS_SPM.NO_LIMIT means no time limit.</td>
</tr>
<tr>
<td>task_name</td>
<td>Evolve task name</td>
</tr>
<tr>
<td>description</td>
<td>Description of the task (maximum 256 characters)</td>
</tr>
</tbody>
</table>

Return Values

SQL evolve task unique name

153.6.6 CREATE_STGTAB_BASELINE Procedure

This procedure creates a staging table used for transporting SQL plan baselines from one system to another.

Syntax

DBMS_SPM.CREATE_STGTAB_BASELINE (  
  table_name IN VARCHAR2,  
  table_owner IN VARCHAR2 := NULL,  
  tablespace_name IN VARCHAR2 := NULL);
Parameters

Table 153-11  CREATE_STGTAB_BASELINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of staging table to create for the purpose of packing and unpacking SQL plan baselines</td>
</tr>
<tr>
<td>table_owner</td>
<td>Name of owner of the staging table. Default NULL means current schema is the table owner.</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Name of tablespace. Default NULL means create staging table in the default tablespace.</td>
</tr>
</tbody>
</table>

Usage Notes

The creation of staging table is the first step. To migrate SQL plan baselines from one system to another, the user/DBA has to perform a series of steps as follows:

1. Create a staging table in the source system
2. Select SQL plan baselines in the source system and pack them into the staging table
3. Export staging table into a flat file using Oracle EXP utility or Data Pump
4. Transfer flat file to the target system
5. Import staging table from the flat file using Oracle IMP utility or Data Pump
6. Select SQL plan baselines from the staging table and unpack them into the target system

153.6.7 DROP_EVOLVE_TASK Procedure

The procedure drops an evolved task.

Syntax

```
DBMS_SPM.DROP_EVOLVE_TASK  (  
    task_name        IN  VARCHAR2);  
```

Parameters

Table 153-12  DROP_EVOLVE_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to drop</td>
</tr>
</tbody>
</table>

153.6.8 DROP_SQL_PLAN_BASELINE Function

This function drops a single plan, or all plans associated with a SQL statement.

Syntax

```
DBMS_SPM.DROP_SQL_PLAN_BASELINE (  
    sql_handle     IN VARCHAR2 := NULL,  
    plan_name      IN VARCHAR2 := NULL)  
RETURN PLS_INTEGER;  
```
Parameters

Table 153-13 DROP_SQL_PLAN_BASELINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_handle</td>
<td>SQL statement handle. It identifies plans associated with a SQL statement that are to be dropped. If NULL then plan_name must be specified.</td>
</tr>
<tr>
<td>plan_name</td>
<td>Plan name. It identifies a specific plan. Default NULL means to drop all plans associated with the SQL statement identified by sql_handle.</td>
</tr>
</tbody>
</table>

Return Values

The number of plans dropped

153.6.9 EVOLVE_SQL_PLAN_BASELINE Function

This function evolves SQL plan baselines associated with one or more SQL statements. A SQL plan baseline is evolved when one or more of its non-accepted plans is changed to an accepted plan or plans.

If interrogated by the user (parameter verify = 'YES'), the execution performance of each non-accepted plan is compared against the performance of a plan chosen from the associated SQL plan baseline. If the non-accepted plan performance is found to be better than SQL plan baseline performance, the non-accepted plan is changed to an accepted plan provided such action is permitted by the user (parameter commit = 'YES').

The second form of the function employs a plan list format.

Syntax

DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE (  
  sql_handle   IN VARCHAR2 := NULL,  
  plan_name    IN VARCHAR2 := NULL,  
  time_limit   IN INTEGER  := DBMS_SPM.AUTO_LIMIT,  
  verify       IN VARCHAR2 := 'YES',  
  commit       IN VARCHAR2 := 'YES')  
RETURN CLOB;

DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE (  
  plan_list    IN DBMS_SPM.NAME_LIST,  
  time_limit   IN INTEGER  := DBMS_SPM.AUTO_LIMIT,  
  verify       IN VARCHAR2 := 'YES',  
  commit       IN VARCHAR2 := 'YES')  
RETURN CLOB;
Parameters

Table 153-14  EVOLVE_SQL_PLAN_BASELINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_handle</td>
<td>SQL statement identifier. Unless plan_name is specified, NULL means to consider all statements with non-accepted plans in their SQL plan baselines.</td>
</tr>
<tr>
<td>plan_name</td>
<td>Plan identifier. Default NULL means to consider all non-accepted plans in the SQL plan baseline of either the identified SQL statement or all SQL statements if sql_handle is NULL.</td>
</tr>
<tr>
<td>plan_list</td>
<td>A list of plan names. Each plan in the list can belong to same or different SQL statement.</td>
</tr>
<tr>
<td>time_limit</td>
<td>Time limit in number of minutes. This applies only if verify = 'YES'. The time limit is global and it is used as follows: The time limit for first non-accepted plan verification is set equal to the input value; the time limit for second non-accepted plan verification is set equal to (input value - time spent in first plan verification); and so on. • DBMS_SPM.AUTO_LIMIT (Default) lets the system choose an appropriate time limit based on the number of plan verifications required to be done. • DBMS_SPM.NO_LIMIT means there is no time limit. • A positive integer value represents a user specified time limit.</td>
</tr>
<tr>
<td>verify</td>
<td>Specifies whether to execute the plans and compare the performance before changing non-accepted plans into accepted plans. A performance verification involves executing a non-accepted plan and a plan chosen from corresponding SQL plan baseline and comparing their performance statistics. If non-accepted plan shows performance improvement, it is changed to an accepted plan. • 'YES' (Default) - verifies that a non-accepted plan gives better performance before changing it to an accepted plan • 'NO' - directs not to execute plans but only to change non-accepted plans into accepted plans</td>
</tr>
<tr>
<td>commit</td>
<td>Specifies whether to update the ACCEPTED status of non-accepted plans from 'NO' to 'YES'. • 'YES' (Default) - perform updates of qualifying non-accepted plans and generate a report that shows the updates and the result of performance verification when verify = 'YES'. • 'NO' - generate a report without any updates. Note that commit = 'NO' together with verify = 'NO' represents a no-op.</td>
</tr>
</tbody>
</table>

Return Values

A CLOB containing a formatted text report showing non-accepted plans in sequence, each with a possible change of its ACCEPTED status, and if verify = 'YES' the result of their performance verification.

Usage Notes

Invoking this subprogram requires the ADMINISTER SQL MANAGEMENT OBJECT privilege.
153.6.10 EXECUTE_EVOLVE_TASK Function

The function executes a previously created evolve task.

Syntax

```sql
DBMS_SPM.EXECUTE_EVOLVE_TASK  (  
    task_name        IN  VARCHAR2,
    execution_name   IN  VARCHAR2  := NULL,
    execution_desc   IN  VARCHAR2  := NULL);  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Evolve task name</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name to qualify and identify an execution. If not specified, it is generated by the advisor and returned by the function.</td>
</tr>
<tr>
<td>execution_desc</td>
<td>Description of the execution (maximum 256 characters)</td>
</tr>
</tbody>
</table>

Return Values

Name of the new execution

153.6.11 IMPLEMENT_EVOLVE_TASK Function

The function implements all the actions recommended by an evolve task.

Syntax

```sql
DBMS_SPM.IMPLEMENT_EVOLVE_TASK  (  
    task_name       IN  VARCHAR2,
    task_owner      IN  VARCHAR2  := NULL,
    execution_name  IN  VARCHAR2  := NULL,
    force           IN BOOLEAN    := FALSE)  
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to report</td>
</tr>
<tr>
<td>task_owner</td>
<td>Owner of the evolve task. Defaults to the current schema owner.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name to qualify and identify an execution. If NULL, the action will be taken for the last task execution.</td>
</tr>
</tbody>
</table>
### Table 153-16 (Cont.) IMPLEMENT_EVOLVE_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>Accept all plans even if the advisor did not recommend such an action. The default is FALSE requiring acceptance of the plan only if the plan is verified and shows sufficient improvement in benefit.</td>
</tr>
</tbody>
</table>

#### Return Values

The number of plans accepted

### 153.6.12 INTERRUPT_EVOLVE_TASK Procedure

The procedure interrupts a currently executing evolve task. The task ends its operations as at a normal exit and the user can access the intermediate results. The task can be resumed later.

#### Syntax

```sql
DBMS_SPM.INTERRUPT_EVOLVE_TASK  (
    task_name        IN  VARCHAR2);
```

#### Parameters

Table 153-17 INTERRUPT_EVOLVE_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to interrupt</td>
</tr>
</tbody>
</table>

### 153.6.13 LOAD_PLANS_FROM_AWR Function

This function loads the SQL Management Base (SMB) with SQL plan baselines for a set of SQL statements using the plans from the AWR, and returns the number of plans loaded.

#### Syntax

```sql
DBMS_SPM.LOAD_PLANS_FROM_AWR
    begin_snap      IN  NUMBER,
    end_snap        IN  NUMBER,
    basic_filter    IN  VARCHAR2 := NULL,
    fixed           IN  VARCHAR2 := 'NO',
    enabled         IN  VARCHAR2 := 'YES',
    commit_rows     IN  NUMBER := 1000)
RETURN PLS_INTEGER;
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin_snap</td>
<td>Begin snapshot</td>
</tr>
<tr>
<td>end_snap</td>
<td>End snapshot</td>
</tr>
<tr>
<td>basic_filter</td>
<td>SQL predicate to filter the SQL from AWR.</td>
</tr>
<tr>
<td></td>
<td>NULL means all plans in AWR are selected.</td>
</tr>
<tr>
<td></td>
<td>Specifies the SQL predicate that filters the SQL from the shared SQL area</td>
</tr>
<tr>
<td></td>
<td>defined on attributes of the SQLSET_ROW.</td>
</tr>
<tr>
<td>fixed</td>
<td>Default 'NO' means the loaded plans will not change the current 'fixed'</td>
</tr>
<tr>
<td></td>
<td>property of the SQL plan baseline into which they are loaded.</td>
</tr>
<tr>
<td>enabled</td>
<td>Default 'YES' means the loaded plans will be considered by the optimizer</td>
</tr>
<tr>
<td>commit_rows</td>
<td>Number of SQL plans to load before doing a periodic commit.</td>
</tr>
<tr>
<td>dbid</td>
<td>The DBID that is used for imported or PDB-level AWR data.</td>
</tr>
</tbody>
</table>

Usage Notes

Requires the Administer SQL Management Object privilege

See Also:

For information on the SQLSET_ROW objects, see SQLSET_ROW Object Type.

153.6.14 LOAD_PLANS_FROM_CURSOR_CACHE Functions

This function loads one or more plans present in the cursor cache for a SQL
statement, or a set of SQL statements. It has four overloads: using SQL statement
text, using SQL handle, using SQL ID, or using attribute_name and attribute_value
pair.

Syntax

```sql
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (  
    sql_id            IN  VARCHAR2,  
    plan_hash_value   IN  NUMBER   := NULL,  
    sql_text          IN  CLOB,  
    fixed             IN  VARCHAR2 := 'NO',  
    enabled           IN  VARCHAR2 := 'YES')  
RETURN PLS_INTEGER;
```

```sql
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (  
    sql_id            IN  VARCHAR2,  
    plan_hash_value   IN  NUMBER   := NULL,  
```
### Parameters

**Table 153-19  LOAD_PLANS_FROM_CURSOR_CACHE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQL statement identifier. Identifies a SQL statement in the cursor cache. Note: In the third overload the text of identified SQL statement is extracted from cursor cache and is used to identify the SQL plan baseline into which the plan(s) are loaded. If the SQL plan baseline doesn't exist it is created.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Plan identifier. Default NULL means capture all plans present in the cursor cache for the SQL statement identified by SQL_ID.</td>
</tr>
<tr>
<td>sql_text</td>
<td>SQL text to use in identifying the SQL plan baseline into which the plans are loaded. If the SQL plan baseline does not exist, it is created. The use of text is crucial when the user tunes a SQL statement by adding hints to its text and then wants to load the resulting plan(s) into the SQL plan baseline of the original SQL statement.</td>
</tr>
<tr>
<td>sql_handle</td>
<td>SQL handle to use in identifying the SQL plan baseline into which the plans are loaded. The sql_handle must denote an existing SQL plan baseline. The use of handle is crucial when the user tunes a SQL statement by adding hints to its text and then wants to load the resulting plan(s) into the SQL plan baseline of the original SQL statement.</td>
</tr>
<tr>
<td>fixed</td>
<td>Default 'NO' means the loaded plans are used as non-fixed plans. Value 'YES' means the loaded plans are used as fixed plans and the SQL plan baseline will not be evolved over time.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>One of possible attribute names:</td>
</tr>
<tr>
<td></td>
<td>• <code>SQL_TEXT</code></td>
</tr>
<tr>
<td></td>
<td>• <code>PARSING_SCHEMA_NAME</code></td>
</tr>
<tr>
<td></td>
<td>• <code>MODULE</code></td>
</tr>
<tr>
<td></td>
<td>• <code>ACTION</code></td>
</tr>
</tbody>
</table>
Table 153-19  (Cont.) LOAD_PLANS_FROM_CURSOR_CACHE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_value</td>
<td>Attribute value is used as a search pattern of LIKE predicate if attribute name is 'SQL_TEXT'. Otherwise, it is used as an equality search value. For example, for specifying attribute_name =&gt; 'SQL_TEXT', and attribute_value =&gt; '% HR-123 %' means applying SQL_TEXT LIKE '% HR-123 %' as a selection filter. Similarly, specifying attribute_name =&gt; 'MODULE', and attribute_value =&gt; 'HR' means applying 'MODULE = 'HR' as a plan selection filter. The attribute value is upper-cased except when it is enclosed in double quotes or attribute name is 'SQL_TEXT'.</td>
</tr>
<tr>
<td>enabled</td>
<td>Default 'YES' means the loaded plans are enabled for use by the optimizer.</td>
</tr>
</tbody>
</table>

Return Values

Number of plans loaded

Usage Notes

Invoking this subprogram requires the ADMINISTER SQL MANAGEMENT OBJECT privilege.

153.6.15 LOAD_PLANS_FROM_SQLSET Function

This function loads plans stored in a SQL tuning set (STS) into SQL plan baselines. The plans loaded from STS are not verified for performance but added as accepted plans to existing or new SQL plan baselines. This function can be used to seed SQL management base with new SQL plan baselines.

Syntax

```sql
DBMS_SPM.LOAD_PLANS_FROM_SQLSET (  
    sqlset_name      IN  VARCHAR2,  
    sqlset_owner     IN  VARCHAR2 := NULL,  
    basic_filter     IN  VARCHAR2 := NULL,  
    fixed            IN  VARCHAR2 := 'NO',  
    enabled          IN  VARCHAR2 := 'YES',  
    commit_rows      IN  NUMBER   := 1000)  
RETURN PLS_INTEGER;
```

Parameters

Table 153-20  LOAD_PLANS_FROM_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Name of the STS from where the plans are loaded into SQL plan baselines.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Owner of STS. NULL means current schema is the owner.</td>
</tr>
</tbody>
</table>
Table 153-20  (Cont.) LOAD_PLANS_FROM_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic_filter</td>
<td>A filter applied to the STS to select only qualifying plans to be loaded. The filter can take the form of any WHERE clause predicate that can be specified against the view DBA_SQLSET_STATMENTS. For example: basic_filter =&gt; 'sql_text like '%'LOAD_STS%'' or basic_filter =&gt; 'sql_id='b62q7nc33gzwx'''.</td>
</tr>
<tr>
<td>fixed</td>
<td>Default 'NO' means the loaded plans are used as non-fixed plans. Value 'YES' means the loaded plans are used as fixed plans and the SQL plan baseline will not be evolved over time.</td>
</tr>
<tr>
<td>enabled</td>
<td>Default 'YES' means the loaded plans are enabled for use by the optimizer</td>
</tr>
<tr>
<td>commit_rows</td>
<td>Number of SQL plans to load before doing a periodic commit. This helps to shorten the undo log.</td>
</tr>
</tbody>
</table>

Return Values

The number of plans loaded

Usage Notes

- To load plans from a remote system, first load the plans into an STS on the remote system, export/import the STS from remote to local system, and then use this function.
- To load plans from Automatic Workload Repository (AWR), first load the plans stored in AWR snapshots into an STS, and then use this procedure.
- The user can also capture plans resident in the cursor cache for one or more SQL statements into an STS, and then use this procedure.

153.6.16 MIGRATE_STORED_OUTLINE Functions

This function migrates stored outlines for one or more SQL statements to plan baselines in the SQL management base (SMB). Users can specify which stored outline(s) to be migrated based on outline name, SQL text, or outline category, or migrate all stored outlines in the system to SQL plan baselines.

This second overload of the function migrates stored outlines for one or more SQL statements to plan baselines in the SQL management base (SMB) given one or more outline names.

Syntax

```sql
DBMS_SPM.MIGRATE_STORED_OUTLINE (
    attribute_name     IN  VARCHAR2,
    attribute_value    IN  CLOB,
    fixed              IN  VARCHAR2 := 'NO'
) RETURN CLOB;

DBMS_SPM.MIGRATE_STORED_OUTLINE (
    outln_list         IN  DBMS_SPM.NAME_LIST,
    fixed              IN  VARCHAR2 := 'NO'
) RETURN CLOB;
```
Parameters

Table 153-21 MIGRATE_STORED_OUTLINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| attribute_name | Specifies the type of parameter used in attribute_value to identify the migrated stored outlines. It is case insensitive. Possible values:  
|                | • outline_name                                                             |
|                | • sql_text                                                                  |
|                | • category                                                                  |
|                | • all                                                                       |
| attribute_value| Based on attribute_name, this can be:                                        |
|                | • Name of stored outline to be migrated                                      |
|                | • SQL text of stored outlines to be migrated                                 |
|                | • Category of stored outlines to be migrated                                 |
|                | • NULL if attribute_name is all                                             |
| fixed          | NO (default) or YES. Specifies the "fixed" status of the plans generated during migration. By default, plans are generated as "non-fixed" plans. |
| outln_list     | List of outline names to be migrated                                        |

Return Values

A CLOB containing a formatted report to describe the statistics during the migration, including:

• Number of stored outlines successfully migrated
• Number of stored outlines (and also the corresponding outline names) failed to be migrated and the reasons for the failure

Usage Note

• When the user specifies an outline name, the function migrates stored outlines to plan baseline based on given outline name, which uniquely identifies a single stored outline to be migrated.

• When the user specifies SQL text, the function migrates all stored outlines created for a given SQL statement. A single SQL statement can have multiple stored outlines created for it under different category names. One plan baseline plan is created for each stored outline. The new plan baselines have category names set to DEFAULT. The module name of a plan baseline is set to be the same as the stored outline.

• When the user specifies a category name, the function migrates all stored outlines with the given category name. Only one stored outline exists per category per SQL statement. One plan baseline is created for each stored outline.

• When user specifies to migrate all, the function migrates all stored outlines in the system to plan baselines. One plan baseline is created for each stored outline.
153.6.17 PACK_STGTAB_BASELINE Function

This function packs (exports) SQL plan baselines from SQL management base into a staging table.

Syntax

```
DBMS_SPM.PACK_STGTAB_BASELINE (  
    table_name       IN VARCHAR2,  
    table_owner      IN VARCHAR2 := NULL,  
    sql_handle       IN VARCHAR2 := NULL,  
    plan_name        IN VARCHAR2 := NULL,  
    sql_text         IN CLOB := NULL,  
    creator          IN VARCHAR2 := NULL,  
    origin           IN VARCHAR2 := NULL,  
    enabled          IN VARCHAR2 := NULL,  
    accepted         IN VARCHAR2 := NULL,  
    fixed            IN VARCHAR2 := NULL,  
    module           IN VARCHAR2 := NULL,  
    action           IN VARCHAR2 := NULL)  
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of staging table into which SQL plan baselines are packed (case insensitive unless double quoted)</td>
</tr>
<tr>
<td>table_owner</td>
<td>Name of staging table owner. Default NULL means current schema is the table owner</td>
</tr>
<tr>
<td>sql_handle</td>
<td>SQL handle (case sensitive)</td>
</tr>
<tr>
<td>plan_name</td>
<td>Plan name (case sensitive, % wildcards accepted)</td>
</tr>
<tr>
<td>sql_text</td>
<td>SQL text string (case sensitive, % wildcards accepted)</td>
</tr>
<tr>
<td>creator</td>
<td>Creator of SQL plan baseline (case insensitive unless double quoted)</td>
</tr>
<tr>
<td>origin</td>
<td>Origin of SQL plan baseline, should be 'MANUAL-LOAD', 'AUTO-CAPTURE', 'MANUAL_SQLTUNE' or 'AUTO-SQLTUNE' (case insensitive)</td>
</tr>
<tr>
<td>enabled</td>
<td>Must be 'YES' or 'NO' (case insensitive)</td>
</tr>
<tr>
<td>accepted</td>
<td>Must be 'YES' or 'NO' (case insensitive)</td>
</tr>
<tr>
<td>fixed</td>
<td>Must be 'YES' or 'NO' (case insensitive)</td>
</tr>
<tr>
<td>module</td>
<td>Module (case sensitive)</td>
</tr>
<tr>
<td>action</td>
<td>Action (case sensitive)</td>
</tr>
</tbody>
</table>

Return Values

Number of SQL plan baselines packed
153.6.18 RESET_EVOLVE_TASK Procedure

This procedure resets an evolve task to its initial state.

All intermediate results will be removed from the task. Call this procedure on a task that is not currently executing.

Syntax

```sql
DBMS_SPM.RESET_EVOLVE_TASK  (
    task_name    IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to reset</td>
</tr>
</tbody>
</table>

153.6.19 RESUME_EVOLVE_TASK Procedure

The procedure resumes a previously interrupted task.

Syntax

```sql
DBMS_SPM.RESUME_EVOLVE_TASK  (
    task_name    IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to resume</td>
</tr>
</tbody>
</table>

153.6.20 REPORT_AUTO_EVOLVE_TASK Function

The procedure displays the results of an execution of an automatic evolve task.

Syntax

```sql
DBMS_SPM.REPORT_AUTO_EVOLVE_TASK  (
    type            IN  VARCHAR2  := TYPE_TEXT,
    level           IN  VARCHAR2  := LEVEL_TYPICAL,
    section         IN  VARCHAR2  := SECTION_ALL,
    object_id       IN  NUMBER    := NULL,
    execution_name  IN  VARCHAR2 := NULL)
RETURN CLOB;
```
Parameters

Table 153-25 REPORT_AUTO_EVOLVE_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of the report. Possible values are TEXT, HTML, XML</td>
</tr>
<tr>
<td>level</td>
<td>Format of the report. Possible values are BASIC, TYPICAL, ALL.</td>
</tr>
<tr>
<td>section</td>
<td>Particular section in the report. Possible values are: SUMMARY, FINDINGS, PLANS, INFORMATION, ERRORS, ALL.</td>
</tr>
<tr>
<td>object_id</td>
<td>Identifier of the advisor framework object that represents a single plan. If NULL, the report is generated for all objects.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name to qualify and identify an execution. If NULL, the report is generated for the last task execution.</td>
</tr>
</tbody>
</table>

Return Values

The report

153.6.21 REPORT_EVOLVE_TASK Function

The procedure displays the results of an evolved task.

Syntax

```
DBMS_SPM.REPORT_EVOLVE_TASK  (
   task_name       IN  VARCHAR2,
   type            IN  VARCHAR2  := TYPE_TEXT,
   level           IN  VARCHAR2  := LEVEL_TYPICAL,
   section         IN  VARCHAR2  := SECTION_ALL,
   object_id       IN  NUMBER    := NULL,
   task_owner      IN  VARCHAR2  := NULL,
   execution_name  IN  VARCHAR2  := NULL)
RETURN CLOB;
```

Parameters

Table 153-26 REPORT_EVOLVE_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of task to report</td>
</tr>
<tr>
<td>type</td>
<td>Type of the report. Possible values are TEXT, HTML, XML</td>
</tr>
<tr>
<td>level</td>
<td>Format of the report. Possible values are BASIC, TYPICAL, ALL.</td>
</tr>
<tr>
<td>section</td>
<td>Particular section in the report. Possible values are: SUMMARY, FINDINGS, PLANS, INFORMATION, ERRORS, ALL.</td>
</tr>
<tr>
<td>object_id</td>
<td>Identifier of the advisor framework object that represents a single plan. If NULL, the report is generated for all objects.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Owner of the evolve task. Defaults to the current schema owner.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name to qualify and identify an execution. If NULL, the report is generated for the last task execution.</td>
</tr>
</tbody>
</table>
Return Values

The report

153.6.22 SET_EVOLVE_TASK_PARAMETER Procedure

The procedure sets a parameter of an evolve task, either a VARCHAR2 or a NUMBER.

Syntax

```sql
DBMS_SPM.SET_EVOLVE_TASK_PARAMETER  (
    task_name     IN  VARCHAR2,
    parameter     IN  VARCHAR2,
    value         IN  NUMBER);
```

```sql
DBMS_SPM.SET_EVOLVE_TASK_PARAMETER  (
    task_name     IN  VARCHAR2  := NULL,
    parameter     IN  VARCHAR2,
    value         IN  VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Evolve task name</td>
</tr>
<tr>
<td>parameter</td>
<td>Name of the parameter to set</td>
</tr>
<tr>
<td>value</td>
<td>New value of the parameter</td>
</tr>
</tbody>
</table>

For parameter of type VARCHAR2, the possible parameters are:

- **ALTERNATE_PLAN_SOURCE**: Determines which sources to search for alternate plans: CURSOR_CACHE, AUTOMATIC_WORKLOAD_REPOSITORY, or SQL_TUNING_SETS. Multiple values can be combined by delimiting with a plus symbol (+). The default value is CURSOR_CACHE+AUTOMATIC_WORKLOAD_REPOSITORY.

- **ALTERNATE_PLAN_BASELINE**: Value EXISTING indicates that alternate plans should be loaded for statements with existing SQL plan baselines. Value NEW indicates that alternate plans should be loaded for statements which do not have a SQL plan baseline, in which case a new baseline for the statement will be created. Value EXISTING+NEW will load alternate plans in both cases.

For parameter of type NUMBER, the possible parameters are:

- **TIME_LIMIT**: Global time limit (default DBMS_SPM.AUTO_LIMIT) in minutes. This is the total time allowed for the task.
- **ALTERNATE_PLAN_LIMIT**: The maximum number of plans in total (not for each statement) to load from alternate sources. Default value is 10.
153.6.23 UNPACK_STGTAB_BASELINE Function

This function unpacks (imports) SQL plan baselines from a staging table into SQL management base.

Syntax

```sql
DBMS_SPM.UNPACK_STGTAB_BASELINE (  
  table_name       IN VARCHAR2,  
  table_owner      IN VARCHAR2 := NULL,  
  sql_handle       IN VARCHAR2 := NULL,  
  plan_name        IN VARCHAR2 := NULL,  
  sql_text         IN CLOB := NULL,  
  creator          IN VARCHAR2 := NULL,  
  origin           IN VARCHAR2 := NULL,  
  enabled          IN VARCHAR2 := NULL,  
  accepted         IN VARCHAR2 := NULL,  
  fixed            IN VARCHAR2 := NULL,  
  module           IN VARCHAR2 := NULL,  
  action           IN VARCHAR2 := NULL)  
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of staging table from which SQL plan baselines are unpacked (case insensitive unless double quoted)</td>
</tr>
<tr>
<td>table_owner</td>
<td>Name of staging table owner. Default NULL means current schema is the table owner</td>
</tr>
<tr>
<td>sql_handle</td>
<td>SQL handle (case sensitive)</td>
</tr>
<tr>
<td>plan_name</td>
<td>Plan name (case sensitive, % wildcards accepted)</td>
</tr>
<tr>
<td>sql_text</td>
<td>SQL text string (case sensitive, % wildcards accepted)</td>
</tr>
<tr>
<td>creator</td>
<td>Creator of SQL plan baseline (case insensitive unless double quoted)</td>
</tr>
<tr>
<td>origin</td>
<td>Origin of SQL plan baseline, should be 'MANUAL-LOAD', 'AUTO-CAPTURE', 'MANUAL_SQLTUNE' or 'AUTO-SQLTUNE' (case insensitive)</td>
</tr>
<tr>
<td>enabled</td>
<td>Must be 'YES' or 'NO' (case insensitive)</td>
</tr>
<tr>
<td>accepted</td>
<td>Must be 'YES' or 'NO' (case insensitive)</td>
</tr>
<tr>
<td>fixed</td>
<td>Must be 'YES' or 'NO' (case insensitive)</td>
</tr>
<tr>
<td>module</td>
<td>Module (case sensitive)</td>
</tr>
<tr>
<td>action</td>
<td>Action (case sensitive)</td>
</tr>
</tbody>
</table>

Return Values

Number of plans unpacked
The DBMS_SQL package provides an interface to use dynamic SQL to parse any data manipulation language (DML) or data definition language (DDL) statement using PL/SQL.

For example, you can enter a DROP TABLE statement from within a stored procedure by using the PARSE Procedures supplied with the DBMS_SQL package.

This chapter contains the following topics:

• Overview
• Security Model
• Constants
• Exceptions
• Operational Notes
• Examples
• Data Structures
• Summary of DBMS_SQL Subprograms

See Also:
For more information on native dynamic SQL, see Oracle Database PL/SQL Language Reference.

154.1 DBMS_SQL Overview

Oracle lets you write stored procedures and anonymous PL/SQL blocks that use dynamic SQL. Dynamic SQL statements are not embedded in your source program; rather, they are stored in character strings that are input to, or built by, the program at runtime. This enables you to create more general-purpose procedures. For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.

Native Dynamic SQL is an alternative to DBMS_SQL that lets you place dynamic SQL statements directly into PL/SQL blocks. In most situations, Native Dynamic SQL is easier to use and performs better than DBMS_SQL. However, Native Dynamic SQL itself has certain limitations:

• There is no support for so-called Method 4 (for dynamic SQL statements with an unknown number of inputs or outputs)
• There are some tasks that can only be performed using DBMS_SQL. For tasks that require DBMS_SQL, see Oracle Database PL/SQL Language Reference.

The ability to use dynamic SQL from within stored procedures generally follows the model of the Oracle Call Interface (OCI).
PL/SQL differs somewhat from other common programming languages, such as C. For example, addresses (also called pointers) are not user-visible in PL/SQL. As a result, there are some differences between the Oracle Call Interface and the DBMS_SQL package. These differences include the following:

- The OCI binds by address and the DBMS_SQL package binds by value.
- With DBMS_SQL you must call VARIABLE_VALUE to retrieve the value of an OUT parameter for an anonymous block, and you must call COLUMN_VALUE after fetching rows to retrieve the values of the columns in the rows into your program.
- The current release of the DBMS_SQL package does not provide CANCEL cursor procedures.
- Indicator variables are not required, because NULLs are fully supported as values of a PL/SQL variable.

154.2 DBMS_SQL Security Model

DBMS_SQL is a SYS-owned package compiled with AUTHID CURRENT_USER. Any DBMS_SQL subprogram called from an anonymous PL/SQL block runs with the privileges of the current user.

Preventing Malicious or Accidental Access of Open Cursor Numbers

An error, ORA-29471, is raised when any DBMS_SQL subprogram is called with a cursor number that does not denote an open cursor. When the error is raised, an alert is issued to the alert log and DBMS_SQL becomes inoperable for the life of the session.

If the actual value for the cursor number in a call to the IS_OPEN Function denotes a cursor currently open in the session, the return value is TRUE. If the actual value is NULL, then the return value is FALSE. Otherwise, this raises an ORA-29471 error.

Preventing Inappropriate Use of a Cursor

Cursors are protected from security breaches that subvert known existing cursors.

Checks are made when binding and executing. Optionally, checks may be performed for every single DBMS_SQL subprogram call. The check is:

- The current_user is the same on calling the subprogram as it was on calling the most recent parse.
- The enabled roles on calling the subprogram must be identical to the enabled roles on calling the most recent parse.
- The container is the same on calling the subprogram as it was on calling the most recent parse.

Consistent with the use of definer's rights subprograms, roles do not apply. If either check fails, then an ORA-29470 error is raised.

The mechanism for defining when checks are performed is a new overload for the OPEN_CURSOR subprogram, which takes a formal parameter, security_level, with allowed values NULL, 1 and 2.
- When security_level = 1 (or is NULL), the checks are made only when binding and executing.
- When security_level = 2, the checks are always made.

**Upgrade Considerations**

This security regime is stricter than those in the previous releases. As a consequence, users of DBMS_SQL may encounter runtime errors on upgrade.

### 154.3 DBMS_SQL Constants

The DBMS_SQL Constants package provides constants that are used with the language_flag parameter of the PARSE Procedures.

These constants are described in the following table.

**Table 154-1  DBMS_SQL Constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V6</td>
<td>INTEGER</td>
<td>0</td>
<td>Specifies Oracle database version 6 behavior</td>
</tr>
<tr>
<td>NATIVE</td>
<td>INTEGER</td>
<td>1</td>
<td>Specifies normal behavior for the database to which the program is connected</td>
</tr>
<tr>
<td>V7</td>
<td>INTEGER</td>
<td>2</td>
<td>Specifies Oracle database version 7 behavior</td>
</tr>
<tr>
<td>FOREIGN_SYNTAX</td>
<td>INTEGER</td>
<td>4294967295</td>
<td>Specifies a non-Oracle database syntax and behavior. The SQL statement to be parsed needs to be translated first using the SQL translation profile set in the database session. The SQL translation profile is a database schema object that directs how SQL statements are translated to Oracle. An error is raised if a profile is not set.</td>
</tr>
</tbody>
</table>

**Related Topics**

- **PARSE Procedures**
  This procedure parses the given statement in the given cursor. All statements are parsed immediately. In addition, DDL statements are run immediately when parsed.
154.4 DBMS_SQL Operational Notes

These operational notes describe processing queries, processing updates, inserts, and deletes, and locating errors.

Processing Queries
If you are using dynamic SQL to process a query, then you must perform the following steps:

1. Specify the variables that are to receive the values returned by the \texttt{SELECT} statement by calling the \texttt{DEFINE\_COLUMN} Procedures, the \texttt{DEFINE\_COLUMN\_LONG} Procedure, or the \texttt{DEFINE\_ARRAY} Procedure.
2. Run your \texttt{SELECT} statement by calling the \texttt{EXECUTE} Function.
3. Call the \texttt{FETCH\_ROWS} Function (or \texttt{EXECUTE\_AND\_FETCH}) to retrieve the rows that satisfied your query.
4. Call \texttt{COLUMN\_VALUE} Procedure or \texttt{COLUMN\_VALUE\_LONG} Procedure to determine the value of a column retrieved by the \texttt{FETCH\_ROWS} Function for your query. If you used anonymous blocks containing calls to PL/SQL procedures, then you must call the \texttt{VARIABLE\_VALUE} Procedures to retrieve the values assigned to the output variables of these procedures.

Processing Updates, Inserts, and Deletes
If you are using dynamic SQL to process an \texttt{INSERT}, \texttt{UPDATE}, or \texttt{DELETE}, then you must perform the following steps:

1. Run your \texttt{INSERT}, \texttt{UPDATE}, or \texttt{DELETE} statement by calling the \texttt{EXECUTE} Function.
2. If statements have the \texttt{returning} clause, then you must call the \texttt{VARIABLE\_VALUE} Procedures to retrieve the values assigned to the output variables.

Locating Errors
The \texttt{DBMS\_SQL} package has additional functions for obtaining information about the last referenced cursor in the session. The values returned by these functions are meaningful only immediately after a SQL statement is run. In addition, some error-locating functions are meaningful only after certain \texttt{DBMS\_SQL} calls. For example, you call the \texttt{LAST\_ERROR\_POSITION} Function immediately after calling one of the \texttt{PARSE} Procedures.

154.5 DBMS_SQL Execution Flow

These functions comprise the DBMS_SQL execution flow.

1. \texttt{OPEN\_CURSOR}
2. \texttt{PARSE}
3. \texttt{BIND\_VARIABLE, BIND\_VARIABLE\_PKG} or \texttt{BIND\_ARRAY}
4. \texttt{DEFINE\_COLUMN\_DEFINE\_COLUMN\_LONG} or \texttt{DEFINE\_ARRAY}
5. \texttt{EXECUTE}
6. FETCH_ROWS or EXECUTE_AND_FETCH
7. VARIABLE_VALUE, VARIABLE_PKG, COLUMN_VALUE or COLUMN_VALUE_LONG
8. CLOSE_CURSOR

154.5.1 OPEN_CURSOR

To process a SQL statement, you must have an open cursor. When you call the OPEN_CURSOR Functions, you receive a cursor ID number for the data structure representing a valid cursor maintained by Oracle.

These cursors are distinct from cursors defined at the precompiler, OCI, or PL/SQL level, and are used only by the DBMS_SQL package.

Related Topics
- OPEN_CURSOR Functions
  This function opens a new cursor.

154.5.2 PARSE

Every SQL statement must be parsed by calling the PARSE procedures. Parsing the statement checks the statement’s syntax and associates it with the cursor in your program.

You can parse any DML or DDL statement. DDL statements are run on the parse, which performs the implied commit.

The execution flow of DBMS_SQL is shown in Figure 154-1.
Figure 154-1  DBMS_SQL Execution Flow

Related Topics
- **PARSE Procedures**
  This procedure parses the given statement in the given cursor. All statements are parsed immediately. In addition, DDL statements are run immediately when parsed.
154.5.3 BIND_VARIABLE, BIND_VARIABLE_PKG or BIND_ARRAY

Many DML statements require that data in your program be input to Oracle. When you define a SQL statement that contains input data to be supplied at runtime, you must use placeholders in the SQL statement to mark where data must be supplied.

For each placeholder in the SQL statement, you must call one of the BIND_ARRAY Procedures, or BIND_VARIABLE Procedures, or the BIND_VARIABLE_PKG Procedure to supply the value of a variable in your program (or the values of an array) to the placeholder. When the SQL statement is subsequently run, Oracle uses the data that your program has placed in the output and input, or bind variables.

DBMS_SQL can run a DML statement multiple times — each time with a different bind variable. The BIND_ARRAY procedure lets you bind a collection of scalars, each value of which is used as an input variable once for each EXECUTE. This is similar to the array interface supported by the OCI.

Note that the datatype of the values bound to placeholders cannot be PL/SQL-only datatypes.

154.5.4 DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY

The DEFINE_COLUMN, DEFINE_COLUMN_LONG, and DEFINE_ARRAY procedures specify the variables that receive SELECT values on a query.

The columns of the row being selected in a SELECT statement are identified by their relative positions as they appear in the select list, from left to right. For a query, you must call one of the define procedures (DEFINE_COLUMN Procedures, DEFINE_COLUMN_LONG Procedure, or DEFINE_ARRAY Procedure) to specify the variables that are to receive the SELECT values, much the way an INTO clause does for a static query.

Use the DEFINE_COLUMN_LONG procedure to define LONG columns, in the same way that DEFINE_COLUMN is used to define non-LONG columns. You must call DEFINE_COLUMN_LONG before using the COLUMN_VALUE_LONG Procedure to fetch from the LONG column.

Use the DEFINE_ARRAY procedure to define a PL/SQL collection into which you want to fetch rows in a single SELECT statement. DEFINE_ARRAY provides an interface to fetch multiple rows at one fetch. You must call DEFINE_ARRAY before using the COLUMN_VALUE procedure to fetch the rows.

154.5.5 EXECUTE

Call the EXECUTE Function to run your SQL statement.

Related Topics

- EXECUTE Function
  This function executes a given cursor. This function accepts the ID number of the cursor and returns the number of rows processed.
154.5.6 FETCH_ROWS or EXECUTE_AND_FETCH

The FETCH_ROWS Function retrieves the rows that satisfy the query. Each successive fetch retrieves another set of rows, until the fetch is unable to retrieve any more rows. Instead of calling EXECUTE Function and then FETCH_ROWS, you may find it more efficient to call EXECUTE_AND_FETCH Function if you are calling EXECUTE for a single execution.

Related Topics

- FETCH_ROWS Function
  This function fetches a row from a given cursor.

- EXECUTE Function
  This function executes a given cursor. This function accepts the ID number of the cursor and returns the number of rows processed.

- EXECUTE_AND_FETCH Function
  This function executes the given cursor and fetches rows.

154.5.7 VARIABLE_VALUE, VARIABLE_VALUE_PKG, COLUMN_VALUE, or COLUMN_VALUE_LONG

The type of call determines which procedure or function to use.

For queries, call the COLUMN_VALUE Procedure to determine the value of a column retrieved by the FETCH_ROWS Function.

For anonymous blocks containing calls to PL/SQL procedures or DML statements with returning clause, call the VARIABLE_VALUE Procedures or the VARIABLE_VALUE_PKG Procedure to retrieve the values assigned to the output variables when statements were run.

To fetch only part of a LONG database column (which can be up to two gigabytes in size), use the DEFINE_COLUMN_LONG Procedure. You can specify the offset (in bytes) into the column value, and the number of bytes to fetch.

154.5.8 CLOSE_CURSOR

When you no longer need a cursor for a session, close the cursor by calling the CLOSE_CURSOR Procedure. If you are using an Oracle Open Gateway, then you may need to close cursors at other times as well. Consult your Oracle Open Gateway documentation for additional information.

Related Topics

- CLOSE_CURSOR Procedure
  This procedure closes a given cursor.
154.6 DBMS_SQL Exceptions

This exception is raised by the COLUMN_VALUE Procedure or the VARIABLE_VALUE Procedures when the type of the given OUT parameter (for where to put the requested value) is different from the type of the value.

inconsistent_type EXCEPTION;
pragma exception_init(inconsistent_type, -6562);

Related Topics

- COLUMN_VALUE Procedure
  This procedure returns the value of the cursor element for a given position in a given cursor. This procedure is used to access the data fetched by calling FETCH_ROWS.

- VARIABLE_VALUE Procedures
  This procedure returns the value of the named variable for a given cursor. It is used to return the values of bind variables inside PL/SQL blocks or DML statements with returning clause.

154.7 DBMS_SQL Examples

These example procedures use the DBMS_SQL package.

Example : Using DBMS_SQL Demo

This example does not need dynamic SQL because the text of the statement is known at compile time, but it illustrates the basic concept underlying the package.

The DEMO procedure deletes all of the employees from the EMP table whose salaries are greater than the salary that you specify when you run DEMO.

CREATE OR REPLACE PROCEDURE demo(salary IN NUMBER) AS
  cursor_name INTEGER;
  rows_processed INTEGER;
BEGIN
  cursor_name := dbms_sql.open_cursor;
  DBMS_SQL.PARSE(cursor_name, 'DELETE FROM emp WHERE sal > :x',
                  DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(cursor_name, ':x', salary);
  rows_processed := DBMS_SQL.EXECUTE(cursor_name);
  DBMS_SQL.CLOSE_CURSOR(cursor_name);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;

Example 2

The following sample procedure is passed a SQL statement, which it then parses and runs:

CREATE OR REPLACE PROCEDURE exec(STRING IN varchar2) AS
  cursor_name INTEGER;
  ret INTEGER;
BEGIN
cursor_name := DBMS_SQL.OPEN_CURSOR;

DDL statements are run by the parse call, which performs the implied commit.

```
DBMS_SQL.PARSE(cursor_name, string, DBMS_SQL.NATIVE);
ret := DBMS_SQL.EXECUTE(cursor_name);
DBMS_SQL.CLOSE_CURSOR(cursor_name);
```

Creating such a procedure enables you to perform the following operations:

- The SQL statement can be dynamically generated at runtime by the calling program.
- The SQL statement can be a DDL statement or a DML without binds.

For example, after creating this procedure, you could make the following call:

```
exec('create table acct(c1 integer)');
```

You could even call this procedure remotely, as shown in the following example. This lets you perform remote DDL.

```
exec@domain.com('CREATE TABLE acct(c1 INTEGER)');
```

### Example 3

The following sample procedure is passed the names of a source and a destination table, and copies the rows from the source table to the destination table. This sample procedure assumes that both the source and destination tables have the following columns:

| id        | of type NUMBER |
| name      | of type VARCHAR2(30) |
| birthdate | of type DATE |

This procedure does not need the use of dynamic SQL; however, it illustrates the concepts of this package.

```
CREATE OR REPLACE PROCEDURE copy (
    source IN VARCHAR2,
    destination IN VARCHAR2) IS
    id_var             NUMBER;
    name_var           VARCHAR2(30);
    birthdate_var      DATE;
    source_cursor      INTEGER;
    destination_cursor INTEGER;
    ignore             INTEGER;
BEGIN
  -- Prepare a cursor to select from the source table:
  source_cursor := dbms_sql.open_cursor;
  DBMS_SQL.PARSE(source_cursor, 
    'SELECT id, name, birthdate FROM ' || source, 
    DBMS_SQL.NATIVE);
  DBMS_SQL.DEFINE_COLUMN(source_cursor, 1, id_var);
  DBMS_SQL.DEFINE_COLUMN(source_cursor, 2, name_var, 30);
  DBMS_SQL.DEFINE_COLUMN(source_cursor, 3, birthdate_var);
  ignore := DBMS_SQL.EXECUTE(source_cursor);

  -- Prepare a cursor to insert into the destination table:
destination_cursor := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(destination_cursor, 
    'INSERT INTO ' || destination || 
    ' VALUES (:id_bind, :name_bind, :birthdate_bind)', 
    DBMS_SQL.NATIVE);

-- Fetch a row from the source table and insert it into the destination table:
LOOP
    IF DBMS_SQL.FETCH_ROWS(source_cursor)>0 THEN
        -- get column values of the row
        DBMS_SQL.COLUMN_VALUE(source_cursor, 1, id_var);
        DBMS_SQL.COLUMN_VALUE(source_cursor, 2, name_var);
        DBMS_SQL.COLUMN_VALUE(source_cursor, 3, birthdate_var);
        -- Bind the row into the cursor that inserts into the destination table. You
        -- could alter this example to require the use of dynamic SQL by inserting an
        -- if condition before the bind.
        DBMS_SQL.BIND_VARIABLE(destination_cursor, ':id_bind', id_var);
        DBMS_SQL.BIND_VARIABLE(destination_cursor, ':name_bind', name_var);
        DBMS_SQL.BIND_VARIABLE(destination_cursor, ':birthdate_bind', 
            birthdate_var);
        ignore := DBMS_SQL.EXECUTE(destination_cursor);
    ELSE
        -- No more rows to copy:
        EXIT;
    END IF;
END LOOP;

-- Commit and close all cursors:
COMMIT;
DBMS_SQL.CLOSE_CURSOR(source_cursor);
DBMS_SQL.CLOSE_CURSOR(destination_cursor);
EXCEPTION
WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(source_cursor) THEN
        DBMS_SQL.CLOSE_CURSOR(source_cursor);
    END IF;
    IF DBMS_SQL.IS_OPEN(destination_cursor) THEN
        DBMS_SQL.CLOSE_CURSOR(destination_cursor);
    END IF;
    RAISE;
END;
/

Example 4: RETURNING clause

With this clause, INSERT, UPDATE, and DELETE statements can return values of expressions in
bind variables.

If a single row is inserted, updated, or deleted, then use DBMS_SQL.BIND_VARIABLE to bind
these outbinds. To get the values in these bind variables, call DBMS_SQL.VARIABLE_VALUE

Note:

This process is similar to DBMS_SQL.VARIABLE_VALUE, which must be called after
running a PL/SQL block with an outbind inside DBMS_SQL.
i) Single-row insert

CREATE OR REPLACE PROCEDURE single_Row_insert
    (c1 NUMBER, c2 NUMBER, r OUT NUMBER) is
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'INSERT INTO tab VALUES (:bnd1, :bnd2) ' ||
        'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd3', r);
    n := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r); -- get value of outbind variable
    DBMS_SQL.CLOSE_CURSOR(c);
END;
/

ii) Single-row update

CREATE OR REPLACE PROCEDURE single_Row_update
    (c1 NUMBER, c2 NUMBER, r OUT NUMBER) IS
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'UPDATE tab SET c1 = :bnd1, c2 = :bnd2 ' ||
        'WHERE rownum < 2 ' ||
        'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd3', r);
    n := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
    DBMS_SQL.CLOSE_CURSOR(c);
END;
/

iii) Single-row delete

CREATE OR REPLACE PROCEDURE single_Row_Delete
    (c1 NUMBER, r OUT NUMBER) is
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'DELETE FROM tab WHERE ROWNUM = :bnd1 ' ||
        'RETURNING c1*c2 INTO :bnd2', DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd2', r);
    n := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.VARIABLE_VALUE(c, 'bnd2', r);-- get value of outbind variable
    DBMS_SQL.CLOSE_CURSOR(c);
END;
/

iv) Multiple-row insert
CREATE OR REPLACE PROCEDURE multi_Row_insert
    (c1 DBMS_SQL.NUMBER_TABLE, c2 DBMS_SQL.NUMBER_TABLE,
     r OUT DBMS_SQL.NUMBER_TABLE) is
    c NUMBER;
    n NUMBER;
    BEGIN
        c := DBMS_SQL.OPEN_CURSOR;
        DBMS_SQL.PARSE(c, 'insert into tab VALUES (:bnd1, :bnd2) ' ||
                        'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
        DBMS_SQL.BIND_ARRAY(c, 'bnd1', c1);
        DBMS_SQL.BIND_ARRAY(c, 'bnd2', c2);
        DBMS_SQL.BIND_ARRAY(c, 'bnd3', r);
        n := DBMS_SQL.EXECUTE(c);
        DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
        DBMS_SQL.CLOSE_CURSOR(c);
    END;
/

v) Multiple-row update.

CREATE OR REPLACE PROCEDURE multi_Row_update
    (c1 NUMBER, c2 NUMBER, r OUT DBMS_SQL.NUMBER_TABLE) IS
    c NUMBER;
    n NUMBER;
    BEGIN
        c := DBMS_SQL.OPEN_CURSOR;
        DBMS_SQL.PARSE(c, 'UPDATE tab SET c1 = :bnd1 WHERE c2 = :bnd2 ' ||
                        'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
        DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
        DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
        DBMS_SQL.BIND_ARRAY(c, 'bnd3', r);
        n := DBMS_SQL.EXECUTE(c);
        DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
        DBMS_SQL.CLOSE_CURSOR(c);
    END;
/

Note:

bnd1 and bnd2 can be arrays too. The value of the expression for all the rows
updated will be in bnd3. There is no way to determine which rows were updated for
each value of bnd1 and bnd2.

vi) Multiple-row delete

CREATE OR REPLACE PROCEDURE multi_row_delete
    (c1 DBMS_SQL.NUMBER_TABLE,
     r OUT DBMS_SQL.NUMBER_TABLE) is
    c NUMBER;
    n NUMBER;
    BEGIN
        c := DBMS_SQL.OPEN_CURSOR;
        DBMS_SQL.PARSE(c, 'DELETE FROM tab WHERE c1 = :bnd1' ||
                        'RETURNING c1*c2 INTO :bnd2', DBMS_SQL.NATIVE);
        DBMS_SQL.BIND_ARRAY(c, 'bnd1', c1);
        DBMS_SQL.BIND_ARRAY(c, 'bnd2', c2);
        DBMS_SQL.BIND_ARRAY(c, 'bnd3', r);
        n := DBMS_SQL.EXECUTE(c);
        DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
        DBMS_SQL.CLOSE_CURSOR(c);
    END;
/
vii) outbound in bulk PL/SQL

CREATE OR REPLACE PROCEDURE foo (n NUMBER, square OUT NUMBER) IS
BEGIN square := n * n; END;
/

CREATE OR REPLACE PROCEDURE bulk_plsql
    (n DBMS_SQL.NUMBER_TABLE, square OUT DBMS_SQL.NUMBER_TABLE) IS
    c NUMBER;
    r NUMBER;
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'BEGIN foo(:bnd1, :bnd2); END;', DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_ARRAY(c, 'bnd1', n);
    DBMS_SQL.BIND_ARRAY(c, 'bnd2', square);
    r := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.VARIABLE_VALUE(c, 'bnd2', square);
END;
/

Note:

DBMS_SQL.BIND_ARRAY of number Table internally binds a number. The number of times statement is run depends on the number of elements in an inbind array.

Example 5: Binds and Defines of User-defined Types in DBMS_SQL

CREATE TYPE dnames_var IS VARRAY(7) OF VARCHAR2(30)
/

CREATE TABLE depts (region VARCHAR2(25), dept_names dnames_var)
/
INSERT INTO depts VALUES('Europe', dnames_var('Shipping','Sales','Finance'))
/
INSERT INTO depts VALUES('Americas', dnames_var('Sales','Finance','Shipping'))
/
INSERT INTO depts VALUES('Asia',
    dnames_var('Finance','Payroll','Shipping','Sales'))
/

CREATE OR REPLACE PROCEDURE update_depts(new_dnames dnames_var, region VARCHAR2)
IS
    some_dnames dnames_var;
    c      NUMBER;
    r      NUMBER;
    sql_stmt VARCHAR2(32767) :=
        'UPDATE depts SET dept_names = :b1 WHERE region = :b2 RETURNING dept_names INTO :b3';
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;

    DBMS_SQL.PARSE(c, sql_stmt, dbms_sql.native);
DBMS_SQL.BIND_VARIABLE(c, 'b1', new_dnames);
DBMS_SQL.BIND_VARIABLE(c, 'b2', region);
DBMS_SQL.BIND_VARIABLE(c, 'b3', some_dnames);

r := DBMS_SQL.EXECUTE(c);

-- Get value of outbind variable
DBMS_SQL.VARIABLE_VALUE(c, 'b3', some_dnames);

DBMS_SQL.CLOSE_CURSOR(c);

-- select dept_names
sql_stmt := 'SELECT dept_names FROM depts WHERE region = :b1';

c := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(c, sql_stmt, dbms_sql.native);
DBMS_SQL.DEFINE_COLUMN(c, 1, some_dnames);
DBMS_SQL.BIND_VARIABLE(c, 'b1', region);

r := DBMS_SQL.EXECUTE_AND_FETCH(c);

DBMS_SQL.COLUMN_VALUE(c, 1, some_dnames);

DBMS_SQL.CLOSE_CURSOR(c);

-- loop through some_dnames collections
FOR i IN some_dnames.FIRST .. some_dnames.LAST  LOOP
   DBMS_OUTPUT.PUT_LINE('Dept. Name = ' || some_dnames(i) || ' Updated!');
END LOOP;
END;
/

DECLARE
new_dnames dnames_var;
BEGIN
   new_dnames := dnames_var('Benefits', 'Advertising', 'Contracting', 'Executive', 'Marketing');
   update_depts(new_dnames, 'Asia');
END;
/

154.8 DBMS_SQL Data Structures

The DBMS_SQL package defines RECORD type and TABLE type data structures.

RECORD Types

- DBMS_SQL DESC_REC Record Type (deprecated)
- DBMS_SQL DESC_REC2 Record Type
- DBMS_SQL DESC_REC3 Record Type
- DBMS_SQL DESC_REC4 Record Type

TABLE Types for DESCRIBE_COLUMNS Procedures

- DBMS_SQL DESC_TAB Table Type
DBMS_SQL DESC_TAB2 Table Type
DBMS_SQL DESC_TAB3 Table Type
DBMS_SQL DESC_TAB4 Table Type

TABLE Types For Scalar and LOB Collections

DBMS_SQL bulk operations are only supported with these predefined DBMS_SQL TABLE types.

- BFILE_TABLE Table Type
- BINARY_DOUBLE_TABLE Table Type
- BINARY_FLOAT_TABLE Table Type
- BLOB_TABLE Table Type
- CLOB_TABLE Table Type
- DATE_TABLE Table Type
- INTERVAL_DAY_TO_SECOND_TABLE Table Type
- INTERVAL_YEAR_TO_MONTH_TABLE Table Type
- NUMBER_TABLE Table Type
- TIME_TABLE Table Type
- TIME_WITH_TIME_ZONE_TABLE Table Type
- TIMESTAMP_TABLE Table Type
- TIMESTAMP_WITH_LTZ_TABLE Table Type
- TIMESTAMP_WITH_TIME_ZONE_TABLE Table Type
- UROWID_TABLE Table Type
- VARCHAR2_TABLE Table Type
- VARCHAR2A Table Type
- VARCHAR2S Table Type

154.8.1 DBMS_SQL DESC_REC Record Type

This record type holds the describe information for a single column in a dynamic query.

Note:

This type has been deprecated in favor of the DESC_REC2 Record Type.

It is the element type of the DESC_TAB table type and the DESCRIBE_COLUMNS Procedure.

Syntax

```sql
TYPE desc_rec IS RECORD (
  col_type            BINARY_INTEGER := 0,
  col_max_len         BINARY_INTEGER := 0,
);```
Fields

**Table 154-2 DESC_REC Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>col_type</code></td>
<td>Type of column</td>
</tr>
<tr>
<td><code>col_max_len</code></td>
<td>Maximum column length</td>
</tr>
<tr>
<td><code>col_name</code></td>
<td>Name of column</td>
</tr>
<tr>
<td><code>col_name_len</code></td>
<td>Length of column name</td>
</tr>
<tr>
<td><code>col_schema_name</code></td>
<td>Column schema name</td>
</tr>
<tr>
<td><code>col_schema_name_len</code></td>
<td>Length of column schema name</td>
</tr>
<tr>
<td><code>col_precision</code></td>
<td>Precision of column</td>
</tr>
<tr>
<td><code>col_scale</code></td>
<td>Scale of column</td>
</tr>
<tr>
<td><code>col_charsetid</code></td>
<td>Column character set id</td>
</tr>
<tr>
<td><code>col_charsetform</code></td>
<td>Column character set form</td>
</tr>
<tr>
<td><code>col_null_ok</code></td>
<td>NULL column flag; TRUE, if NULL possible</td>
</tr>
</tbody>
</table>

154.8.2 DBMS_SQL DESC_REC2 Record Type

DESC_REC2 is the element type of the DESC_TAB2 table type and the DESCRIBE_COLUMNS2 Procedure.

This record type is identical to DESC_REC except for the `col_name` field, which has been expanded to the maximum possible size for VARCHAR2. It is therefore preferred to DESC_REC because column name values can be greater than 32 characters. DESC_REC is deprecated as a result.

Syntax

```sql
TYPE desc_rec2 IS RECORD (  
  col_type binary_integer := 0,  
  col_max_len binary_integer := 0,  
  col_name varchar2(32767) := '',  
  col_name_len binary_integer := 0,  
  col_schema_name varchar2(32) := '',  
  col_schema_name_len binary_integer := 0,  
  col_precision binary_integer := 0,  
  col_scale binary_integer := 0,  
  col_charsetid binary_integer := 0,  
  col_charsetform binary_integer := 0,  
  col_null_ok boolean := TRUE);  
```
### Fields

Table 154-3  DESC_REC2 Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_type</td>
<td>Type of column</td>
</tr>
<tr>
<td>col_max_len</td>
<td>Maximum column length</td>
</tr>
<tr>
<td>col_name</td>
<td>Name of column</td>
</tr>
<tr>
<td>col_name_len</td>
<td>Length of column name</td>
</tr>
<tr>
<td>col_schema_name</td>
<td>Column schema name</td>
</tr>
<tr>
<td>col_schema_name_len</td>
<td>Length of column schema name</td>
</tr>
<tr>
<td>col_precision</td>
<td>Precision of column</td>
</tr>
<tr>
<td>col_scale</td>
<td>Scale of column</td>
</tr>
<tr>
<td>col_charsetid</td>
<td>Column character set id</td>
</tr>
<tr>
<td>col_charsetform</td>
<td>Column character set form</td>
</tr>
<tr>
<td>col_null_ok</td>
<td>NULL column flag; TRUE, if NULL possible</td>
</tr>
</tbody>
</table>

### Related Topics

- **DESCRIBE_COLUMNS2 Procedure**
  
  This procedure describes the specified column. This is an alternative to DESCRIBE_COLUMNS Procedure.

### 154.8.3 DBMS_SQL DESC_REC3 Record Type

DESC_REC3 is the element type of the DESC_TAB3 table type and the DESCRIBE_COLUMNS3 Procedure.

DESC_REC3 is identical to DESC_REC2 except for two additional fields to hold the type name (type_name_len) and type name len (type_name_len_len) of a column in a dynamic query. These two fields hold the type name and type name length when the column is a user-defined type (a collection or object type). The col_type_name and col_type_name_len fields are only populated when the col_type field's value is 109, the Oracle type number for user-defined types.

### Syntax

```sql
TYPE desc_rec3 IS RECORD (  
    col_type               binary_integer := 0,  
    col_max_len            binary_integer := 0,  
    col_name               varchar2(32767) := '',  
    col_name_len           binary_integer := 0,  
    col_schema_name        varchar2(32) := '',  
    col_schema_name_len    binary_integer := 0,  
    col_precision          binary_integer := 0,  
    col_scale              binary_integer := 0,  
    col_charsetid          binary_integer := 0,  
    col_charsetform        binary_integer := 0,  
    col_null_ok            boolean := TRUE,  
    col_type_name          varchar2(32767) := '',  
    col_type_name_len      binary_integer := 0);```

**Chapter 154**

DBMS_SQL Data Structures

**154-18**
Fields

Table 154-4  DESC_REC3 Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_type</td>
<td>Type of column</td>
</tr>
<tr>
<td>col_max_len</td>
<td>Maximum column length</td>
</tr>
<tr>
<td>col_name</td>
<td>Name of column</td>
</tr>
<tr>
<td>col_name_len</td>
<td>Length of column name</td>
</tr>
<tr>
<td>col_schema_name</td>
<td>Column schema name</td>
</tr>
<tr>
<td>col_schema_name_len</td>
<td>Length of column schema name</td>
</tr>
<tr>
<td>col_precision</td>
<td>Precision of column</td>
</tr>
<tr>
<td>col_scale</td>
<td>Scale of column</td>
</tr>
<tr>
<td>col_charsetid</td>
<td>Column character set ID</td>
</tr>
<tr>
<td>col_charsetform</td>
<td>Column character set form</td>
</tr>
<tr>
<td>col_null_ok</td>
<td>NULL column flag; TRUE, if NULL possible</td>
</tr>
<tr>
<td>col_type_name</td>
<td>User-define type column type name, this field is valid when col_type is 109</td>
</tr>
<tr>
<td>col_type_name_len</td>
<td>Length of user-define type column type name, this field is valid when col_type is 109</td>
</tr>
</tbody>
</table>

Related Topics

- Describe_COLUMNS3 Procedure
  This procedure describes the specified column. This is an alternative to Describe_COLUMNS Procedure.

154.8.4 DBMS_SQL DESC_REC4 Record Type

DESC_REC4 is the element type of the DESC_TAB4 table type and the Describe_COLUMNS3 Procedure.

DESC_REC4 is identical to DESC_REC3 except that it supports longer identifiers in the fields that hold the schema name (col_schema_name) and type name (col_type_name) of a column in a dynamic query.

Syntax

```
TYPE desc_rec4 IS RECORD (  
    col_type               binary_integer := 0,  
    col_max_len            binary_integer := 0,  
    col_name               varchar2(32767) := '',  
    col_name_len           binary_integer := 0,  
    col_schema_name        DBMS_ID := '',  
    col_schema_name_len    binary_integer := 0,  
    col_precision          binary_integer := 0,  
    col_scale              binary_integer := 0,  
    col_charsetid          binary_integer := 0,  
    col_charsetform        binary_integer := 0,  
    col_null_ok            boolean := TRUE,  
);```
col_type_name          DBMS_ID := '',
col_type_name_len      binary_integer := 0);

See Also:
Oracle Database PL/SQL Language Reference for more information about the predefined subtype DBMS_ID.

Fields

Table 154-5   DESC_REC4 Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_type</td>
<td>Type of column</td>
</tr>
<tr>
<td>col_max_len</td>
<td>Maximum column length</td>
</tr>
<tr>
<td>col_name</td>
<td>Name of column</td>
</tr>
<tr>
<td>col_name_len</td>
<td>Length of column name</td>
</tr>
<tr>
<td>col_schema_name</td>
<td>Column schema name</td>
</tr>
<tr>
<td>col_schema_name_len</td>
<td>Length of column schema name</td>
</tr>
<tr>
<td>col_precision</td>
<td>Precision of column</td>
</tr>
<tr>
<td>col_scale</td>
<td>Scale of column</td>
</tr>
<tr>
<td>col_charsetid</td>
<td>Column character set ID</td>
</tr>
<tr>
<td>col_charsetform</td>
<td>Column character set form</td>
</tr>
<tr>
<td>col_null_ok</td>
<td>NULL column flag; TRUE, if NULL possible</td>
</tr>
<tr>
<td>col_type_name</td>
<td>User-define type column type name, this field is valid when col_type is 109</td>
</tr>
<tr>
<td>col_type_name_len</td>
<td>Length of user-define type column type name, this field is valid when col_type is 109</td>
</tr>
</tbody>
</table>

Related Topics

- DESCRIPTION_COLUMNS3 Procedure
  This procedure describes the specified column. This is an alternative to DESCRIPTION_COLUMNS Procedure.

154.8.5 DBMS_SQL BFILE_TABLE Table Type

This is a table of BFILE.

Syntax

```
TYPE bfile_table IS TABLE OF BFILE INDEX BY BINARY_INTEGER;
```
154.8.6 DBMS_SQL BINARY_DOUBLE_TABLE Table Type

This is a table of BINARY_DOUBLE.

Syntax

```
TYPE binary_double_table IS TABLE OF BINARY_DOUBLE INDEX BY BINARY_INTEGER;
```

154.8.7 DBMS_SQL BINARY_FLOAT_TABLE Table Type

This is a table of BINARY_FLOAT.

Syntax

```
TYPE binary_float_table IS TABLE OF BINARY_FLOAT INDEX BY BINARY_INTEGER;
```

154.8.8 DBMS_SQL BLOB_TABLE Table Type

This is a table of BLOB.

Syntax

```
TYPE blob_table IS TABLE OF BLOB INDEX BY BINARY_INTEGER;
```

154.8.9 DBMS_SQL CLOB_TABLE Table Type

This is a table of CLOB.

Syntax

```
TYPE clob_table IS TABLE OF CLOB INDEX BY BINARY_INTEGER;
```

154.8.10 DBMS_SQL DATE_TABLE Table Type

This is a table of DATE.

Syntax

```
type date_table IS TABLE OF DATE INDEX BY BINARY_INTEGER;
```

154.8.11 DBMS_SQL DESC_TAB Table Type

This is a table of DESC_REC Record Type.

Syntax

```
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

Related Topics

- DBMS_SQL DESC_REC Record Type
  This record type holds the describe information for a single column in a dynamic query.
154.8.12 DBMS_SQL DESC_TAB2 Table Type

This is a table of DESC_REC2 Record Type.

Syntax

```sql
TYPE desc_tab2 IS TABLE OF desc_rec2 INDEX BY BINARY_INTEGER;
```

Related Topics

- [DBMS_SQL DESC_REC2 Record Type](#)
  
  DESC_REC2 is the element type of the DESC_TAB2 table type and the DESCRIBE_COLUMNS2 Procedure.

154.8.13 DBMS_SQL DESC_TAB3 Table Type

This is a table of DESC_REC3 Record Type.

Syntax

```sql
TYPE desc_tab3 IS TABLE OF desc_rec3 INDEX BY BINARY_INTEGER;
```

Related Topics

- [DBMS_SQL DESC_REC3 Record Type](#)
  
  DESC_REC3 is the element type of the DESC_TAB3 table type and the DESCRIBE_COLUMNS3 Procedure.

154.8.14 DBMS_SQL DESC_TAB4 Table Type

This is a table of DBMS_SQL DESC_REC4 Record Type.

Syntax

```sql
TYPE DESC_TAB4 IS TABLE OF DESC_REC4 INDEX BY BINARY_INTEGER;
```

Related Topics

- [DBMS_SQL DESC_REC4 Record Type](#)
  
  DESC_REC4 is the element type of the DESC_TAB4 table type and the DESCRIBE_COLUMNS3 Procedure.

154.8.15 DBMS_SQL INTERVAL_DAY_TO_SECOND_TABLE Table Type

This is a table of DSINTERVAL_UNCONSTRAINED.

Syntax

```sql
TYPE interval_day_to_second_Table IS TABLE OF DSINTERVAL_UNCONSTRAINED INDEX BY binary_integer;
```
154.8.16 DBMS_SQL INTERVAL_YEAR_TO_MONTH_TABLE Table Type

This is a table of YMINTERVAL_UNCONSTRAINED.

Syntax

```sql
TYPE interval_year_to_month_table IS TABLE OF YMINTERVAL_UNCONSTRAINED
    INDEX BY BINARY_INTEGER;
```

154.8.17 DBMS_SQL NUMBER_TABLE Table Type

This is a table of NUMBER.

Syntax

```sql
TYPE number_table IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

154.8.18 DBMS_SQL TIME_TABLE Table Type

This is a table of TIME_UNCONSTRAINED.

Syntax

```sql
TYPE time_table IS TABLE OF TIME_UNCONSTRAINED INDEX BY BINARY_INTEGER;
```

154.8.19 DBMS_SQL TIME_WITH_TIME_ZONE_TABLE Table Type

This is a table of TIME_TZ_UNCONSTRAINED.

Syntax

```sql
TYPE time_with_time_zone_table IS TABLE OF TIME_TZ_UNCONSTRAINED
    INDEX BY BINARY_INTEGER;
```

154.8.20 DBMS_SQL TIMESTAMP_TABLE Table Type

This is a table of TIMESTAMP_UNCONSTRAINED.

Syntax

```sql
TYPE timestamp_table IS TABLE OF TIMESTAMP_UNCONSTRAINED INDEX BY BINARY_INTEGER;
```

154.8.21 DBMS_SQL TIMESTAMP_WITH_LTZ_TABLE Table Type

This is a table of TIMESTAMP_LTZ_UNCONSTRAINED

Syntax

```sql
TYPE timestamp_with_ltz_table IS TABLE OF
    TIMESTAMP_LTZ_UNCONSTRAINED INDEX BY binary_integer;
```
154.8.22 DBMS_SQL TIMESTAMP_WITH_TIME_ZONE_TABLE Table Type

This is a table of TIMESTAMP_TZ_UNCONSTRAINED.

Syntax

    TYPE timestamp_with_time_zone_table IS TABLE OF
        TIMESTAMP_TZ_UNCONSTRAINED INDEX BY binary_integer;

154.8.23 DBMS_SQL UROWID_TABLE Table Type

This is a table of UROWID.

Syntax

    TYPE urowid_table IS TABLE OF UROWID INDEX BY BINARY_INTEGER;

154.8.24 DBMS_SQL VARCHAR2_TABLE Table Type

This is table of VARCHAR2(2000).

Syntax

    TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

154.8.25 DBMS_SQL VARCHAR2A Table Type

This is table of VARCHAR2(32767).

Syntax

    TYPE varchar2a IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;

154.8.26 DBMS_SQL VARCHAR2S Table Type

This is table of VARCHAR2(256).

*Note:*

This type has been superseded by the VARCHAR2A Table Type. Although it is currently retained for backward compatibility of legacy code, it is in the process of deprecation and will be de-supported in a future release.

Syntax

    TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
154.9 Summary of DBMS_SQL Subprograms

This table lists the DBMS_SQL subprograms and briefly describes them.

Table 154-6  DBMS_SQL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIND_ARRAY Procedures</td>
<td>Binds a given value to a given collection.</td>
</tr>
<tr>
<td>BIND_VARIABLE Procedures</td>
<td>Binds a given value to a given variable.</td>
</tr>
<tr>
<td>BIND_VARIABLE_PKG Procedure</td>
<td>Binds a given value to a given package variable.</td>
</tr>
<tr>
<td>CLOSE_CURSOR Procedure</td>
<td>Closes given cursor and frees memory.</td>
</tr>
<tr>
<td>COLUMN_VALUE Procedure</td>
<td>Returns value of the cursor element for a given position in a cursor.</td>
</tr>
<tr>
<td>COLUMN_VALUE_LONG Procedure</td>
<td>Returns a selected part of a LONG column, that has been defined using DEFINE_COLUMN_LONG.</td>
</tr>
<tr>
<td>DEFINE_ARRAY Procedure</td>
<td>Defines a collection to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>DEFINE_COLUMN Procedures</td>
<td>Defines a column to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>DEFINE_COLUMN_CHAR Procedure</td>
<td>Defines a column of type CHAR to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>DEFINE_COLUMN_LONG Procedure</td>
<td>Defines a LONG column to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>DEFINE_COLUMN_RAW Procedure</td>
<td>Defines a column of type RAW to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>DEFINE_COLUMN_ROWID Procedure</td>
<td>Defines a column of type ROWID to be selected from the given cursor, used only with SELECT statements.</td>
</tr>
<tr>
<td>DESCRIBE_COLUMNS Procedure</td>
<td>Describes the columns for a cursor opened and parsed through DBMS_SQL.</td>
</tr>
<tr>
<td>DESCRIBE_COLUMNS2 Procedure</td>
<td>Describes the specified column, an alternative to DESCRIBE_COLUMNS Procedure.</td>
</tr>
<tr>
<td>DESCRIBE_COLUMNS3 Procedure</td>
<td>Describes the specified column, an alternative to DESCRIBE_COLUMNS Procedure.</td>
</tr>
<tr>
<td>EXECUTE Function</td>
<td>Executes a given cursor.</td>
</tr>
<tr>
<td>EXECUTE_AND_FETCH Function</td>
<td>Executes a given cursor and fetch rows.</td>
</tr>
<tr>
<td>FETCH_ROWS Function</td>
<td>Fetches a row from a given cursor.</td>
</tr>
<tr>
<td>GET_NEXT_RESULT Procedures</td>
<td>Gets the statement of the next result returned to the caller of the recursive statement or, if this callee sets itself as the client for the recursive statement, the next result returned to this callee as client.</td>
</tr>
<tr>
<td>IS_OPEN Function</td>
<td>Returns TRUE if given cursor is open.</td>
</tr>
<tr>
<td>LAST_ERROR_POSITION Function</td>
<td>Returns byte offset in the SQL statement text where the error occurred.</td>
</tr>
<tr>
<td>LAST_ROW_COUNT Function</td>
<td>Returns cumulative count of the number of rows fetched</td>
</tr>
<tr>
<td>LAST_ROW_ID Function</td>
<td>Returns ROWID of last row processed.</td>
</tr>
</tbody>
</table>
Table 154-6  (Cont.) DBMS_SQL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_SQL_FUNCTION_CODE Function</td>
<td>Returns SQL function code for statement.</td>
</tr>
<tr>
<td>OPEN_CURSOR Functions</td>
<td>Returns cursor ID number of new cursor.</td>
</tr>
<tr>
<td>PARSE Procedures</td>
<td>Parses given statement.</td>
</tr>
<tr>
<td>RETURN_RESULT Procedures</td>
<td>Returns the result of an executed statement to the client application.</td>
</tr>
<tr>
<td>TO_CURSOR_NUMBER Function</td>
<td>Takes an OPENed strongly or weakly-typed ref cursor and transforms it into a DBMS_SQL cursor number.</td>
</tr>
<tr>
<td>TO_REFCURSOR Function</td>
<td>Takes an OPENed, PARSeEd, and EXECUTEEd cursor and transforms/migrates it into a PL/SQL manageable REFCURSOR (a weakly-typed cursor) that can be consumed by PL/SQL native dynamic SQL switched to use native dynamic SQL.</td>
</tr>
<tr>
<td>VARIABLE_VALUE Procedures</td>
<td>Returns value of named variable for given cursor.</td>
</tr>
<tr>
<td>VARIABLE_VALUE_PKG Procedure</td>
<td>Returns value of named variable for given cursor. It is used to return the values of bind variables inside PL/SQL blocks or DML statements with returning clause for a declared package. The type of the variable must be declared in the package specification.</td>
</tr>
</tbody>
</table>

154.9.1 BIND_ARRAY Procedures

This procedure binds a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement.

Syntax

```sql
DBMS_SQL.BIND_ARRAY (  
  c IN INTEGER,  
  name IN VARCHAR2,  
  table_variable IN <datatype>  
, index1 IN INTEGER,  
  index2 IN INTEGER ) );
```

Where the `table_variable` and its corresponding `datatype` can be any one of the following matching pairs:

- `<clob_tab>` Clob_Table
- `<bflt_tab>` Binary_Float_Table
- `<bdbl_tab>` Binary_Double_Table
- `<blob_tab>` Blob_Table
- `<bfile_tab>` Bfile_Table
- `<date_tab>` Date_Table
- `<num_tab>` Number_Table
- `<urowid_tab>` Urowid_Table
- `<vchr2_tab>` Varchar2_Table
- `<tm_tab>` Time_Table
- `<ttz_tab>` Time_With_Time_Zone_Table
- `<tms_tab>` Timestamp_Table
- `<ttz_tab>` Timestamp_With_Ttz_Table
- `<tstz_tab>` Timestamp_With_Time_Zone_Table
- `<tstz_tab>` Timestamp_With_Time_Zone_Table
Notice that the **BIND_ARRAY** procedure is overloaded to accept different datatype.

### Parameters

**Table 154-7  **BIND_ARRAY** Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to which you want to bind a value.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the collection in the statement.</td>
</tr>
<tr>
<td>table_variable</td>
<td>Local variable that has been declared as <code>&lt;datatype&gt;</code>.</td>
</tr>
<tr>
<td>index1</td>
<td>Index for the table element that marks the lower bound of the range.</td>
</tr>
<tr>
<td>index2</td>
<td>Index for the table element that marks the upper bound of the range.</td>
</tr>
</tbody>
</table>

### Usage Notes

For binding a range, the table must contain the elements that specify the range — `tab(index1)` and `tab(index2)` — but the range does not have to be dense. Index1 must be less than or equal to index2. All elements between `tab(index1)` and `tab(index2)` are used in the bind.

If you do not specify indexes in the bind call, and two different binds in a statement specify tables that contain a different number of elements, then the number of elements actually used is the minimum number between all tables. This is also the case if you specify indexes — the minimum range is selected between the two indexes for all tables.

Not all bind variables in a query have to be array binds. Some can be regular binds and the same value are used for each element of the collections in expression evaluations (and so forth).

### Bulk Array Binds

Bulk selects, inserts, updates, and deletes can enhance the performance of applications by bundling many calls into one. The **DBMS_SQL** package lets you work on collections of data using the PL/SQL table type.

*Table items* are unbounded homogeneous collections. In persistent storage, they are like other relational tables and have no intrinsic ordering. But when a table item is brought into the workspace (either by querying or by navigational access of persistent data), or when it is created as the value of a PL/SQL variable or parameter, its elements are given subscripts that can be used with array-style syntax to get and set the values of elements.

The subscripts of these elements need not be dense, and can be any number including negative numbers. For example, a table item can contain elements at locations -10, 2, and 7 only.

When a table item is moved from transient workspace to persistent storage, the subscripts are not stored; the table item is unordered in persistent storage.

At bind time the table is copied out from the PL/SQL buffers into local **DBMS_SQL** buffers (the same as for all scalar types) and then the table is manipulated from the local **DBMS_SQL** buffers. Therefore, if you change the table after the bind call, then that change does not affect the way the execute acts.
Types for Scalar and LOB Collections

You can declare a local variable as one of the following table-item types, which are defined as public types in DBMS_SQL.

- TYPE `binary_double_table` IS TABLE OF BINARY_DOUBLE INDEX BY BINARY_INTEGER;
- TYPE `binary_float_table` IS TABLE OF BINARY_FLOAT INDEX BY BINARY_INTEGER;
- TYPE `bfile_table` IS TABLE OF BFILE INDEX BY BINARY_INTEGER;
- TYPE `blob_table` IS TABLE OF BLOB INDEX BY BINARY_INTEGER;
- TYPE `clob_table` IS TABLE OF CLOB INDEX BY BINARY_INTEGER;
- TYPE `date_table` IS TABLE OF DATE INDEX BY BINARY_INTEGER;
- TYPE `interval_day_to_second_table` IS TABLE OF dsinterval_unconstrained INDEX BY BINARY_INTEGER;
- TYPE `interval_year_to_MONTH_table` IS TABLE OF yminterval_unconstrained INDEX BY BINARY_INTEGER;
- TYPE `number_table` IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
- TYPE `time_table` IS TABLE OF time_unconstrained INDEX BY BINARY_INTEGER;
- TYPE `time_with_time_zone_table` IS TABLE OF time_tz_unconstrained INDEX BY BINARY_INTEGER;
- TYPE `timestamp_table` IS TABLE OF timestamp_unconstrained INDEX BY BINARY_INTEGER;
- TYPE `timestamp_with_ltz_table` IS TABLE OF timestamp_ltz_unconstrained INDEX BY BINARY_INTEGER;
- TYPE `timestamp_with_time_zone_table` IS TABLE OF timestamp_tz_unconstrained INDEX BY BINARY_INTEGER;
- TYPE `urowid_table` IS TABLE OF UROWID INDEX BY BINARY_INTEGER;
- TYPE `varchar2_table` IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

Example 154-1  Examples Using Bulk DML

This series of examples shows how to use bulk array binds (table items) in the SQL DML statements INSERT, UPDATE and DELETE.

Here is an example of a bulk INSERT statement that demonstrates adding seven new employees to the `emp` table:

```sql
DECLARE
    stmt VARCHAR2(200);
    empno_array    DBMS_SQL.NUMBER_TABLE;
    emname_array   DBMS_SQL.VARCHAR2_TABLE;
    jobs_array     DBMS_SQL.VARCHAR2_TABLE;
    mgr_array      DBMS_SQL.NUMBER_TABLE;
    hiredate_array DBMS_SQL.VARCHAR2_TABLE;
    sal_array      DBMS_SQL.NUMBER_TABLE;
    comm_array     DBMS_SQL.NUMBER_TABLE;
    deptno_array   DBMS_SQL.NUMBER_TABLE;
    c              NUMBER;
    dummy          NUMBER;
BEGIN
    empno_array(1):= 9001;
    empno_array(2):= 9002;
    empno_array(3):= 9003;
    stmt := 'INSERT INTO emp (empno, empname, jobs, mgr, hiredate, sal, comm, deptno) VALUES (';
    FOR i IN 1..7
        stmt := stmt || (i <= 3 => '9001, ',
                        i = 4 => '9002, ',
                        i = 5 => '9003, ',
                        i <= 7 => '
                        '); stmt := stmt || '9001)';
    stmt := stmt || '); EXECUTE IMMEDIATE stmt;';
END;
```
empno_array(4):= 9004;
empno_array(5):= 9005;
empno_array(6):= 9006;
empno_array(7):= 9007;

empname_array(1) := 'Dopey';
empname_array(2) := 'Grumpy';
empname_array(3) := 'Doc';
empname_array(4) := 'Happy';
empname_array(5) := 'Bashful';
empname_array(6) := 'Sneezy';
empname_array(7) := 'Sleepy';

jobs_array(1) := 'Miner';
jobs_array(2) := 'Miner';
jobs_array(3) := 'Miner';
jobs_array(4) := 'Miner';
jobs_array(5) := 'Miner';
jobs_array(6) := 'Miner';
jobs_array(7) := 'Miner';

mgr_array(1) := 9003;
mgr_array(2) := 9003;
mgr_array(3) := 9003;
mgr_array(4) := 9003;
mgr_array(5) := 9003;
mgr_array(6) := 9003;
mgr_array(7) := 9003;

hiredate_array(1) := '06-DEC-2006';
hiredate_array(2) := '06-DEC-2006';
hiredate_array(3) := '06-DEC-2006';
hiredate_array(4) := '06-DEC-2006';
hiredate_array(5) := '06-DEC-2006';
hiredate_array(6) := '06-DEC-2006';
hiredate_array(7) := '06-DEC-2006';

sal_array(1):= 1000;
sal_array(2):= 1000;
sal_array(3):= 1000;
sal_array(4):= 1000;
sal_array(5):= 1000;
sal_array(6):= 1000;
sal_array(7):= 1000;

comm_array(1):= 0;
comm_array(2):= 0;
comm_array(3):= 0;
comm_array(4):= 0;
comm_array(5):= 0;
comm_array(6):= 0;
comm_array(7):= 0;

depnto_array(1):= 11;
depnto_array(2):= 11;
depnto_array(3):= 11;
depnto_array(4):= 11;
depnto_array(5):= 11;
depnto_array(6):= 11;
depnto_array(7):= 11;
Here is an example of a bulk UPDATE statement that demonstrates updating salaries for four existing employees in the emp table:

```
DECLARE
    stmt VARCHAR2(200);
    empno_array     DBMS_SQL.NUMBER_TABLE;
    salary_array    DBMS_SQL.NUMBER_TABLE;
    c               NUMBER;
    dummy           NUMBER;
BEGIN
    empno_array(1) := 7369;
    empno_array(2) := 7876;
    empno_array(3) := 7900;
    empno_array(4) := 7934;
    salary_array(1) := 10000;
    salary_array(2) := 10000;
    salary_array(3) := 10000;
    salary_array(4) := 10000;
    stmt := 'update emp set sal = :salary_array
        WHERE empno = :num_array';
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_ARRAY(c, ':num_array', empno_array);
    DBMS_SQL.BIND_ARRAY(c, ':salary_array', salary_array);
    dummy := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.CLOSE_CURSOR(c);
EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
        DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
SHOW ERRORS;
```
In a `DELETE` statement, for example, you could bind an array in the `WHERE` clause and have the statement be run for each element in the array:

```sql
BEGIN
    dept_no_array(1) := 10; dept_no_array(2) := 20;
    dept_no_array(3) := 30; dept_no_array(4) := 40;
    dept_no_array(5) := 30; dept_no_array(6) := 40;
    stmt := 'delete from emp where deptno = :dept_array';
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_ARRAY(c, ':dept_array', dept_no_array, 1, 4);
    dummy := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.CLOSE_CURSOR(c);
EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
        DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
```

In the preceding example, only elements 1 through 4 are used as specified by the `BIND_ARRAY` call. Each element of the array potentially deletes a large number of employees from the database.

154.9.2 BIND_VARIABLE Procedures

These procedures bind a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement.

**Syntax**

```sql
DBMS_SQL.BIND_VARIABLE (
    c IN INTEGER,
    name IN VARCHAR2,
    value IN <datatype>);
```

Where `<datatype>` can be any one of the following types:

- ADT (user-defined object types)
- BINARY_DOUBLE
- BINARY_FLOAT
- BFILE
- BLOB
- BOOLEAN
- CLOB CHARACTER SET ANY_CS
- DATE
- DSINTERVAL_UNCONSTRAINED
- NESTED table
- NUMBER
- OPAQUE types
REF
TIME_UNCONSTRAINED
TIME_TZ_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
VARRAY
YMINTERVAL_UNCONSTRAINED

Notice that BIND_VARIABLE is overloaded to accept different datatype.

The following syntax is also supported for BIND_VARIABLE. The square brackets [] indicate an optional parameter for the BIND_VARIABLE procedure.

\[
\text{DBMS_SQL.BIND_VARIABLE (}
\begin{align*}
c & \quad \text{IN INTEGER}, \\
\text{name} & \quad \text{IN VARCHAR2}, \\
\text{value} & \quad \text{IN VARCHAR2 CHARACTER SET ANY_CS [,out_value_size IN INTEGER]} \\
\end{align*}
\]

To bind CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

\[
\text{DBMS_SQL.BIND_VARIABLE_CHAR (}
\begin{align*}
c & \quad \text{IN INTEGER}, \\
\text{name} & \quad \text{IN VARCHAR2}, \\
\text{value} & \quad \text{IN CHAR CHARACTER SET ANY_CS [,out_value_size IN INTEGER]} \\
\end{align*}
\]

\[
\text{DBMS_SQL.BIND_VARIABLE_RAW (}
\begin{align*}
c & \quad \text{IN INTEGER}, \\
\text{name} & \quad \text{IN VARCHAR2}, \\
\text{value} & \quad \text{IN RAW [,out_value_size IN INTEGER]} \\
\end{align*}
\]

\[
\text{DBMS_SQL.BIND_VARIABLE_ROWID (}
\begin{align*}
c & \quad \text{IN INTEGER}, \\
\text{name} & \quad \text{IN VARCHAR2}, \\
\text{value} & \quad \text{IN ROWID} \\
\end{align*}
\]

Pragmas

\[
\text{pragma restrict_references(bind_variable,WNDS);} \\
\]

Parameters

Table 154-8  BIND_VARIABLE Procedures Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to which you want to bind a value.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the variable in the statement. The length of the bind variable name must be &lt;=30 bytes.</td>
</tr>
<tr>
<td>value</td>
<td>Value that you want to bind to the variable in the cursor. For IN and IN/OUT variables, the value has the same type as the type of the value being passed in for this parameter.</td>
</tr>
</tbody>
</table>
### Table 154-8 (Cont.) BIND_VARIABLE Procedures Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_value_size</td>
<td>Maximum expected OUT value size, in bytes, for the VARCHAR2, RAW, CHAR OUT or IN/OUT variable. If no size is given, then the length of the current value is used. This parameter must be specified if the value parameter is not initialized.</td>
</tr>
</tbody>
</table>

### Usage Notes

If the variable is an IN or IN/OUT variable or an IN collection, then the given bind value must be valid for the variable or array type. Bind values for OUT variables are ignored.

The bind variables or collections of a SQL statement are identified by their names. When binding a value to a bind variable or bind array, the string identifying it in the statement must contain a leading colon, as shown in the following example:

```sql
SELECT emp_name FROM emp WHERE SAL > :X;
```

For this example, the corresponding bind call would look similar to

```sql
BIND_VARIABLE(cursor_name, ':X', 3500);
```

or

```sql
BIND_VARIABLE (cursor_name, 'X', 3500);
```

### 154.9.3 BIND_VARIABLE_PKG Procedure

This procedures binds a variable given value or set of values to a given variable in a cursor, based on the name of the variable in the statement. The type of the variable must be declared in the package specification. Bulk operations are not supported for these types.

#### Syntax

```sql
DBMS_SQL.BIND_VARIABLE_PKG (  
c              IN INTEGER,  
name           IN VARCHAR2,  
value          IN <datatype>);
```

Where `<datatype>` can be any one of the following data types:

- RECORD
- VARRAY
- NESTED TABLE
- INDEX BY PLS_INTEGER TABLE
- INDEX BY BINARY_INTEGER TABLE
Table 154-9  BIND_VARIABLE_PKG Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor from which to get the values.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the variable in the statement for which you are retrieving the value.</td>
</tr>
</tbody>
</table>
| value     | - Single row option: Returns the value of the variable for the specified position. Oracle raises the exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to BIND_VARIABLE_PKG.  
- Array option: Local variable that has been declared <table_type> |

Example 154-2  Dynamic SQL using DBMS_SQL.BIND_VARIABLE_PKG to Bind a Package Variable

The variables types are declared in the package specification. The BIND_VARIABLE_PKG is used to bind the variable v1 in the cursor SQL statement.

```sql
CREATE OR REPLACE PACKAGE ty_pkg AS
    TYPE rec IS RECORD ( n1 NUMBER, n2 NUMBER);
    TYPE trec IS TABLE OF rec INDEX BY BINARY_INTEGER;
    TYPE trect IS TABLE OF NUMBER;
    TYPE trecv IS VARRAY(100) OF NUMBER;
END ty_pkg;
/
CREATE OR REPLACE PROCEDURE dyn_sql_ibbi AS
    dummy NUMBER;
    cur NUMBER;
    v1 ty_pkg.trec;
    str VARCHAR2(3000);
    n1 NUMBER;
    n2 NUMBER;
BEGIN
    FOR i in 1..3 LOOP
        v1(i).n1 := i*10;
        v1(i).n2 := i*20;
    END LOOP;
    str := 'SELECT * FROM TABLE(:v1)' ;
    cur := DBMS_SQL.OPEN_CURSOR();
    DBMS_SQL.PARSE(cur, str, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE_PKG(cur, ':v1', v1);
    dummy := DBMS_SQL.EXECUTE(cur);
    DBMS_SQL.DEFINE_COLUMN(cur, 1, n1);
    DBMS_SQL.DEFINE_COLUMN(cur, 2, n2);
    LOOP
        IF DBMS_SQL.FETCH_ROWS(cur) > 0 THEN
            -- get column values of the row
        END LOOP;
    END LOOP;
END dyn_sql_ibbi;
/
```

Chapter 154  Summary of DBMS_SQL Subprograms
DBMS_SQL.COLUMN_VALUE(cur, 1, n1);
DBMS_SQL.COLUMN_VALUE(cur, 2, n2);
DBMS_OUTPUT.PUT_LINE('n1 = '||n1||' n2 = '||n2);
ELSE
  -- No more rows
  EXIT;
END IF;
END LOOP;
DBMS_SQL.CLOSE_CURSOR(cur);
END dyn_sql_ibbi;
/
EXEC dyn_sql_ibbi;
n1 = 10 n2 = 20
n1 = 20 n2 = 40
n1 = 30 n2 = 60

154.9.4 CLOSE_CURSOR Procedure
This procedure closes a given cursor.

Syntax

DBMS_SQL.CLOSE_CURSOR (    c    IN OUT INTEGER);

Pragmas

pragma restrict_references(close_cursor,RNDS,WNDS);

Parameters

Table 154-10  CLOSE_CURSOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>IN</td>
<td>ID number of the cursor that you want to close.</td>
</tr>
<tr>
<td>c</td>
<td>OUT</td>
<td>Cursor is set to null. After you call CLOSE_CURSOR, the memory allocated to the cursor is released and you can no longer fetch from that cursor.</td>
</tr>
</tbody>
</table>

154.9.5 COLUMN_VALUE Procedure
This procedure returns the value of the cursor element for a given position in a given cursor. This procedure is used to access the data fetched by calling FETCH_ROWS.

Syntax

DBMS_SQL.COLUMN_VALUE (    c    IN INTEGER,
position    IN INTEGER,
value      OUT <datatype>
[,column_error OUT NUMBER]
[,actual_length OUT INTEGER]);
Where square brackets [ ] indicate optional parameters and <datatype> can be any one of the following types:

BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_TZ_UNCONSTRAINED
TIME_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED
user-defined object types
collections (VARRAYs and nested tables)
REFs
Opaque types

For variables containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

DBMS_SQL.COLUMN_VALUE_CHAR (  
c              IN  INTEGER,
position        IN  INTEGER,
value           OUT CHAR CHARACTER SET ANY_CS
[,column_error    OUT NUMBER]
[,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_RAW (  
c              IN  INTEGER,
position        IN  INTEGER,
value           OUT RAW
[,column_error    OUT NUMBER]
[,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_ROWID (  
c              IN  INTEGER,
position        IN  INTEGER,
value           OUT ROWID
[,column_error    OUT NUMBER]
[,actual_length   OUT INTEGER]);

The following syntax enables the COLUMN_VALUE procedure to accommodate bulk operations:

DBMS_SQL.COLUMN_VALUE(  
c                 IN             INTEGER,
position          IN             INTEGER,
<param_name>      IN OUT NOCOPY  <table_type>);

Where the <param_name> and its corresponding <table_type> can be any one of these matching pairs:

bdbl_tab Binary_Double_Table
bflt_tab Binary_Float_Table
Pragmas

```sql
pragma restrict_references(column_value,RNDS,WNDS);
```

Parameters

### Table 154-11 COLUMN_VALUE Procedure Parameters (Single Row)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor from which you are fetching the values.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the cursor.</td>
</tr>
<tr>
<td></td>
<td>The first column in a statement has position 1.</td>
</tr>
<tr>
<td>value</td>
<td>Returns the value at the specified column.</td>
</tr>
<tr>
<td></td>
<td>Oracle raises exception <code>ORA-06562, inconsistent_type</code>, if the type of this</td>
</tr>
<tr>
<td></td>
<td>output parameter differs from the actual type of the value, as defined by</td>
</tr>
<tr>
<td></td>
<td>the call to <code>DEFINE_COLUMN</code>.</td>
</tr>
<tr>
<td>column_error</td>
<td>Returns any error code for the specified column value.</td>
</tr>
<tr>
<td>actual_length</td>
<td>The actual length, before any truncation, of the value in the specified</td>
</tr>
<tr>
<td></td>
<td>column.</td>
</tr>
</tbody>
</table>

### Table 154-12 COLUMN_VALUE Procedure Parameters (Bulk)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor from which you are fetching the values.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the cursor.</td>
</tr>
<tr>
<td></td>
<td>The first column in a statement has position 1.</td>
</tr>
<tr>
<td>&lt;param_name&gt;</td>
<td>Local variable that has been declared <code>&lt;table_type&gt;</code>. <code>&lt;param_name&gt;</code> is an <code>IN OUT NOCOPY</code> parameter for bulk operations.</td>
</tr>
<tr>
<td></td>
<td>For bulk operations, the subprogram appends the new elements at the appropriate (implicitly maintained) index. For instance if on utilizing the <code>DEFINE_ARRAY Procedure</code> a batch size (the <code>cnt</code> parameter) of 10 rows was specified and a start index (<code>lower_bound</code>) of 1 was specified, then the first call to this subprogram after calling the <code>FETCH_ROWS Function</code> will populate elements at index 1..10, and the next call will populate elements 11..20, and so on.</td>
</tr>
</tbody>
</table>
Exceptions

INCONSISTENT_TYPE (ORA-06562) is raised if the type of the given OUT parameter value is different from the actual type of the value. This type was the given type when the column was defined by calling procedure DEFINE_COLUMN.

154.9.6 COLUMN_VALUE_LONG Procedure

This procedure gets part of the value of a long column.

Syntax

DBMS_SQL.COLUMN_VALUE_LONG (  
c            IN  INTEGER,  
position     IN  INTEGER,  
length       IN  INTEGER,  
offset       IN  INTEGER,  
value        OUT VARCHAR2,  
value_length OUT INTEGER);

Pragmas

pragma restrict_references(column_value_long,RNDS,WNDS);

Parameters

Table 154-13  COLUMN_VALUE_LONG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor ID number of the cursor from which to get the value.</td>
</tr>
<tr>
<td>position</td>
<td>Position of the column of which to get the value.</td>
</tr>
<tr>
<td>length</td>
<td>Number of bytes of the long value to fetch.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset into the long field for start of fetch.</td>
</tr>
<tr>
<td>value</td>
<td>Value of the column as a VARCHAR2.</td>
</tr>
<tr>
<td>value_length</td>
<td>Number of bytes actually returned in value.</td>
</tr>
</tbody>
</table>

154.9.7 DEFINE_ARRAY Procedure

This procedure defines the collection for column into which you want to fetch rows (with a FETCH_ROWS call). This procedure lets you do batch fetching of rows from a single SELECT statement. A single fetch call brings over a number of rows into the PL/SQL aggregate object.

When you fetch the rows, they are copied into DBMS_SQL buffers until you run a COLUMN_VALUE call, at which time the rows are copied into the table that was passed as an argument to the COLUMN_VALUE call.

Scalar and LOB Types for Collections

You can declare a local variable as one of the following table-item types, and then fetch any number of rows into it using DBMS_SQL. (These are the same types as you can specify for the BIND_ARRAY procedure.)
TYPE binary_double_table
  IS TABLE OF BINARY_DOUBLE  INDEX BY BINARY_INTEGER;

TYPE binary_float_table
  IS TABLE OF BINARY_FLOAT   INDEX BY BINARY_INTEGER;

TYPE bfile_table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;

TYPE blob_table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;

TYPE clob_table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;

TYPE date_table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;

TYPE interval_day_to_second_Table
  IS TABLE OF dsinterval_unconstrained
    INDEX BY BINARY_INTEGER;

TYPE interval_year_to_MONTH_Table
  IS TABLE OF yminterval_unconstrained
    INDEX BY BINARY_INTEGER;

TYPE number_table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;

TYPE time_table     IS TABLE OF time_unconstrained
    INDEX BY BINARY_INTEGER;

TYPE time_with_time_zone_table
  IS TABLE OF time_tz_unconstrained
    INDEX BY BINARY_INTEGER;

TYPE timestamp_table
  IS TABLE OF timestamp_unconstrained
    INDEX BY BINARY_INTEGER;

TYPE timestamp_with_ltz_Table
  IS TABLE OF timestamp_ltz_unconstrained
    INDEX BY BINARY_INTEGER;

TYPE timestamp_with_time_zone_Table
  IS TABLE OF timestamp_tz_unconstrained
    INDEX BY BINARY_INTEGER;

TYPE urowid_table   IS TABLE OF UROWID         INDEX BY BINARY_INTEGER;

TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

Syntax

DBMS_SQL.DEFINE_ARRAY (c, position, <table_variable>, cnt, lower_bnd, IN <datatype>);

Where <table_variable> and its corresponding <datatype> can be any one of the following matching pairs, DEFINE_ARRAY being overloaded to accept different datatypes:

<table>
<thead>
<tr>
<th>&lt;table_variable&gt;</th>
<th>&lt;datatype&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>clob_tab&gt;</td>
<td>Clob_Table</td>
</tr>
<tr>
<td>bflt_tab&gt;</td>
<td>Binary_Float_Table</td>
</tr>
<tr>
<td>bdbl_tab&gt;</td>
<td>Binary_Double_Table</td>
</tr>
<tr>
<td>blob_tab&gt;</td>
<td>Blob_Table</td>
</tr>
<tr>
<td>bfile_tab&gt;</td>
<td>Bfile_Table</td>
</tr>
<tr>
<td>date_tab&gt;</td>
<td>Date_Table</td>
</tr>
<tr>
<td>num_tab&gt;</td>
<td>Number_Table</td>
</tr>
<tr>
<td>urowid_tab&gt;</td>
<td>Urowid_Table</td>
</tr>
<tr>
<td>vchr2_tab&gt;</td>
<td>Varchar2_Table</td>
</tr>
<tr>
<td>tm_tab&gt;</td>
<td>Time_Table</td>
</tr>
<tr>
<td>ttz_tab&gt;</td>
<td>Time_With_Time_Zone_Table</td>
</tr>
<tr>
<td>tms_tab&gt;</td>
<td>Timestamp_Table</td>
</tr>
<tr>
<td>tstz_tab&gt;</td>
<td>Timestamp_With_Ltz_Table</td>
</tr>
<tr>
<td>tstmz_tab&gt;</td>
<td>Timestamp_With_Time_Zone_Table</td>
</tr>
<tr>
<td>ids_tab&gt;</td>
<td>Interval_Day_To_Second_Table</td>
</tr>
<tr>
<td>ids Tab&gt;</td>
<td>Interval_Day_To_Second_Table</td>
</tr>
<tr>
<td>iym_tab&gt;</td>
<td>Interval_Year_To_Month_Table</td>
</tr>
</tbody>
</table>
Pragmas

pragma restrict_references(define_array,RNDS,WNDS);

The subsequent FETCH_ROWS call fetch "count" rows. When the COLUMN_VALUE call is made, these rows are placed in positions lower_bnd, lower_bnd+1, lower_bnd+2, and so on. While there are still rows coming, the user keeps issuing FETCH_ROWS/COLUMN_VALUE calls. The rows keep accumulating in the table specified as an argument in the COLUMN_VALUE call.

Parameters

Table 154-14  DEFINE_ARRAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to which you want to bind an array.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the array being defined.</td>
</tr>
<tr>
<td></td>
<td>The first column in a statement has position 1.</td>
</tr>
<tr>
<td>table_variable</td>
<td>Local variable that has been declared as &lt;datatype&gt;.</td>
</tr>
<tr>
<td>cnt</td>
<td>Number of rows that must be fetched.</td>
</tr>
<tr>
<td>lower_bnd</td>
<td>Results are copied into the collection, starting at this lower bound index.</td>
</tr>
</tbody>
</table>

Usage Notes

The count (cnt) must be an integer greater than zero; otherwise an exception is raised. The lower_bnd can be positive, negative, or zero. A query on which a DEFINE_ARRAY call was issued cannot contain array binds.

Examples

PROCEDURE BULK_PLSQL(deptid NUMBER)
    TYPE namelist IS TABLE OF employees.last_name%TYPE;
    TYPE sallist IS TABLE OF employees.salary%TYPE;
    names    namelist;
    sals     sallist;
    c        NUMBER;
    r        NUMBER;
    sql_stmt VARCHAR2(32767) :=
        'SELECT last_name, salary FROM employees WHERE department_id = :b1';

BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, sql_stmt, dbms_sql.native);
    DBMS_SQL.BIND_VARIABLE(c, 'b1', deptid);
    DBMS_SQL.DEFINE_ARRAY(c, 1, names, 5);
    DBMS_SQL.DEFINE_ARRAY(c, 2, sals, 5);
    r := DBMS_SQL.EXECUTE(c);
    LOOP
        r := DBMS_SQL.FETCH_ROWS(c);
        DBMS_SQL.COLUMN_VALUE(c, 1, names);
Example 154-3  Example: Defining an Array

The following examples show how to use the DEFINE_ARRAY procedure:

```plsql
DECLARE
    c       NUMBER;
    d       NUMBER;
    n_tab   DBMS_SQL.NUMBER_TABLE;
    indx    NUMBER := -10;
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'select n from t order by 1', DBMS_SQL.NATIVE);
    DBMS_SQL.DEFINE_ARRAY(c, 1, n_tab, 10, indx);
    d := DBMS_SQL.EXECUTE(c);
    LOOP
        d := DBMS_SQL.FETCH_ROWS(c);
        DBMS_SQL.COLUMN_VALUE(c, 1, n_tab);
        EXIT WHEN d != 10;
    END LOOP;
    DBMS_SQL.CLOSE_CURSOR(c);
EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
        DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
```

Each time the preceding example calls FETCH_ROWS Function, it fetches 10 rows that are kept in DBMS_SQL buffers. When the COLUMN_VALUE Procedure is called, those rows move into the PL/SQL table specified (in this case n_tab), at positions -10 to -1, as specified in the DEFINE statements. When the second batch is fetched in the loop, the rows go to positions 0 to 9; and so on.

A current index into each array is maintained automatically. This index is initialized to "indx" at EXECUTE time and is updated every time COLUMN_VALUE is called. If you reexecute at any point, then the current index for each DEFINE is reinitialized to "indx".

In this way the entire result of the query is fetched into the table. When FETCH_ROWS cannot fetch 10 rows, it returns the number of rows actually fetched (if no rows could be fetched, then it returns zero) and exits the loop.
Here is another example of using the `DEFINE_ARRAY` procedure:

Consider a table `MULTI_TAB` defined as:

```sql
CREATE TABLE multi_tab (num NUMBER, 
dat1 DATE, 
var VARCHAR2(24), 
dat2 DATE)
```

To select everything from this table and move it into four PL/SQL tables, you could use the following simple program:

```sql
DECLARE 
c       NUMBER;
d       NUMBER;
n_tab  DBMS_SQL.NUMBER_TABLE;
d_tab1 DBMS_SQL.DATE_TABLE;
v_tab  DBMS_SQL.VARCHAR2_TABLE;
d_tab2 DBMS_SQL.DATE_TABLE;
indx NUMBER := 10;
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'select * from multi_tab order by 1', DBMS_SQL.NATIVE);
    DBMS_SQL.DEFINE_ARRAY(c, 1, n_tab,  5, indx);
    DBMS_SQL.DEFINE_ARRAY(c, 2, d_tab1, 5, indx);
    DBMS_SQL.DEFINE_ARRAY(c, 3, v_tab,  5, indx);
    DBMS_SQL.DEFINE_ARRAY(c, 4, d_tab2, 5, indx);
    d := DBMS_SQL.EXECUTE(c);
    LOOP 
        d := DBMS_SQL.FETCH_ROWS(c);
        DBMS_SQL.COLUMN_VALUE(c, 1, n_tab);
        DBMS_SQL.COLUMN_VALUE(c, 2, d_tab1);
        DBMS_SQL.COLUMN_VALUE(c, 3, v_tab);
        DBMS_SQL.COLUMN_VALUE(c, 4, d_tab2);
        EXIT WHEN d != 5;
    END LOOP;
    DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

The four tables can be used for anything. One usage might be to use `BIND_ARRAY` to move the rows to another table by using a statement such as `INSERT into SOME_T values (:a, :b, :c, :d);

*/

EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
        DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
154.9.8 DEFINE_COLUMN Procedures

This procedure defines a column to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the DEFINE_COLUMN_CHAR Procedure, DEFINE_COLUMN_LONG Procedure, DEFINE_COLUMN_RAW Procedure and DEFINE_COLUMN_ROWID Procedure.

Syntax

DBMS_SQL.DEFINE_COLUMN (  
c              IN INTEGER,  
position       IN INTEGER,  
column         IN <datatype>);  

Where <datatype> can be any one of the following types:

BINARY_DOUBLE  
BINARY_FLOAT  
BFILE  
BLOB  
CLOB CHARACTER SET ANY_CS  
DATE  
DSINTERVAL_UNCONSTRAINED  
NUMBER  
TIME_UNCONSTRAINED  
TIME_TZ_UNCONSTRAINED  
TIMESTAMP_LTZ_UNCONSTRAINED  
TIMESTAMP_TZ_UNCONSTRAINED  
TIMESTAMP_UNCONSTRAINED  
UROWID  
YMINTERVAL_UNCONSTRAINED  
user-defined object types  
collections (VARRAYs and nested tables)  
REFs  
Opaque types

Note that DEFINE_COLUMN is overloaded to accept different datatypes.

The following syntax is also supported for the DEFINE_COLUMN procedure:

DBMS_SQL.DEFINE_COLUMN (  
c              IN INTEGER,  
position       IN INTEGER,  
column         IN VARCHAR2 CHARACTER SET ANY_CS,  
column_size    IN INTEGER);  

Pragmas

pragma restrict_references(define_column,RNDS,WNDS);
Parameters

Table 154-15 DEFINE_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the row being defined to be selected</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the row being defined. The first column in a statement has position 1.</td>
</tr>
<tr>
<td>column</td>
<td>Value of the column being defined. The type of this value determines the type for the column being defined.</td>
</tr>
<tr>
<td>column_size</td>
<td>Maximum expected size of the column value in bytes for columns of type VARCHAR2.</td>
</tr>
</tbody>
</table>

Usage Notes

When using character length semantics the maximum number of bytes that can be returned for a column value of type VARCHAR2 is calculated as: column_size * maximum character byte size for the current character set. For example, specifying the column_size as 10 means that a maximum of 30 (10*3) bytes can be returned when using character length semantics with a UTF8 character set regardless of the number of characters this represents.

154.9.9 DEFINE_COLUMN_CHAR Procedure

This procedure defines a column with CHAR data to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the DEFINE_COLUMN Procedures, DEFINE_COLUMN_LONG Procedure, DEFINE_COLUMN_RAW Procedure and DEFINE_COLUMN_ROWID Procedure.

Syntax

```sql
DBMS_SQL.DEFINE_COLUMN_CHAR (  
  c              IN INTEGER,  
  position       IN INTEGER,  
  column         IN CHAR CHARACTER SET ANY_CS,  
  column_size    IN INTEGER);
```

Pragmas

```sql
pragma restrict_references(define_column,RNDS,WNDS);
```

Parameters

Table 154-16 DEFINE_COLUMN_CHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the row being defined to be selected</td>
</tr>
</tbody>
</table>
Table 154-16  (Cont.) DEFINE_COLUMN_CHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Relative position of the column in the row being defined. The first column</td>
</tr>
<tr>
<td></td>
<td>in a statement has position 1.</td>
</tr>
<tr>
<td>column</td>
<td>Value of the column being defined. The type of this value determines the</td>
</tr>
<tr>
<td></td>
<td>type for the column being defined.</td>
</tr>
<tr>
<td>column_size</td>
<td>Maximum expected size of the column value in characters for columns of</td>
</tr>
<tr>
<td></td>
<td>type CHAR.</td>
</tr>
</tbody>
</table>

154.9.10 DEFINE_COLUMN_LONG Procedure

This procedure defines a LONG column for a SELECT cursor. The column being defined is identified by its relative position in the SELECT list of the statement for the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the DEFINE_COLUMN Procedures, DEFINE_COLUMN_CHAR Procedure, DEFINE_COLUMN_RAW Procedure and DEFINE_COLUMN_ROWID Procedure.

Syntax

```
DBMS_SQL.DEFINE_COLUMN_LONG (  
c              IN INTEGER,  
position       IN INTEGER);
```

Parameters

Table 154-17  DEFINE_COLUMN_LONG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the row being defined to be selected.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the row being defined. The first column</td>
</tr>
<tr>
<td></td>
<td>in a statement has position 1.</td>
</tr>
</tbody>
</table>

154.9.11 DEFINE_COLUMN_RAW Procedure

This procedure defines a column of type RAW to be selected from the given cursor.

This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the DEFINE_COLUMN Procedures, DEFINE_COLUMN_CHAR Procedure, DEFINE_COLUMN_LONG Procedure and DEFINE_COLUMN_ROWID Procedure.

Syntax

```
DBMS_SQL.DEFINE_COLUMN_RAW (  
c              IN INTEGER,  
position       IN INTEGER,
```

...
column IN RAW,
column_size IN INTEGER);

Pragmas
pragma restrict_references(define_column,RNDS,WNDS);

Parameters

Table 154-18 DEFINE_COLUMN_RAW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the row being defined to be selected.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the row being defined. The first column in a statement has position 1.</td>
</tr>
<tr>
<td>column</td>
<td>Value of the column being defined. The type of this value determines the type for the column being defined.</td>
</tr>
<tr>
<td>column_size</td>
<td>Maximum expected size of the column value in bytes for columns of RAW type.</td>
</tr>
</tbody>
</table>

154.9.12 DEFINE_COLUMN_ROWID Procedure

This procedure defines a column of type ROWID to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the DEFINE_COLUMN Procedures, DEFINE_COLUMN_CHAR Procedure, DEFINE_COLUMN_LONG Procedure and DEFINE_COLUMN_RAW Procedure.

Syntax

DBMS_SQL.DEFINE_COLUMN_ROWID (  
c IN INTEGER,  
position IN INTEGER,  
column IN ROWID);

Pragmas
pragma restrict_references(define_column,RNDS,WNDS);

Parameters

Table 154-19 DEFINE_COLUMN_ROWID Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the row being defined to be selected.</td>
</tr>
<tr>
<td>position</td>
<td>Relative position of the column in the row being defined. The first column in a statement has position 1.</td>
</tr>
<tr>
<td>column</td>
<td>Value of the column being defined. The type of this value determines the type for the column being defined.</td>
</tr>
</tbody>
</table>
154.9.13 DESCRIBE_COLUMNS Procedure

This procedure describes the columns for a cursor opened and parsed through DBMS_SQL.

Syntax

```sql
DBMS_SQL.DESCRIBE_COLUMNS (
   c              IN  INTEGER,
   col_cnt        OUT INTEGER,
   desc_t         OUT DESC_TAB);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the columns being described</td>
</tr>
<tr>
<td>col_cnt</td>
<td>Number of columns in the select list of the query</td>
</tr>
<tr>
<td>desc_t</td>
<td>Describe table to fill in with the description of each of the columns of the query</td>
</tr>
</tbody>
</table>

Example 154-4 Describe Columns

This code can be used as a substitute to the SQL*Plus DESCRIBE call by using a SELECT * query on the table that you want to describe.

```sql
DECLARE
   c           NUMBER;
   d           NUMBER;
   col_cnt     INTEGER;
   f           BOOLEAN;
   rec_tab     DBMS_SQL.DESC_TAB;
   col_num    NUMBER;
PROCEDURE print_rec(rec in DBMS_SQL.DESC_REC) IS
BEGIN
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.PUT_LINE('col_type            =    ' || rec.col_type);
   DBMS_OUTPUT.PUT_LINE('col_maxlen          =    ' || rec.col_max_len);
   DBMS_OUTPUT.PUT_LINE('col_name            =    ' || rec.col_name);
   DBMS_OUTPUT.PUT_LINE('col_name_len        =    ' || rec.col_name_len);
   DBMS_OUTPUT.PUT_LINE('col_schema_name     =    ' || rec.col_schema_name);
   DBMS_OUTPUT.PUT_LINE('col_schema_name_len =    ' || rec.col_schema_name_len);
   DBMS_OUTPUT.PUT_LINE('col_precision       =    ' || rec.col_precision);
   DBMS_OUTPUT.PUT_LINE('col_scale           =    ' || rec.col_scale);
   DBMS_OUTPUT.PUT('col_null_ok         =    ');
   IF (rec.col_null_ok) THEN
      DBMS_OUTPUT.PUT_LINE('true');
   ELSE
      DBMS_OUTPUT.PUT_LINE('false');
   END IF;
END;
BEGIN
   DBMS_OUTPUT.PUT_LINE('col_null_ok         =    ');
   IF (rec.col_null_ok) THEN
      DBMS_OUTPUT.PUT_LINE('true');
   ELSE
      DBMS_OUTPUT.PUT_LINE('false');
   END IF;
END;
```

c := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(c, 'SELECT * FROM scott.bonus', DBMS_SQL.NATIVE);
d := DBMS_SQL.EXECUTE(c);
DBMS_SQL.DESCRIBE_COLUMNS(c, col_cnt, rec_tab);

/*
* Following loop could simply be for j in 1..col_cnt loop.
* Here we are simply illustrating some of the PL/SQL table
* features.
*/
col_num := rec_tab.first;
IF (col_num IS NOT NULL) THEN
    LOOP
        print_rec(rec_tab(col_num));
        col_num := rec_tab.next(col_num);
        EXIT WHEN (col_num IS NULL);
    END LOOP;
END IF;

DBMS_SQL.CLOSE_CURSOR(c);
END;
/

154.9.14 DESCRIBE_COLUMNS2 Procedure

This procedure describes the specified column. This is an alternative to DESCRIBE_COLUMNS Procedure.

Syntax

DBMS_SQL.DESCRIBE_COLUMNS2 ( 
    c IN INTEGER, 
    col_cnt OUT INTEGER, 
    desc_t OUT DESC_TAB2);

Pragmas

PRAGMA RESTRICT_REFERENCES(describe_columns2,WNDS);

Parameters

| Table 154-21 DESCRIBE_COLUMNS2 Procedure Parameters |
|----------------------------------------|------------------------------------------------|
| Parameter | Description |
| c          | ID number of the cursor for the columns being described. |
| col_cnt    | Number of columns in the select list of the query. |
| desc_t     | Describe table to fill in with the description of each of the columns of the query. This table is indexed from one to the number of elements in the select list of the query. |
Related Topics

- **DESCRIBE_COLUMNS Procedure**
  This procedure describes the columns for a cursor opened and parsed through DBMS_SQL.

### 154.9.15 DESCRIBE_COLUMNS3 Procedure

This procedure describes the specified column. This is an alternative to DESCRIBE_COLUMNS Procedure.

#### Syntax

```sql
DBMS_SQL.DESCRIBE_COLUMNS3 (  
c              IN  INTEGER,  
col_cnt        OUT INTEGER,  
desc_t         OUT DESC_TAB3);

BMS_SQL.DESCRIBE_COLUMNS3 (  
c              IN  INTEGER,  
col_cnt        OUT INTEGER,  
desc_t         OUT DESC_TAB4);
```

#### Pragmas

PRAGMA RESTRICT_REFERENCES(describe_columns3,WNDS);

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor for the columns being described.</td>
</tr>
<tr>
<td>col_cnt</td>
<td>Number of columns in the select list of the query.</td>
</tr>
<tr>
<td>desc_t</td>
<td>Describe table to fill in with the description of each of the columns of the query. This table is indexed from one to the number of elements in the select list of the query.</td>
</tr>
</tbody>
</table>

#### Usage Notes

The cursor passed in by the cursor ID has to be OPENed and PARSeD, otherwise an "invalid cursor id" error is raised.

#### Examples

```sql
CREATE TYPE PROJECT_T AS OBJECT  
    ( projname          VARCHAR2(20),  
      mgr               VARCHAR2(20))  
/

CREATE TABLE projecttab(deptno NUMBER, project HR.PROJECT_T)  
/

DECLARE  
curid      NUMBER;  
desctab    DBMS_SQL.DESC_TAB3;  
colcnt     NUMBER;  
sql_stmt   VARCHAR2(200) := 'select * from projecttab';```
BEGIN

curid := DBMS_SQL.OPEN_CURSOR;

DBMS_SQL.PARSE(curid, sql_stmt, DBMS_SQL.NATIVE);

DBMS_SQL.DESCRIBE_COLUMNS3(curid, colcnt, desctab);

FOR i IN 1 .. colcnt LOOP
    IF desctab(i).col_type = 109 THEN
        DBMS_OUTPUT.PUT(desctab(i).col_name || ' is user-defined type: ');
        DBMS_OUTPUT.PUT_LINE(desctab(i).col_schema_name || '.' ||
                              desctab(i).col_type_name);
    END IF;
END LOOP;

DBMS_SQL.CLOSE_CURSOR(curid);
END;
/

Output:

PROJECT is user-defined type: HR.PROJECT_T

Related Topics

• DESCRIBE_COLUMNS Procedure
  This procedure describes the columns for a cursor opened and parsed through
  DBMS_SQL.

154.9.16 EXECUTE Function

This function executes a given cursor. This function accepts the ID number of the
cursor and returns the number of rows processed.

The return value is only valid for INSERT, UPDATE, and DELETE statements; for other
types of statements, including DDL, the return value is undefined and must be ignored.

Syntax

DBMS_SQL.EXECUTE (  
    c   IN INTEGER)  
RETURN INTEGER;

Parameters

Table 154-23  EXECUTE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Cursor ID number of the cursor to execute.</td>
</tr>
</tbody>
</table>

Return Values

Returns number of rows processed
Usage Notes

The DBMS_SQL cursor that is returned by the TO_CURSOR_NUMBER Function performs in the same way as a DBMS_SQL cursor that has already been executed. Consequently, calling EXECUTE for this cursor will cause an error.

154.9.17 EXECUTE_AND_FETCH Function

This function executes the given cursor and fetches rows.

This function provides the same functionality as calling EXECUTE and then calling FETCH_ROWS. Calling EXECUTE_AND_FETCH instead, however, may reduce the number of network round-trips when used against a remote database.

The EXECUTE_AND_FETCH function returns the number of rows actually fetched.

Syntax

```sql
DBMS_SQL.EXECUTE_AND_FETCH (  
c              IN INTEGER,  
exact          IN BOOLEAN DEFAULT FALSE)  
RETURN INTEGER;
```

Pragmas

```sql
pragma restrict_references(execute_and_fetch,WNDS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor to execute and fetch.</td>
</tr>
<tr>
<td>exact</td>
<td>Set to TRUE to raise an exception if the number of rows actually matching the query differs from one. Note: Oracle does not support the exact fetch TRUE option with LONG columns. Even if an exception is raised, the rows are still fetched and available.</td>
</tr>
</tbody>
</table>

Return Values

Returns designated rows

154.9.18 FETCH_ROWS Function

This function fetches a row from a given cursor.

You can call FETCH_ROWS repeatedly as long as there are rows remaining to be fetched. These rows are retrieved into a buffer, and must be read by calling COLUMN_VALUE, for each column, after each call to FETCH_ROWS.

The FETCH_ROWS function accepts the ID number of the cursor to fetch, and returns the number of rows actually fetched.
Syntax

DBMS_SQL.FETCH_ROWS (c IN INTEGER)
RETURN INTEGER;

Pragmas

pragma restrict_references(fetch_rows,WNDS);

Parameters

Table 154-25 FETCH_ROWS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number.</td>
</tr>
</tbody>
</table>

Return Values

Returns a row from a given cursor

154.9.19 GET_NEXT_RESULT Procedures

This procedure gets the statement of the next result returned to the caller of the recursive statement or, if this caller sets itself as the client for the recursive statement, the next result returned to this caller as client.

The statements are returned in same order as they are returned by the RETURN_RESULT Procedures.

Syntax

DBMS_SQL.GET_NEXT_RESULT (c IN INTEGER,
rc OUT SYS_REFCURSOR);

DBMS_SQL.GET_NEXT_RESULT (c IN INTEGER,
rc OUT integer);

Parameters

Table 154-26 GET_NEXT_RESULT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Recursive statement cursor</td>
</tr>
<tr>
<td>rc</td>
<td>Cursor or ref cursor of the statement of the next returned result</td>
</tr>
</tbody>
</table>

Exceptions

ORA-01403 no_data_found: This is raised when there is no further returned statement result.
Usage Notes

- After the cursor of a statement result is retrieved, the caller must close the cursor properly when it is no longer needed.
- The cursors for all unretrieved returned statements will be closed after the cursor of the recursive statement is closed.

Examples

```sql
DECLARE
    c  INTEGER;
    rc SYS_REFCURSOR;
BEGIN
    c := DBMS_SQL.OPEN_CURSOR(treat_as_client_for_results => TRUE);
    DBMS_SQL.PARSE(c => c,
        statement => 'begin proc; end;');
    DBMS_SQL.EXECUTE(c);
    LOOP
        DBMS_SQL.GET_NEXT_RESULT(c, rc);
        EXCEPTIONS
            WHEN no_data_found THEN
                EXIT;
        END;
    LOOP
    END LOOP;
END;
```

154.9.20 IS_OPEN Function

This function checks to see if the given cursor is currently open.

Syntax

```sql
DBMS_SQL.IS_OPEN (c IN INTEGER) RETURN BOOLEAN;
```

Pragmas

```sql
pragma restrict_references(is_open,RNDS,WNDS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c</code></td>
<td>Cursor ID number of the cursor to check.</td>
</tr>
</tbody>
</table>
Return Values

Returns TRUE for any cursor number that has been opened but not closed, and FALSE for a NULL cursor number. Note that the CLOSE_CURSOR Procedure nulls out the cursor variable passed to it.

Exceptions

ORA-29471 DBMS_SQL access denied: This is raised if an invalid cursor ID number is detected. Once a session has encountered and reported this error, every subsequent DBMS_SQL call in the same session will raise this error, meaning that DBMS_SQL is non-operational for this session.

154.9.21 LAST_ERROR_POSITION Function

This function returns the byte offset in the SQL statement text where the error occurred. The first character in the SQL statement is at position 0.

Syntax

DBMS_SQL.LAST_ERROR_POSITION
    RETURN INTEGER;

Pragmas

pragma restrict_references(last_error_position,RNDS,WNDS);

Return Values

Returns the byte offset in the SQL statement text where the error occurred

Usage Notes

Call this function after a PARSE call, before any other DBMS_SQL procedures or functions are called.

154.9.22 LAST_ROW_COUNT Function

This function returns the cumulative count of the number of rows fetched.

Syntax

DBMS_SQL.LAST_ROW_COUNT
    RETURN INTEGER;

Pragmas

pragma restrict_references(last_row_count,RNDS,WNDS);

Return Values

Returns the cumulative count of the number of rows fetched

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call. If called after an EXECUTE call, then the value returned is zero.
154.9.23 LAST_ROW_ID Function

This function returns the ROWID of the last row processed.

Syntax

```
DBMS_SQL.LAST_ROW_ID
   RETURN ROWID;
```

Pragmas

```
pragma restrict_references(last_row_id,RNDS,WNDS);
```

Return Values

Returns the ROWID of the last row processed

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call.

154.9.24 LAST_SQL_FUNCTION_CODE Function

This function returns the SQL function code for the statement.

These codes are listed in the Oracle Call Interface Programmer's Guide.

Syntax

```
DBMS_SQL.LAST_SQL_FUNCTION_CODE
   RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(last_sql_function_code,RNDS,WNDS);
```

Return Values

Returns the SQL function code for the statement

Usage Notes

You must call this function immediately after the SQL statement is run; otherwise, the return value is undefined.

154.9.25 OPEN_CURSOR Functions

This function opens a new cursor.

The security_level parameter allows for application of fine-grained control to the security of the opened cursor.

Syntax

```
DBMS_SQL.OPEN_CURSOR (   treat_as_client_for_results   IN   BOOLEAN   DEFAULT FALSE)   RETURN INTEGER;
```
DBMS_SQL.OPEN_CURSOR (  
  security_level                 IN     INTEGER,  
  treat_as_client_for_results    IN     BOOLEAN    DEFAULT FALSE)  
RETURN INTEGER;

Parameters

Table 154-28  OPEN_CURSOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>security_level</td>
<td>Specifies the level of security protection to enforce on the opened cursor.</td>
</tr>
<tr>
<td></td>
<td>Valid security level values are 0, 1, and 2. When a NULL argument value is</td>
</tr>
<tr>
<td></td>
<td>provided to this overload, as well as for cursors opened using the</td>
</tr>
<tr>
<td></td>
<td>overload of open_cursor without the</td>
</tr>
<tr>
<td></td>
<td>security_level parameter, the default security level value 1</td>
</tr>
<tr>
<td></td>
<td>will be enforced on the opened cursor.</td>
</tr>
<tr>
<td></td>
<td>• Level 0 - allows all DBMS_SQL operations on the cursor without any</td>
</tr>
<tr>
<td></td>
<td>security checks. The cursor may be fetched from, and even re-bound and</td>
</tr>
<tr>
<td></td>
<td>re-executed, by code running with a different effective userid or roles</td>
</tr>
<tr>
<td></td>
<td>than those in effect at the time the cursor was parsed. This level of</td>
</tr>
<tr>
<td></td>
<td>security is off by default.</td>
</tr>
<tr>
<td></td>
<td>• Level 1 - requires that the referenced container, effective userid, and</td>
</tr>
<tr>
<td></td>
<td>roles of the caller to DBMS_SQL for bind and execute operations on this</td>
</tr>
<tr>
<td></td>
<td>cursor must be the same as those of the caller of the most recent parse</td>
</tr>
<tr>
<td></td>
<td>operation on this cursor.</td>
</tr>
<tr>
<td></td>
<td>• Level 2 - requires that the referenced container, effective userid, and</td>
</tr>
<tr>
<td></td>
<td>roles of the caller to DBMS_SQL for all bind, execute, define, describe,</td>
</tr>
<tr>
<td></td>
<td>and fetch operations on this cursor must be the same as those of the</td>
</tr>
<tr>
<td></td>
<td>caller of the most recent parse operation on this cursor.</td>
</tr>
<tr>
<td>treat_as_client_for</td>
<td>Allows the caller of the recursive statement to set itself as the client</td>
</tr>
<tr>
<td>results</td>
<td>to receive the statement results returned from the recursive statement to</td>
</tr>
<tr>
<td></td>
<td>client. The statement results returned may be retrieved</td>
</tr>
<tr>
<td></td>
<td>by the GET_NEXT_RESULT Procedures.</td>
</tr>
</tbody>
</table>

Pragmas

pragma restrict_references(open_cursor,RNDS,WNDS);

Return Values

Returns the cursor ID number of the new cursor

Usage Notes

• When you no longer need this cursor, you must close it explicitly by calling the
  CLOSE_CURSOR Procedure.

• You can use cursors to run the same SQL statement repeatedly or to run a new
  SQL statement. When a cursor is reused, the contents of the corresponding cursor
  data area are reset when the new SQL statement is parsed. It is never necessary
  to close and reopen a cursor before reusing it.
154.9.26 PARSE Procedures

This procedure parses the given statement in the given cursor. All statements are parsed immediately. In addition, DDL statements are run immediately when parsed.

There are multiple versions of the PARSE procedure:

- Taking a VARCHAR2 statement as an argument
- Taking a segmented string, one taking VARCHAR2A, a TABLE OF VARCHAR2(32767), and another, taking VARCHAR2S, a TABLE OF VARCHAR2(256), as argument. These overloads concatenate elements of a PL/SQL table statement and parse the resulting string. You can use these procedures to parse a statement that is longer than the limit for a single VARCHAR2 variable by splitting up the statement.
- Taking a CLOB statement as an argument. You can use the CLOB overload version of the parse procedure to parse a SQL statement larger than 32K bytes.

Syntax

Each version has multiple overloads.

```sql
DBMS_SQL.PARSE (c IN INTEGER, statement IN VARCHAR2, language_flag IN INTEGER[, edition IN VARCHAR2 DEFAULT NULL], apply_crossedition_trigger IN VARCHAR2 DEFAULT NULL, fire_apply_trigger IN BOOLEAN DEFAULT TRUE[, schema IN VARCHAR2 DEFAULT NULL[, container IN VARCHAR2]])

DBMS_SQL.PARSE (c IN INTEGER, statement IN CLOB, language_flag IN INTEGER[, edition IN VARCHAR2 DEFAULT NULL], apply_crossedition_trigger IN VARCHAR2 DEFAULT NULL, fire_apply_trigger IN BOOLEAN DEFAULT TRUE[, schema IN VARCHAR2 DEFAULT NULL[, container IN VARCHAR2]])

DBMS_SQL.PARSE (c IN INTEGER, statement IN VARCHAR2A, lb IN INTEGER, ub IN INTEGER, lfflg IN BOOLEAN, language_flag IN INTEGER[, edition IN VARCHAR2 DEFAULT NULL], apply_crossedition_trigger IN VARCHAR2 DEFAULT NULL, fire_apply_trigger IN BOOLEAN DEFAULT TRUE[, schema IN VARCHAR2 DEFAULT NULL[, container IN VARCHAR2]])

DBMS_SQL.PARSE (c IN INTEGER, statement IN VARCHAR2s, lb IN INTEGER,
```
Parameters

Table 154-29  PARSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor in which to parse the statement.</td>
</tr>
<tr>
<td>statement</td>
<td>SQL statement to be parsed. SQL statements larger than 32K that may be stored in CLOBs.</td>
</tr>
<tr>
<td></td>
<td>Unlike a PL/SQL statement, your SQL statement must not include a final semicolon. For example:</td>
</tr>
<tr>
<td></td>
<td>DBMS_SQL.PARSE(cursor1, 'BEGIN proc; END;', 2);</td>
</tr>
<tr>
<td></td>
<td>DBMS_SQL.PARSE(cursor1, 'INSERT INTO tab VALUES(1)', 2);</td>
</tr>
<tr>
<td>lb</td>
<td>Lower bound for elements in the statement</td>
</tr>
<tr>
<td>ub</td>
<td>Upper bound for elements in the statement</td>
</tr>
<tr>
<td>lfflg</td>
<td>If TRUE, then insert a linefeed after each element on concatenation.</td>
</tr>
<tr>
<td>language_flag</td>
<td>Specifies the behavior for the SQL statement. For more information about the possible values and its corresponding behaviors, see DBMS_SQL Constants</td>
</tr>
<tr>
<td>edition</td>
<td>Specifies the edition in which to run the statement under the following conditions:</td>
</tr>
<tr>
<td></td>
<td>• If NULL and container is NULL, the statement will be run in the current edition.</td>
</tr>
<tr>
<td></td>
<td>• If a valid container is specified, passing NULL indicates the statement is to run in the target container's default edition.</td>
</tr>
<tr>
<td></td>
<td>• Given the user and the edition with which the statement is to be executed, the user must have USE privilege on the edition.</td>
</tr>
<tr>
<td></td>
<td>The following general conditions apply. The contents of the string are processed as a SQL identifier; double quotation marks must surround the remainder of the string if special characters or lowercase characters are present in the edition's actual name, and if double quotation marks are not used the contents will be uppercased.</td>
</tr>
</tbody>
</table>
### Table 154-29 (Cont.) PARSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_crossedition_trigger</td>
<td>Specifies the unqualified name of a forward crossedition trigger that is to be applied to the specified SQL. The name is resolved using the edition and current_schema setting in which the statement is to be executed. The trigger must be owned by the user that will execute the statement. If a non-NULL value is specified, the specified crossedition trigger will be executed assuming fire_apply_trigger is TRUE, the trigger is enabled, the trigger is defined on the table which is the target of the statement, the type of the statement matches the trigger's dml_event_clause, any effective WHEN and UPDATE OF restrictions are satisfied, and so on. Other forward crossedition triggers may also be executed, selected using the &quot;crossedition trigger DML rules&quot; applied as if the specified trigger was doing a further DML to the table that is the target of the statement. Non-crossedition triggers and reverse crossedition triggers will not be executed. The contents of the string are processed as a SQL identifier; double quotation marks must surround the remainder of the string if special characters or lowercase characters are present in the trigger's actual name, and if double quotation marks are not used, the contents will be uppercased.</td>
</tr>
<tr>
<td>fire_apply_trigger</td>
<td>Indicates whether the specified apply_crossedition_trigger is itself to be executed, or must only be a guide used in selecting other triggers. This is typically set FALSE when the statement is a replacement for the actions the apply_crossedition_trigger would itself perform. If FALSE, the specified trigger is not executed, but other triggers are still selected for firing as if the specified trigger was doing a DML to the table that is the target of the statement. The apply_crossedition_trigger and fire_apply_trigger parameters are ignored if the statement is not a DML.</td>
</tr>
<tr>
<td>schema</td>
<td>Specifies the schema in which to resolve unqualified object names. If NULL, the current schema is the effective user's schema.</td>
</tr>
<tr>
<td>container</td>
<td>Name of the target container in which the cursor is to run. If NULL or unspecified, the name of the target container is that of the calling container and no container switch is performed. If a valid container name is specified, the current user must be a common user with SET CONTAINER privilege to switch to the target container. If a container switch completes, the effective user will have its default roles.</td>
</tr>
</tbody>
</table>

### Usage Notes

- Using `DBMS_SQL` to dynamically run DDL statements can cause the program to stop responding. For example, a call to a procedure in a package results in the package being locked until the execution returns to the user side. Any operation that results in a conflicting lock, such as dynamically trying to drop the package before the first lock is released, stops the program from running.

- Because client-side code cannot reference remote package variables or constants, you must explicitly use the values of the constants.

For example, the following code does not compile on the client:

```sql
DBMS_SQL.PARSE(cur_hdl, stmt_str, DBMS_SQL.NATIVE); -- uses constant DBMS_SQL.NATIVE
```

The following code works on the client, because the argument is explicitly provided:
DBMS_SQLPARSE(cur_hdl, stmt_str, 1); -- compiles on the client

- The VARCHAR2S type is currently supported for backward compatibility of legacy code. However, you are advised to use VARCHAR2A both for its superior capability and because VARCHAR2S will be deprecated in a future release.

- To parse SQL statements larger than 32 KB, the new CLOB overload version of the PARSE procedure can be used instead of the VARCHAR2A overload.

- If the container parameter value is the same as the calling container, a container switch will not occur. However, the default roles of the current user will be in effect.

Exceptions

If you create a type, procedure, function, or package using DBMS_SQL that has compilation warnings, an ORA-24344 exception is raised, and the PL/SQL unit is still created.

154.9.27 RETURN_RESULT Procedures

This procedure returns the result of an executed statement to the client application.

The result can be retrieved later by the client. Alternatively, it can return the statement result to and be retrieved later by the immediate caller that executes a recursive statement in which this statement result will be returned.

The caller can be:

- A PL/SQL stored procedure executing the recursive statement using DBMS_SQL
- A Java stored procedure using JDBC
- A .NET stored procedure using ADO.NET
- An external procedure using the Oracle Call Interface (OCI)

Syntax

```sql
DBMS_SQL.RETURN_RESULT(
    rc           IN OUT      SYS_REFCURSOR,  
    to_client    IN          BOOLEAN           DEFAULT TRUE);

DBMS_SQL.RETURN_RESULT(
    rc           IN OUT      INTEGER,          
    to_client    IN          BOOLEAN           DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc</td>
<td>Statement cursor or ref cursor</td>
</tr>
<tr>
<td>to_client</td>
<td>Returns (or does not return) the statement result to the client. If not, it is returned to the immediate caller.</td>
</tr>
</tbody>
</table>
Usage Notes

- Currently only a SQL query can be returned, and the return of statement results over remote procedure calls is not supported.
- Once the statement is returned, it is no longer accessible except by the client or the immediate caller to which it is returned.
- Statement results cannot be returned when the statement being executed by the client or any intermediate recursive statement is a SQL query and an error is raised.
- A ref cursor being returned can be strongly or weakly-typed.
- A query being returned can be partially fetched.
- **Because** `EXECUTE IMMEDIATE` **statement provides no interface to retrieve the statement results returned from its recursive statement, the cursors of the statement results returned to the caller of the `EXECUTE IMMEDIATE` statement will be closed when the statement completes. To retrieve the returned statement results from a recursive statement in PL/SQL, use `DBMS_SQL` to execute the recursive statement.**

Examples

```sql
CREATE PROCEDURE proc AS
  rc1 sys_refcursor;
  rc2 sys_refcursor;
BEGIN
  OPEN rc1 FOR SELECT * FROM t1;
  DBMS_SQL.RETURN_RESULT(rc1);
  OPEN rc2 FOR SELECT * FROM t2;
  DBMS_SQL.RETURN_RESULT(rc2);
END;
/
```

154.9.28 TO_CURSOR_NUMBER Function

This function takes an **OPENed** strongly or weakly-typed ref cursor and transforms it into a `DBMS_SQL` cursor number.

Syntax

```sql
DBMS_SQL.TO_CURSOR_NUMBER(
  rc IN OUT SYS_REFCURSOR)
RETURN INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc</td>
<td>REF CURSOR to be transformed into a cursor number</td>
</tr>
</tbody>
</table>

Return Values

Returns a `DBMS_SQL` manageable cursor number transformed from a `REF CURSOR`
Usage Notes

- The REF CURSOR passed in has to be OPENed, otherwise an error is raised.
- Once the REF CURSOR is transformed into a DBMS_SQL cursor number, the REF CURSOR is no longer accessible by any native dynamic SQL operations.
- The DBMS_SQL cursor that is returned by this subprogram performs in the same way as a DBMS_SQL cursor that has already been executed.

Examples

```sql
CREATE OR REPLACE PROCEDURE DO_QUERY(sql_stmt VARCHAR2) IS
    TYPE CurType IS REF CURSOR;
    src_cur         CurType;
    curid           NUMBER;
    desctab         DBMS_SQL.DESC_TAB;
    colcnt          NUMBER;
    namevar         VARCHAR2(50);
    numvar          NUMBER;
    datevar         DATE;
    empno           NUMBER := 100;
BEGIN

    -- sql_stmt := 'select ...... from employees where employee_id = :b1';
    OPEN src_cur FOR sql_stmt USING empno;

    -- Switch from native dynamic SQL to DBMS_SQL
    curid := DBMS_SQL.TO_CURSOR_NUMBER (src_cur);
    DBMS_SQL.DESCRIBE_COLUMNS(curid, colcnt, desctab);

    -- Define columns
    FOR i IN 1 .. colcnt LOOP
        IF desctab(i).col_type = 2 THEN
            DBMS_SQL.DEFINE_COLUMN(curid, i, numvar);
        ELSIF desctab(i).col_type = 12 THEN
            DBMS_SQL.DEFINE_COLUMN(curid, i, datevar);
        ........
        ELSE
            DBMS_SQL.DEFINE_COLUMN(curid, i, namevar, 25);
        END IF;
    END LOOP;

    -- Fetch Rows
    WHILE DBMS_SQL.FETCH_ROWS(curid) > 0 LOOP
        FOR i IN 1 .. colcnt LOOP
            IF (desctab(i).col_type = 1) THEN
                DBMS_SQL.COLUMN_VALUE(curid, i, namevar);
            ELSIF (desctab(i).col_type = 2) THEN
                DBMS_SQL.COLUMN_VALUE(curid, i, numvar);
            ELSIF (desctab(i).col_type = 12) THEN
                DBMS_SQL.COLUMN_VALUE(curid, i, datevar);
            ........
            END IF;
        END LOOP;
    END LOOP;

    DBMS_SQL.CLOSE_CURSOR(curid);
END;
```
154.9.29 TO_REFCURSOR Function

This function takes an OPENed, PARSeD, and EXECUTEd cursor and transforms/migrates it into a PL/SQL manageable REF CURSOR (a weakly-typed cursor) that can be consumed by PL/SQL native dynamic SQL switched to use native dynamic SQL.

This subprogram is only used with SELECT cursors.

Syntax

DBMS_SQL.TO_REFCURSOR(
   cursor_number IN OUT INTEGER)
RETURN SYS_REFCURSOR;

Parameters

Table 154-32 TO_REFCURSOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor_number</td>
<td>Cursor number of the cursor to be transformed into REF CURSOR</td>
</tr>
</tbody>
</table>

Return Values

Returns a PL/SQL REF CURSOR transformed from a DBMS_SQL cursor number

Usage Notes

- The cursor passed in by the cursor_number has to be OPENed, PARSeD, and EXECUTEd; otherwise an error is raised.
- Once the cursor_number is transformed into a REF CURSOR, the cursor_number is no longer accessible by any DBMS_SQL operations.
- After a cursor_number is transformed into a REF CURSOR, using DBMS_SQL.IS_OPEN to check to see if the cursor_number is still open results in an error.
- If the cursor number was last parsed with a valid container parameter, it cannot be converted to a REF CURSOR.

Examples

CREATE OR REPLACE PROCEDURE DO_QUERY(mgr_id NUMBER) IS
   TYPE CurType IS REF CURSOR;
   src_cur CurType;
   curid NUMBER;
   sql_stmt VARCHAR2(200);
   ret INTEGER;
   empnos DBMS_SQL.Number_Table;
   depts DBMS_SQL.Number_Table;
BEGIN
   -- DBMS_SQL.OPEN_CURSOR
   curid := DBMS_SQL.OPEN_CURSOR;

   sql_stmt := 'SELECT EMPLOYEE_ID, DEPARTMENT_ID from employees where MANAGER_ID
= :b1';

DBMS_SQL.PARSE(curid, sql_stmt, DBMS_SQL.NATIVE);
DBMS_SQL.BIND_VARIABLE(curid, 'b1', mgr_id);
ret := DBMS_SQL.EXECUTE(curid);

-- Switch from DBMS_SQL to native dynamic SQL
src_cur := DBMS_SQL.TO_REFCURSOR(curid);

-- Fetch with native dynamic SQL
FETCH src_cur BULK COLLECT INTO empnos, depts;

IF empnos.COUNT > 0 THEN
  DBMS_OUTPUT.PUT_LINE('EMPNO DEPTNO');
  DBMS_OUTPUT.PUT_LINE('----- ------');
  -- Loop through the empnos and depts collections
  FOR i IN 1 .. empnos.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(empnos(i) || ' ' || depts(i));
  END LOOP;
  END LOOP;
END IF;
-- Close cursor
CLOSE src_cur;
END;
/

154.9.30 VARIABLE_VALUE Procedures

This procedure returns the value of the named variable for a given cursor. It is used to return the values of bind variables inside PL/SQL blocks or DML statements with returning clause.

Syntax

DBMS_SQL.VARIABLE_VALUE (c IN INTEGER,
                           name IN VARCHAR2,
                           value OUT NOCOPY <datatype>);

Where <datatype> can be any one of the following types:

- ADT (user-defined object types)
- BINARY_DOUBLE
- BINARY_FLOAT
- BFILE
- BLOB
- BOOLEAN
- CLOB CHARACTER SET ANY_CS
- DATE
- DSINTERVAL_UNCONSTRAINED
- NESTED table
- NUMBER
- OPAQUE types
- REF
- TIME_UNCONSTRAINED
- TIME_TZ_UNCONSTRAINED
- TIMESTAMP_LTZ_UNCONSTRAINED
- TIMESTAMP_TZ_UNCONSTRAINED
- TIMESTAMP_UNCONSTRAINED
- UROWID
For variables containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.VARIABLE_VALUE_CHAR (c IN INTEGER,
                          name IN VARCHAR2,
                          value OUT CHAR CHARACTER SET ANY_CS);

DBMS_SQL.VARIABLE_VALUE_RAW (c IN INTEGER,
                          name IN VARCHAR2,
                          value OUT RAW);

DBMS_SQL.VARIABLE_VALUE_ROWID (c IN INTEGER,
                         name IN VARCHAR2,
                         value OUT ROWID);
```

The following syntax enables the VARIABLE_VALUE procedure to accommodate bulk operations:

```
DBMS_SQL.VARIABLE_VALUE (c IN INTEGER,
                          name IN VARCHAR2,
                          value OUT NOCOPY <table_type>);
```

For bulk operations, `<table_type>` must be a supported DBMS_SQL predefined TABLE type.

See DBMS_SQL Data Structures

Pragmas

```
pragma restrict_references(variable_value,RNDS,WNDS);
```

Parameters

**Table 154-33  VARIABLE_VALUE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor from which to get the values.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the variable for which you are retrieving the value.</td>
</tr>
</tbody>
</table>
| value     | • Single row option: Returns the value of the variable for the specified position. Oracle raises the exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to BIND_VARIABLE.  
  • Array option: Local variable that has been declared `<table_type>`. For bulk operations, value is an OUT NOCOPY parameter. |
154.9.31 VARIABLE_VALUE_PKG Procedure

This procedure returns the value of the named variable for a given cursor.

It is used to return the values of bind variables of collection or record types inside PL/SQL blocks or DML statements with returning clause for a declared package. The type of the variable must be declared in the package specification. Bulk operations are not supported for these types.

Syntax

DBMS_SQL.VARIABLE_VALUE_PKG (  
c                 IN   INTEGER,  
name              IN   VARCHAR2,  
value             OUT NOCOPY <table_type>);

Where <datatype> can be any one of the following data types:

- RECORD
- VARRAY
- NESTED TABLE
- INDEX BY PLS_INTEGER TABLE
- INDEX BY BINARY_INTEGER TABLE

Parameters

Table 154-34  VARIABLE_VALUE_PKG Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>ID number of the cursor from which to get the values.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the variable for which you are retrieving the value.</td>
</tr>
<tr>
<td>value</td>
<td>Single row option: Returns the value of the variable for the specified position. Oracle raises the exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to BIND_VARIABLE_PKG. Array option: Local variable that has been declared &lt;table_type&gt;.</td>
</tr>
</tbody>
</table>

Example 154-5  Dynamic SQL using DBMS_SQL.VARIABLE_VALUE_PKG to Get the Value of a Bind Variable

The data types are declared in the package specification. The VARIABLE_VALUE_PKG is used to get the value of the bind variable v2 in the cursor SQL statement.

CREATE OR REPLACE PACKAGE ty_pkg AS  
TYPE rec IS RECORD  
   ( n1 NUMBER, 
...
n2 NUMBER);
TYPE trect IS TABLE OF NUMBER;
END ty_pkg;
/
CREATE OR REPLACE PROCEDURE dyn_sql_nt AS
  dummy NUMBER;
cur NUMBER;
v1 ty_pkg.trect;
v2 ty_pkg.trect;
str VARCHAR2(3000);
BEGIN
  v1 := ty_pkg.trect(1000);
  str := 'declare v1 ty_pkg.trect;  begin v1:=:v1; v1(1) := 2000; :v2 :=
v1; end;' ;
cur := DBMS_SQL.OPEN_CURSOR();
DBMS_SQL.PARSE(cur, str, DBMS_SQL.NATIVE);
DBMS_SQL.BIND_VARIABLE_PKG(cur, ':v1', v1);
DBMS_SQL.BIND_VARIABLE_PKG(cur, ':v2', v2);
dummy := DBMS_SQL.EXECUTE(cur);
DBMS_SQL.VARIABLE_VALUE_PKG(cur, ':v2', v2);
DBMS_OUTPUT.PUT_LINE('n =  ' || V2(1));
DBMS_SQL.CLOSE_CURSOR(cur);
END dyn_sql_nt;
/
EXEC dyn_sql_nt;

n = 2000
The `DBMS_SQL_MONITOR` package provides information about Real-Time SQL Monitoring and Real-Time Database Operation Monitoring.

This chapter contains the following topics:

- DBMS_SQL_MONITOR Overview
- DBMS_SQL_MONITOR Security Model
- DBMS_SQL_MONITOR Constants
- Summary of DBMS_SQL_MONITOR Subprograms

See Also:

DBMS_SQLTUNE

155.1 DBMS_SQL_MONITOR Overview

The `DBMS_SQL_MONITOR` package provides information about Real-Time SQL Monitoring and Real-Time Database Operation Monitoring.

These features provide automatic monitoring of SQL statements, PL/SQL blocks, or composite database operations that are considered high-cost. A simple database operation is a single SQL statement or PL/SQL procedure or function. A composite database operation is activity between two defined points in time in a database session. The monitored data is collected in the `V$SQL_MONITOR` and `V$SQL_PLAN_MONITOR` views.

The following subprograms begin and end monitoring of a composite database operation:

- BEGIN_OPERATION Function
- END_OPERATION Procedure

The following subprograms report on monitoring data collected in `V$SQL_MONITOR` and `V$SQL_PLAN_MONITOR`:

- REPORT_SQL_MONITOR Function
- REPORT_SQL_MONITOR_XML Function
- REPORT_SQL_MONITOR_LIST Function
- REPORT_SQL_MONITOR_LIST_XML Function
155.2 DBMS_SQL_MONITOR Security Model

This package is available to PUBLIC and executes with invoker's rights privileges. The reporting functions require privileges to select data from the catalog as provided by the role SELECT_CATALOG_ROLE.

155.3 DBMS_SQL_MONITOR Constants

The DBMS_SQL_MONITOR package uses the constants shown in the following table.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE_TRACKING</td>
<td>VARCHAR2 (30)</td>
<td>'Y'</td>
<td>Force track the composite database operation when the operation starts</td>
</tr>
<tr>
<td>NO_FORCE_TRACKING</td>
<td>VARCHAR2 (30)</td>
<td>'N'</td>
<td>Do not force track the composite database operation when the operation starts. It is only tracked when it has consumed 5 seconds of CPU or I/O time.</td>
</tr>
</tbody>
</table>

155.4 Summary of DBMS_SQL_MONITOR Subprograms

This table lists and describes the DBMS_SQL_MONITOR package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_OPERATION Function</td>
<td>This function starts a database operation in the current session.</td>
</tr>
<tr>
<td>END_OPERATION Procedure</td>
<td>This function ends a database operation in the current session. If the specified database operation does not exist, then this function has no effect.</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR Function</td>
<td>This function builds a detailed report with monitoring information for a SQL statement, PL/SQL block, or database operation.</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_XML Function</td>
<td>This function is identical to the REPORT_SQL_MONITOR function, except that the return type is XMLType.</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_LIST Function</td>
<td>This function builds a report for all or a subset of database operations that have been monitored by Oracle Database.</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_LIST_XML Function</td>
<td>This function is identical to the REPORT_SQL_MONITOR_LIST function, except that it returns XMLType.</td>
</tr>
</tbody>
</table>
155.4.1 BEGIN OPERATION Function

This function starts a database operation in the current session.

Syntax

```
DBMS_SQL_MONITOR.BEGIN_OPERATION (  
  dbop_name       IN VARCHAR2,  
  dbop_eid        IN NUMBER   := NULL,  
  forced_tracking IN VARCHAR2 := NO_FORCE_TRACKING,  
  attribute_list  IN VARCHAR2 := NULL,  
  session_id      IN NUMBER   := NULL,  
  session_serial  IN NUMBER   := NULL)  
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbop_name</td>
<td>Name for the composite database operation.</td>
</tr>
<tr>
<td>dbop_eid</td>
<td>Unique identifier for the current execution of the composite database operation.</td>
</tr>
<tr>
<td>forced_tracking</td>
<td>Whether tracking is forced. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- FORCE_TRACKING - forces the composite database operation to be tracked when the operation starts. You can also use the string variable Y.</td>
</tr>
<tr>
<td></td>
<td>- NO_FORCE_TRACKING - tracks the operation only when it has consumed at least 5 seconds of CPU or I/O time. You can also use the string variable N.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>List of user-created attributes. It is a comma-separated list of name-value pairs (for example, 'table_name=emp, operation=load').</td>
</tr>
<tr>
<td>session_id</td>
<td>Session ID of the session to be monitored. If omitted (or null), then the database monitors the current session.</td>
</tr>
<tr>
<td>session_serial</td>
<td>Serial number of the session to be monitored. If omitted (or null), then the database uses only the session ID to determine the session.</td>
</tr>
</tbody>
</table>

Return Values

This function returns the database operation execution ID. If the value is null for dbop_eid, then the database generates a unique value.

155.4.2 END_OPERATION Procedure

This function ends a database operation in the current session. If the specified database operation does not exist, then this function has no effect.

Syntax

```
DBMS_SQL_MONITOR.END_OPERATION(  
  dbop_name       IN VARCHAR2, 
)
```


```
dbop_eid   IN NUMBER)
RETURN NUMBER;
```

### Parameters

**Table 155-4  END_OPERATION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbop_name</td>
<td>Name of a composite database operation</td>
</tr>
<tr>
<td>dbop_eid</td>
<td>Unique identifier for the current execution of the composite database operation</td>
</tr>
</tbody>
</table>

### 155.4.3 REPORT_SQL_MONITOR Function

This function builds a detailed report with monitoring information for a SQL statement, PL/SQL block, or database operation.

For each operation, it gives key information and associated global statistics. Use this function to get detailed monitoring information for a database operation.

The target database operation for this report can be:

- The last database operation monitored by Oracle Database (default, no parameter).
- The last database operation executed in the specified session and monitored by Oracle Database. The session is identified by its session ID and optionally its serial number (-1 is current session).
- The last execution of a specific database operation identified by its `sql_id`.
- A specific execution of a database operation identified by the combination `sql_id`, `sql_exec_start`, and `sql_exec_id`.
- The last execution of a specific database operation identified by `dbop_name`.
- The specific execution of a database operation identified by the combination `dbop_name`, `dbop_exec_id`.

### Syntax

```sql
DBMS_SQL_MONITOR.REPORT_SQL_MONITOR (
  sql_id                    IN VARCHAR2 DEFAULT  NULL,
  dbop_name                 IN VARCHAR2 DEFAULT  NULL,
  dbop_exec_id              IN NUMBER   DEFAULT  NULL,
  session_id                IN NUMBER   DEFAULT  NULL,
  session_serial            IN NUMBER   DEFAULT  NULL,
  sql_exec_start            IN DATE     DEFAULT  NULL,
  sql_exec_id               IN NUMBER   DEFAULT  NULL,
  inst_id                   IN NUMBER   DEFAULT  NULL,
  start_time_filter         IN DATE     DEFAULT  NULL,
  end_time_filter           IN DATE     DEFAULT  NULL,
  instance_id_filter        IN NUMBER   DEFAULT  NULL,
  parallel_filter           IN VARCHAR2 DEFAULT  'YES',
  plan_line_filter          IN NUMBER   DEFAULT  NULL,
  event_detail              IN VARCHAR2 DEFAULT  'YES',
  bucket_max_count          IN NUMBER   DEFAULT  128,
  bucket_interval           IN NUMBER   DEFAULT  NULL,
)
```
Parameters

Table 155-5  REPORT_SQL_MONITOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQL_ID of the simple database operation for which monitoring information should be displayed. Use NULL (default) to display monitoring information for the last simple database operation monitored by Oracle.</td>
</tr>
<tr>
<td>dbop_name</td>
<td>DBOP_NAME for which monitoring information of the composite database operation is displayed</td>
</tr>
<tr>
<td>dbop_exec_id</td>
<td>Execution ID for the composite database operation for which monitoring information is displayed</td>
</tr>
<tr>
<td>session_id</td>
<td>Targets only the subset of statements executed and monitored on behalf of the specified session. Default is NULL. Use -1 or USERENV('SID') for the current session.</td>
</tr>
<tr>
<td>session_serial</td>
<td>In addition to session_id, you can specify the session serial number to ensure the desired session incarnation is targeted. This is ignored when session_id is NULL.</td>
</tr>
<tr>
<td>sql_exec_start</td>
<td>Time at which execution of the monitored SQL was started. Only applicable when sql_id is specified. Used to display monitoring information for a particular execution of sql_id. When NULL (default), the last execution of sql_id is shown.</td>
</tr>
<tr>
<td>sql_exec_id</td>
<td>A numeric ID generated internally by SQL monitor to identify different executions of the same SQL statement. Thus each execution will have the same sql_id but a different sql_exec_id. Only applicable when sql_id is specified and is used to display monitoring information for a particular execution of sql_id. When NULL (default), the last execution of sql_id is shown.</td>
</tr>
<tr>
<td>inst_id</td>
<td>Looks only at queries started on the specified instance. Use -1 to target the current instance. The default, NULL will target all instances.</td>
</tr>
<tr>
<td>start_time_filter</td>
<td>If not NULL, the report shows activity from V$ACTIVE_SESSION_HISTORY started after this date. If NULL, the reported activity starts once the targeted database operation has started.</td>
</tr>
<tr>
<td>end_time_filter</td>
<td>If not NULL, the report shows activity from V$ACTIVE_SESSION_HISTORY started before this date. If NULL, the reported activity ends when the targeted database operation has ended or SYSDATE if the operation is still executing.</td>
</tr>
<tr>
<td>instance_id_filter</td>
<td>Only looks at activity for the specified instance. Use NULL (the default) to target all instances. Only relevant if the query runs in parallel.</td>
</tr>
</tbody>
</table>
Table 155-5 (Cont.) REPORT_SQL_MONITOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel_filter</td>
<td>Parallel filter applies only to parallel execution and allows you to select only a subset of the processes involved in the parallel execution. The string <code>parallel_filter</code> can be:</td>
</tr>
<tr>
<td></td>
<td>- NULL - target all parallel execution servers as well as the query coordinator</td>
</tr>
<tr>
<td></td>
<td>- ['qc'][servers(&lt;svr_grp&gt;, &lt;svr_set&gt;, &lt;srv_num&gt;)] where any NULL value is interpreted as ALL</td>
</tr>
<tr>
<td>plan_line_filter</td>
<td>Selects activity and execution statistics for the specified line number in the plan of a SQL.</td>
</tr>
<tr>
<td>event_detail</td>
<td>When set to NO, the activity is aggregated by <code>wait_class</code> only. Use YES (default) to aggregate by <code>wait_class</code>, <code>event_name</code>.</td>
</tr>
<tr>
<td>bucket_max_count</td>
<td>Specifies the maximum number of buckets to create in the report</td>
</tr>
<tr>
<td>bucket_interval</td>
<td>Represents the exact time interval, in seconds, of all histogram buckets. If specified, <code>bucket_max_count</code> is ignored.</td>
</tr>
<tr>
<td>base_path</td>
<td>URL path for flex HTML resources since flex HTML format requires access to external files (Java scripts and the flash swf file).</td>
</tr>
<tr>
<td>last_refresh_time</td>
<td>If not NULL (default), the time when the report was last retrieved (SYSDATE attribute of the report tag). Use this option when you want to display the report of an running query and when that report is refreshed on a regular basis. This optimizes the size of the report since only the new changed information will be returned. In particular, the following will be optimized:</td>
</tr>
<tr>
<td></td>
<td>- SQL text will not be returned when this option is specified</td>
</tr>
<tr>
<td></td>
<td>- Activity histogram will start at the bucket that intersects that time. The entire content of the bucket is returned, even if <code>last_refresh_time</code> is after the start of that bucket</td>
</tr>
</tbody>
</table>
Table 155-5  (Cont.) REPORT_SQL_MONITOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report_level</td>
<td>Level of detail for the report. Of the following, only one can be specified:</td>
</tr>
<tr>
<td></td>
<td>- NONE: Minimum possible</td>
</tr>
<tr>
<td></td>
<td>- BASIC: This is equivalent to <code>sql_text-plan-xplan-sessions-</code></td>
</tr>
<tr>
<td></td>
<td><code>instance-activity_histogram-plan_histogram-metrics</code> where the token</td>
</tr>
<tr>
<td></td>
<td>`&quot;-&quot; implies that report section will not be included in the report.</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL: Everything but <code>plan_histogram</code></td>
</tr>
<tr>
<td></td>
<td>- ALL: Everything</td>
</tr>
<tr>
<td>type</td>
<td>Report type:</td>
</tr>
<tr>
<td></td>
<td>- TEXT: text report (default)</td>
</tr>
<tr>
<td></td>
<td>- HTML: simple HTML report</td>
</tr>
<tr>
<td></td>
<td>- ACTIVE: database active report. Some information (explain plan,</td>
</tr>
<tr>
<td></td>
<td><code>activity_histogram, metrics and plan_histogram</code>) is only</td>
</tr>
<tr>
<td></td>
<td>shown when this type is selected</td>
</tr>
<tr>
<td>sql_plan_hash_value</td>
<td>Targets only those with the specified plan hash value. Default is NULL.</td>
</tr>
<tr>
<td>con_name</td>
<td>Container name in a multitenant database.</td>
</tr>
</tbody>
</table>

In addition, SQL text can be specified at different levels:
- `-SQL TEXT`: No SQL text in report
- `+SQL_TEXT`: Alright with partial SQL text, that is, up to the first 2000 chars as stored in `GV$SQL_MONITOR`
- `SQL_FULLTEXT`: No full SQL text, that is, `+sql_text`
- `+SQL_FULLTEXT`: Show full SQL text (default)
Return Values

SQL monitor report, an XML document.

Usage Notes

The user invoking this function must have privilege to access the following fixed views:

- GV$SQL_MONITOR
- GV$SQL_PLAN_MONITOR
- GV$ACTIVE_SESSION_HISTORY
- GV$SESSION_LONGOPS
- GV$SQL if SQL full text is requested and its length is greater than 2 KB

155.4.4 REPORT_SQL_MONITOR.XML Function

This function is identical to the REPORT_SQL_MONITOR function, except that the return type is XMLType.

Related Topics

- REPORT_SQL_MONITOR Function
  This function builds a detailed report with monitoring information for a SQL statement, PL/SQL block, or database operation.

155.4.5 REPORT_SQL_MONITOR_LIST Function

This function builds a report for all or a subset of database operations that have been monitored by Oracle Database.

For each database operation, it gives key information and associated global statistics.

Syntax

```
DBMS_SQL_MONITOR.REPORT_SQL_MONITOR_LIST (  
  sql_id IN VARCHAR2 DEFAULT NULL,  
  dbop_name IN VARCHAR2 DEFAULT NULL,  
  monitor_type IN NUMBER   DEFAULT  MONITOR_TYPE_ALL,  
  session_id IN NUMBER   DEFAULT  NULL,  
  session_serial IN NUMBER   DEFAULT  NULL,  
  inst_id IN NUMBER   DEFAULT  NULL,  
  active_since_date IN DATE     DEFAULT  NULL,  
  active_since_sec IN NUMBER   DEFAULT  NULL,  
  last_refresh_time IN DATE     DEFAULT  NULL,  
  report_level IN VARCHAR2 DEFAULT 'TYPICAL',  
  auto_refresh IN NUMBER   DEFAULT  NULL,  
  base_path IN VARCHAR2 DEFAULT NULL,  
  type IN VARCHAR2 DEFAULT 'TEXT',  
  con_name IN VARCHAR2 DEFAULT NULL)  
RETURN CLOB;
```
### Parameters

#### Table 155-6  REPORT_SQL_MONITOR_LIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQL ID of the simple database operation for which monitoring information should be displayed. Use NULL (default) to display monitoring information for the last operation monitored by Oracle Database.</td>
</tr>
<tr>
<td>dbop_name</td>
<td>DBOP_NAME for which monitoring information of the composite database operation is displayed.</td>
</tr>
<tr>
<td>monitor_type</td>
<td>Monitor type:</td>
</tr>
<tr>
<td></td>
<td>• MONITOR_TYPE_SQL returns only simple database operations</td>
</tr>
<tr>
<td></td>
<td>• MONITOR_TYPE_DBOP returns composite database operations</td>
</tr>
<tr>
<td></td>
<td>• MONITOR_TYPE_ALL returns all types</td>
</tr>
<tr>
<td>session_id</td>
<td>Targets only the subset of database operations executed and monitored on behalf of the specified session. Default is NULL. Use -1 or USERENV('SID') for the current session.</td>
</tr>
<tr>
<td>session_serial</td>
<td>In addition to session_id, you can specify the session serial number to ensure the desired session incarnation is targeted. This is ignored when session_id is NULL.</td>
</tr>
<tr>
<td>inst_id</td>
<td>Looks only at monitored database operations originating from the specified instance. Use -1 to target the instance where the report executed. To target all instances, use NULL (default).</td>
</tr>
<tr>
<td>active_since_date</td>
<td>If not NULL (default), returns monitored database operations that have been active since the specified time. This includes all operations that are executing, as well as all operations that have completed their execution after the specified start time.</td>
</tr>
<tr>
<td>active_since_sec</td>
<td>If not NULL (default), returns monitored database operations that have been active since the specified time. This includes all operations that are executing, as well as all operations that have completed their execution after the specified date and time. In this case, the start time is specified relative to the current SYSDATE minus a specified number of seconds. For example, use 3600 to limit the report to all operations that have been active in the past 1 hour.</td>
</tr>
<tr>
<td>last_refresh_time</td>
<td>If not NULL (default), the time when the list report was last retrieved. This optimizes the case where an application shows the list and refreshes the report on a regular basis (such as once every 5 seconds). In this case, the report will show details about the execution of monitored queries that have been active since the specified last_refresh_time. For other queries, the report returns the execution key (sql_id, sql_exec_start, and sql_exec_id). Also, for queries that have their first refresh time after the specified date, only the SQL execution key and statistics are returned.</td>
</tr>
<tr>
<td>report_level</td>
<td>Level of detail for the report. The level can be BASIC (SQL text up to 200 character), TYPICAL (which include full SQL text assuming that cursor has not aged out, in which case the SQL text is included up to 2000 characters), or ALL which is the same as TYPICAL.</td>
</tr>
<tr>
<td>auto_refresh</td>
<td>Specifies the duration in seconds after which report data will be automatically refreshed while the monitored SQL or database operation is still executing. This applies to active report types.</td>
</tr>
</tbody>
</table>
Table 155-6  (Cont.) REPORT_SQL_MONITOR_LIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_path</td>
<td>URL path for flex HTML resources since flex HTML format requires access to external files (java scripts and the flash swf file).</td>
</tr>
<tr>
<td>type</td>
<td>Report type:</td>
</tr>
<tr>
<td></td>
<td>• TEXT: text report (default)</td>
</tr>
<tr>
<td></td>
<td>• HTML: simple HTML report</td>
</tr>
<tr>
<td></td>
<td>• ACTIVE: database active report. Some information (explain plan, activity_histogram, metrics, and plan_histogram) is only shown when this type is selected.</td>
</tr>
<tr>
<td></td>
<td>• XML: raw data for the report</td>
</tr>
<tr>
<td>con_name</td>
<td>Container name in a multitenant database.</td>
</tr>
</tbody>
</table>

Return Values

A report in text, XML, or HTML format that contains the list of the database operations monitored.

Usage Notes

• Use the REPORT_SQL_MONITOR Function to get detailed monitoring information for a single database operation.
• The user invoking this function needs to have the privilege to access the fixed views GV$SQL_MONITOR and GV$SQL.

155.4.6 REPORT_SQL_MONITOR_LIST_XML Function

This function is identical to the REPORT_SQL_MONITOR_LIST function, except that it returns XMLType.

Related Topics

• REPORT_SQL_MONITOR_LIST Function
  This function builds a report for all or a subset of database operations that have been monitored by Oracle Database.
The **DBMS_SQL_TRANSLATOR** package provides an interface for creating, configuring, and using SQL translation profiles.

This chapter contains the following topics:

- Security Model
- Constants
- Operational Notes
- Exceptions
- Examples
- Summary of DBMS_SQL_TRANSLATOR Subprograms

See Also:

SQL Translation Framework Architecture and Overview in *Oracle Database Migration Guide*

### 156.1 DBMS_SQL_TRANSLATOR Security Model

**DBMS_SQL_TRANSLATOR** is a invoker's rights package.

When translating a SQL statement or error, the translator package procedure will be invoked with the same current user and current schema as those in which the SQL statement being parsed. The owner of the translator package must be granted the `TRANSLATE SQL` user privilege on the current user.

Additionally, the current user must be granted the `EXECUTE` privilege on the translator package.

### 156.2 DBMS_SQL_TRANSLATOR Constants

**DBMS_SQL_TRANSLATOR** defines several constants to use when specifying parameter values.

These are shown in the following table.
### Table 156-1  DBMS_SQL_TRANSLATOR Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_EDITIONABLE</td>
<td>'EDITIONABLE'</td>
<td>VARCHAR2(30)</td>
<td>Name of the SQL translation profile attribute that specifies whether the SQL translation profile becomes an editioned or noneditioned object if editioning is later enabled for the schema object type SQL translation profile in the owner's schema (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_FOREIGN_SQL_SYNTAX</td>
<td>'FOREIGN_SQL_SYNTAX'</td>
<td>VARCHAR2(30)</td>
<td>Name of the SQL translation profile attribute that indicates if the profile is for translation of foreign SQL syntax (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_LOG_TRANSLATION_ERROR</td>
<td>'TRANSLATION_ERROR'</td>
<td>VARCHAR2(30)</td>
<td>Name of the SQL translation profile attribute that controls if the profile should log translation error in the database alert log (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_RAISE_TRANSLATION_ERROR</td>
<td>'TRANSLATION_ERROR'</td>
<td>VARCHAR2(30)</td>
<td>Name of the SQL translation profile attribute that controls if the profile should raise translation error if a SQL statement or error fails to be translated (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_TRANSLATE_NEW_SQL</td>
<td>'TRANSLATE_NEW_SQL'</td>
<td>VARCHAR2(30)</td>
<td>Name of the SQL translation profile attribute that controls if the profile should translate new SQL statements and errors (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_TRACE_TRANSLATION</td>
<td>'TRACE_TRANSLATION'</td>
<td>VARCHAR2(30)</td>
<td>Name of the SQL translation profile attribute that controls tracing (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_TRANSLATOR</td>
<td>'TRANSLATOR'</td>
<td>VARCHAR2(30)</td>
<td>Name of the SQL translation profile attribute that specifies the translator package (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_VALUE_TRUE</td>
<td>'TRUE'</td>
<td>VARCHAR2(30)</td>
<td>Value to set a SQL translation profile attribute to true (see Operational Notes)</td>
</tr>
<tr>
<td>ATTR_VALUE_FALSE</td>
<td>'FALSE'</td>
<td>VARCHAR2(30)</td>
<td>Value to set a SQL translation profile attribute to false (see Operational Notes)</td>
</tr>
</tbody>
</table>

### 156.3 DBMS_SQL_TRANSLATOR Operational Notes

The subprograms that modify a profile have DDL transaction semantics and when invoked will commit any open transaction in the session.

**ATTR_EDITIONABLE Constant**

Editionable is true by default.

**ATTR_FOREIGN_SQL_SYNTAX Constant**

Foreign SQL syntax is true by default.
ATTR_LOG_TRANSLATION_ERROR Constant

- If log translation is enabled in a SQL translation profile, an alert log is written to the database alert log if no custom translation is found for a SQL statement or error. This allows the user to catch any error in the custom translation in a profile.

- Log translation error is false by default.

ATTR_RAISE_TRANSLATION_ERROR Constant

Raise translation error is false by default.

ATTR_TRANSLATE_NEW_SQL Constant

- The name of the SQL translation profile attribute that controls if the profile should translate new SQL statements and errors. If so, the translator package, if registered, will translate a new SQL statement or error not already translated in custom translations, and also register the new translation as custom translation. If not, any new SQL statement or error encountered will result in a translation error.

- Translate new SQL statements and errors is true by default.

ATTR_TRACE_TRANSLATION Constant

- If tracing is enabled in a SQL translation profile, any SQL statement or error translated by the profile in a database session and its translation is written to the database session’s trace file.

- Tracing is disabled by default.

ATTR_TRANSLATOR Constant

- The translator package must be a PL/SQL package with the following three procedures. The TRANSLATE_SQL Procedure and the TRANSLATE_ERROR Procedure are called to translate SQL statements and errors. The names of the parameters of the translate procedures must be followed.

  PROCEDURE TRANSLATE_SQL(
    sql_text        IN  CLOB,
    translated_text OUT CLOB);

  PROCEDURE TRANSLATE_ERROR(
    error_code          IN  BINARY_INTEGER,
    translated_code     OUT BINARY_INTEGER,
    translated_sqlstate OUT VARCHAR2);

Parameters:

  profile_name        - profile name
  sql_text            - SQL statement to be translated
  translated_text     - translated SQL statement
  error_code          - Oracle error code
  translated_code     - translated error code
  translated_sqlstate - translated SQLSTATE

- When NULL is returned in translated_text, translated_code, or translated_sqlstate, it means that no translation is required and the original SQL statement, error code, or SQLSTATE is used instead.

- The name of the translator package follows the naming rules for database packages of the form [schema.]package_name. When the schema and package names are used, they
are set to uppercase by default unless surrounded by double quotation marks. For example, setting a translator package, `translator => 'dbms_tsql_translator'` is the same as `translator => 'Dbms_Tsql_Translator'` and `translator => 'DBMS_TSQL_TRANSLATOR'`, but not the same as `translator => '"dbms_tsql_translator"'`. The default schema name is the profile owner.

- The translator attribute is not set by default.

### ATTR_VALUE_TRUE Constant

The value to set a SQL translation profile attribute to true.

### ATTR_VALUE_FALSE Constant

The value to set a SQL translation profile attribute to false.

## 156.4 DBMS_SQL_TRANSLATOR Exceptions

This table lists the exceptions raised by the `DBMS_SQL_TRANSLATOR` package.

### Table 156-2 Exceptions Raised by DBMS_SQL_TRANSLATOR

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>29261</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>1031</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>24252</td>
<td>Profile does not exist</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>1918</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_TRANSLATION_FOUND</td>
<td>24253</td>
<td>No translation of the SQL statement or error code found</td>
</tr>
<tr>
<td>PROFILE_EXISTS</td>
<td>955</td>
<td>Profile already exists</td>
</tr>
</tbody>
</table>

## 156.5 DBMS_SQL_TRANSLATOR Examples

This is an example of basic SQL translation using `DBMS_SQL_TRANSLATOR`.

### Basic SQL Translation

```sql
BEGIN
  DBMS_SQL_TRANSLATOR.CREATE_PROFILE(
    profile_name => 'tsql_application');
  DBMS_SQL_TRANSLATOR.SET_ATTRIBUTE(
    profile_name => 'tsql_application',
    attribute_name => DBMS_SQL_TRANSLATOR.ATTR_TRANSLATOR,
    attribute_value => 'migration_repo.sybase_tsql_translator');
END;
```
## 156.6 Summary of DBMS_SQL_TRANSLATOR Subprograms

This table lists the DBMS_SQL_TRANSLATOR subprograms and briefly describes them.

### Table 156-3  DBMS_SQL_TRANSLATOR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_PROFILE Procedure</td>
<td>Creates a SQL translation profile</td>
</tr>
<tr>
<td>DEREGISTER_SQL_TRANSLATION Procedure</td>
<td>Deregisters the custom translation of a SQL statement in a SQL translation profile</td>
</tr>
<tr>
<td>DEREGISTER_ERROR_TRANSLATION Procedure</td>
<td>Deregisters the translation of an Oracle error code and SQLSTATE in a SQL translation profile</td>
</tr>
<tr>
<td>DROP_PROFILE Procedure</td>
<td>Drops a SQL translation profile and its contents</td>
</tr>
<tr>
<td>ENABLE_ERROR_TRANSLATION Procedure</td>
<td>Enables or disables a custom translation of an Oracle error code in a SQL translation profile</td>
</tr>
<tr>
<td>ENABLE_SQL_TRANSLATION Procedure</td>
<td>Enables or disables a custom translation of a SQL statement in a SQL translation profile</td>
</tr>
<tr>
<td>EXPORT_PROFILE Procedure</td>
<td>Exports the content of a SQL translation profile</td>
</tr>
<tr>
<td>IMPORT_PROFILE Procedure</td>
<td>Imports the content of a SQL translation profile</td>
</tr>
<tr>
<td>REGISTER_ERROR_TRANSLATION Procedure</td>
<td>Registers a custom translation of an Oracle error code and SQLSTATE in a SQL translation profile</td>
</tr>
<tr>
<td>REGISTER_SQL_TRANSLATION Procedure</td>
<td>Registers a custom translation of a SQL statement in a SQL translation profile</td>
</tr>
<tr>
<td>SET_ATTRIBUTE Procedure</td>
<td>Sets an attribute of a SQL translation profile</td>
</tr>
<tr>
<td>SQL_HASH Function</td>
<td>Computes the hash value of a SQL statement in a SQL translation profile</td>
</tr>
<tr>
<td>SQL_ID Function</td>
<td>Computes the SQL identifier of a SQL statement in a SQL translation profile</td>
</tr>
<tr>
<td>TRANSLATE_ERROR Procedure</td>
<td>Translates an Oracle error code and an ANSI SQLSTATE using a SQL translation profile</td>
</tr>
<tr>
<td>TRANSLATE_SQL Procedure</td>
<td>Translates a SQL statement using a SQL translation profile</td>
</tr>
</tbody>
</table>

### 156.6.1 CREATE_PROFILE Procedure

This procedure creates a SQL translation profile.

**Syntax**

```sql
DBMS_SQL_TRANSLATOR.CREATE_PROFILE (profile_name IN VARCHAR2);
```
Parameters

Table 156-4  CREATE_PROFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-5  CREATE_PROFILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>PROFILE_EXISTS</td>
<td>Profile already exists</td>
</tr>
</tbody>
</table>

Usage Notes

- A SQL translation profile is a database schema object that resides in SQL translation profile namespace. Its name follows the naming rules for database objects of the form [schema.]name. When the schema and profile names are used in the DBMS_SQL_TRANSLATOR package, they are uppercased unless surrounded by double quotation marks. For example, the translation profile `profile_name => 'tsql_application'` is the same as `profile_name => 'Tsql_Application'` and `profile_name => 'TSQL_APPLICATION'`, but not the same as `profile_name => '"tsql_application"'`.

- A SQL translation profile is an editionable object type.

- A SQL translation profile cannot be created as a common object in a multitenant container database (CDB).

- To destroy a SQL translation profile, use the DROP_PROFILE Procedure.

Examples

```sql
BEGIN
  DBMS_SQL_TRANSLATOR.CREATE_PROFILE(profile_name => 'tsql_application');
END;
```

156.6.2 DEREGISTER_SQL_TRANSLATION Procedure

This procedure deregisters the custom translation of a SQL statement in a SQL translation profile.

Syntax

```sql
DBMS_SQL_TRANSLATOR.DEREGISTER_SQL_TRANSLATION (  
  profile_name  IN VARCHAR2,  
  sql_text      IN CLOB);  
```
Parameters

Table 156-6 DEREGISTER_SQL_TRANSLATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>sql_text</td>
<td>SQL statement</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-7 DEREGISTER_SQL_TRANSLATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>PROFILE_EXISTS</td>
<td>Profile already exists</td>
</tr>
</tbody>
</table>

Examples

BEGIN
    DBMS_SQL_TRANSLATOR.DEREGISTER_SQL_TRANSLATION(
        profile_name => 'tsql_application',
        sql_text     => 'select top 5 * from emp');
END;

156.6.3 DEREGISTER_ERROR_TRANSLATION Procedure

This procedure deregisters the translation of an Oracle error code and SQLSTATE in a SQL translation profile.

Syntax

DBMS_SQL_TRANSLATOR.DEREGISTER_ERROR_TRANSLATION (
    profile_name IN VARCHAR2,
    error_code  IN PLS_INTEGER);

Parameters

Table 156-8 DEREGISTER_ERROR_TRANSLATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>error_code</td>
<td>Oracle error code</td>
</tr>
</tbody>
</table>
Exceptions

### Table 156-9  DEREGISTER_ERROR_TRANSLATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Examples

```sql
BEGIN
    DBMS_SQL_TRANSLATOR.DEREGISTER_ERROR_TRANSLATION(
        profile_name    => 'tsql_application',
        error_code      => 1);
END;
```

### 156.6.4 DROP_PROFILE Procedure

This procedure drops a SQL translation profile and its contents.

#### Syntax

```sql
DBMS_SQL_TRANSLATOR.DROP_PROFILE (
    profile_name IN VARCHAR2);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
</tbody>
</table>

#### Exceptions

### Table 156-11  DROP_PROFILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Examples

```sql
BEGIN
    DBMS_SQL_TRANSLATOR.DROP_PROFILE(
```
156.6.5 ENABLE_ERROR_TRANSLATION Procedure

This procedure enables or disables a custom translation of an Oracle error code in a SQL translation profile.

Syntax

```
DBMS_SQL_TRANSLATOR.ENABLE_ERROR_TRANSLATION (  
    profile_name      IN  VARCHAR2,
    sql_text          IN  CLOB,
    enable            IN  BOOLEAN DEFAULT TRUE);
```

Parameters

Table 156-12  ENABLE_ERROR_TRANSLATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>sql_text</td>
<td>SQL statement</td>
</tr>
<tr>
<td>enable</td>
<td>Enable or disable the translation</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-13  ENABLE_ERROR_TRANSLATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Examples

```
BEGIN
    DBMS_SQL_TRANSLATOR.ENABLE_ERROR_TRANSLATION(  
        profile_name => 'tsql_application',
        sql_text     => 'SELECT TOP 5 * FROM emp',
        enable       => TRUE);
END;
```

156.6.6 ENABLE_SQL_TRANSLATION Procedure

This procedure enables or disables a custom translation of a SQL statement in a SQL translation profile.

Syntax

```
DBMS_SQL_TRANSLATOR.ENABLE_SQL_TRANSLATION (  
    profile_name IN VARCHAR2,
```
sql_text          IN    CLOB,
enable            IN BOOLEAN DEFAULT TRUE);

Parameters

Table 156-14 ENABLE_SQL_TRANSLATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>sql_text</td>
<td>SQL statement</td>
</tr>
<tr>
<td>enable</td>
<td>Enable or disable the translation</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-15 ENABLE_SQL_TRANSLATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Examples

BEGIN
    DBMS_SQL_TRANSLATOR.ENABLE_SQL_TRANSLATION(
        profile_name => 'tsql_application',
        sql_text     => 'select top 5 * from emp',
        enable       => TRUE);
END;

156.6.7 EXPORT_PROFILE Procedure

This procedure exports the content of a SQL translation profile.

Syntax

DBMS_SQL_TRANSLATOR.EXPORT_PROFILE (  
    profile_name IN VARCHAR2,  
    content OUT NOCOPY CLOB);

Parameters

Table 156-16 EXPORT_PROFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>content</td>
<td>Content of profile</td>
</tr>
</tbody>
</table>
Exceptions

Table 156-17  EXPORTPROFILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Usage Notes

• The content of the SQL translation profile is exported in XML format as follows. Note that the profile name will not be exported.

```xml
<SQLTranslationProfile Translator="translator package name"
ForeignSQLSyntax="TRUE|FALSE"
TranslateNewSQL="TRUE|FALSE"
RaiseTranslationError="TRUE|FALSE"
LogTranslationError="TRUE|FALSE"
TraceTranslation="TRUE|FALSE"
Editionable="TRUE|FALSE"/>

<SQLTranslations>
  <SQLTranslation Enabled="TRUE|FALSE">
    <SQLText>original SQL text</SQLText>
    <TranslatedText>translated SQL text</TranslatedText>
  </SQLTranslation>
  ...
</SQLTranslations>

<ErrorTranslations>
  <ErrorTranslation Enabled="TRUE|FALSE">
    <ErrorCode>Oracle error code</ErrorCode>
    <TranslatedCode>translated error code</TranslatedCode>
    <TranslatedSQLSTATE>translated SQLSTATE</TranslatedSQLSTATE>
  </ErrorTranslation>
  ...
</ErrorTranslations>
</SQLTranslationProfile>
```

• To import the content to a SQL translation profile, use the IMPORT_PROFILE Procedure.

Examples

```sql
DECLARE
  content CLOB;
BEGIN
  DBMS_SQL_TRANSLATOR.EXPORT_PROFILE(
    profile_name => 'tsql_application',
    content      => content);
END;
```
156.6.8 IMPORT_PROFILE Procedure

This procedure imports the content of a SQL translation profile.

Syntax

DBMS_SQL_TRANSLATOR.IMPORT_PROFILE (  
    profile_name       IN   VARCHAR2,  
    content            IN   CLOB);  

Parameters

Table 156-18  IMPORT_PROFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>content</td>
<td>Content of profile</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-19  IMPORT_PROFILE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
</tbody>
</table>

Usage Notes

- The content of the SQL translation profile must be in XML format as used by the EXPORT_PROFILE Procedure. All elements and attributes are optional.
- If the profile does not exist, it is created. If it exists, the content overrides any existing attribute, translator package, SQL or error translation registration.
- To export the content to a SQL translation profile, use the EXPORT_PROFILE Procedure.

Examples

DECLARE  
    content CLOB;  
BEGIN  
    DBMS_SQL_TRANSLATOR.IMPORT_PROFILE(  
        profile_name => 'tsql_application',  
        content      => content);  
END;
156.6.9 REGISTER_ERROR_TRANSLATION Procedure

This procedure registers a custom translation of an Oracle error code and SQLSTATE in a SQL translation profile.

Syntax

```
DBMS_SQL_TRANSLATOR.REGISTER_ERROR_TRANSLATION (  
  profile_name          IN   VARCHAR2,  
  error_code            IN   PLS_INTEGER,  
  translated_code       IN   PLS_INTEGER DEFAULT NULL,  
  translated_sqlstate   IN   VARCHAR2 DEFAULT NULL,  
  enable                IN   BOOLEAN DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>error_code</td>
<td>Oracle error code</td>
</tr>
<tr>
<td>translated_code</td>
<td>Translated error code</td>
</tr>
<tr>
<td>translated_sqlstate</td>
<td>Translated SQLSTATE</td>
</tr>
<tr>
<td>enable</td>
<td>Enable or disable the translation</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Usage Notes

- When the Oracle Database translates an Oracle error code using a translation profile, it searches for the registered custom translation first, and only invokes the translator package if no match is found.
- When a translation is registered in a profile, it may be disabled. Oracle Database does not search for disabled translations.
- The old translation of the error code and SQLSTATE, if present, is replaced with the new translation.
- To deregister a translation, use the DEREGISTER_ERROR_TRANSLATION Procedure.
Examples

BEGIN
    DBMS_SQL_TRANSLATOR.REGISTER_ERROR_TRANSLATION(
        profile_name => 'tsql_application',
        error_code   => 1,
        translated_code => 2601);
END;

156.6.10 REGISTER_SQL_TRANSLATION Procedure

This procedure registers a custom translation of a SQL statement in a SQL translation profile.

Syntax

DBMS_SQL_TRANSLATOR.REGISTER_SQL_TRANSLATION (  
    profile_name      IN VARCHAR2,
    sql_text          IN CLOB,
    translated_text   IN CLOB DEFAULT NULL,
    enable            IN BOOLEAN DEFAULT TRUE);

Parameters

Table 156-22  REGISTER_SQL_TRANSLATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>sql_text</td>
<td>SQL statement</td>
</tr>
<tr>
<td>translated_text</td>
<td>Translated SQL statement</td>
</tr>
<tr>
<td>enable</td>
<td>Enable or disable the translation</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-23  REGISTER_SQL_TRANSLATION Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Usage Notes

- When the Oracle Database translates a statement using a translation profile, it searches for the registered custom translation first, and only invokes the translator package if no match is found.
- When a translation is registered in a profile, it may be disabled. Oracle Database does not search for disabled translations.
• When translated_text is NULL, no translation is required and the original statement is used.
• The old translation of the SQL statement, if present, is replaced with the new translation.
• To deregister a translation, use the Deregister_SQL_Translation Procedure.

Examples

BEGIN
    DBMS_SQL_TRANSLATOR.REGISTER_SQL_TRANSLATION(
        profile_name    => 'tsql_application',
        sql_text        => 'select top 5 * from emp',
        translated_text => 'SELECT * FROM emp WHERE rownum <= :SYS_N_001');
END;

156.6.11 SET_ATTRIBUTE Procedure

This procedure sets an attribute of a SQL translation profile.

Syntax

DBMS_SQL_TRANSLATOR.SET_ATTRIBUTE (  
    profile_name      IN  VARCHAR2,  
    attribute_name    IN  VARCHAR2,  
    attribute_value   IN  VARCHAR2;)

Parameters

Table 156-24 SET_ATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>Name of profile</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Name of attribute</td>
</tr>
<tr>
<td>attribute_value</td>
<td>Value of attribute</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-25 SET_ATTRIBUTE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Usage Notes

See Constants
156.6.12 SQL_HASH Function

This procedure computes the hash value of a SQL statement in the session's SQL translation profile.

Syntax

```
DBMS_SQL_TRANSLATOR.SQL_HASH (  
    sql_text         IN    CLOB)  
RETURN NUMBER DETERMINISTIC;
```

Parameters

Table 156-26  SQL_HASH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>SQL statement</td>
</tr>
</tbody>
</table>

Return Values

Returns hash value of the SQL statement in the SQL translation profile

Exceptions

Table 156-27  SQL_HASH Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
</tbody>
</table>

Examples

```
DECLARE
    sqltext CLOB;
    txltext CLOB;
    sqlhash NUMBER;
BEGIN
    sqltext  :=  'SELECT TOP 1 * FROM emp';
    sqlhash  :=  DBMS_SQL_TRANSLATOR.SQL_HASH (sqltext);
    SELECT translated_text INTO txltext
        FROM user_sql_translations
        WHERE sql_hash = sqlhash
        AND DBMS_LOB.COMpare (sql_text, sqltext) = 0;
END;
```

156.6.13 SQL_ID Function

This procedure computes the SQL identifier of a SQL statement in a SQL translation profile.

Syntax

```
DBMS_SQL_TRANSLATOR.SQL_ID (  
    sql_text         IN    CLOB)  
RETURN VARCHAR2 DETERMINISTIC;
```
Parameters

Table 156-28 SQL_ID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>SQL statement</td>
</tr>
</tbody>
</table>

Return Values

Returns the SQL ID of the SQL statement in the SQL translation profile

Exceptions

Table 156-29 SQL_ID Function Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
</tbody>
</table>

Examples

DECLARE
    sqltext CLOB;
    sqlid VARCHAR2(13);
BEGIN
    sqltext := 'SELECT TOP 1 * FROM emp';
    sqlid := DBMS_SQL_TRANSLATOR.SQL_ID(sqltext);
END;

156.6.14 TRANSLATE_ERROR Procedure

This procedure translates an Oracle error code and an ANSI SQLSTATE using the session's SQL translation profile

Syntax

DBMS_SQL_TRANSLATOR.TRANSLATE_ERROR (
    error_code IN PLS_INTEGER,
    translated_code OUT PLS_INTEGER,
    translated_sqlstate OUT NOCOPY VARCHAR2);

Parameters

Table 156-30 TRANSLATE_ERROR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_code</td>
<td>Oracle error code</td>
</tr>
<tr>
<td>translated_code</td>
<td>Translated error code</td>
</tr>
<tr>
<td>translated_sqlstate</td>
<td>Translated SQLSTATE</td>
</tr>
</tbody>
</table>
Exceptions

Table 156-31 TRANSLATE_ERROR Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
<tr>
<td>NO_TRANSLATION_FOUND</td>
<td>No translation of the SQL statement or error code is found</td>
</tr>
</tbody>
</table>

Examples

DECLARE
    translated_code      BINARY_INTEGER;
    translated_sqlstate  VARCHAR2(5);
BEGIN
    DBMS_SQL_TRANSLATOR.TRANSLATE_ERROR(
        error_code           => 1,
        translated_code      => translated_code,
        translated_sqlstate  => translated_sqlstate);
END;

156.6.15 TRANSLATE_SQL Procedure

This procedure translates a SQL statement using a SQL translation profile.

Syntax

DBMS_SQL_TRANSLATOR.TRANSLATE_SQL (
    sql_text          IN           CLOB,
    translated_text   OUT  NOCOPY  CLOB);

Parameters

Table 156-32 TRANSLATE_SQL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>SQL statement</td>
</tr>
<tr>
<td>translated_text</td>
<td>Translated SQL statement</td>
</tr>
</tbody>
</table>

Exceptions

Table 156-33 TRANSLATE_SQL Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument is passed to the PL/SQL interface</td>
</tr>
<tr>
<td>INSUFFICIENT_PRIVILEGE</td>
<td>User has insufficient privilege for the operation</td>
</tr>
<tr>
<td>NO_SUCH_USER</td>
<td>Profile owner does not exist</td>
</tr>
</tbody>
</table>
Table 156-33  (Cont.) TRANSLATE_SQL Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_SUCH_PROFILE</td>
<td>Profile does not exist</td>
</tr>
</tbody>
</table>

Examples

```sql
ALTER SESSION SET SQL_TRANSLATION_PROFILE = tsql_application;

DECLARE
    translated_text CLOB;
BEGIN
    DBMS_SQL_TRANSLATOR.TRANSLATE_SQL(
        sql_text => 'select top 5 * from emp',
        translated_text => translated_text);
END;
```
The DBMS_SQLDIAG package provides an interface to the SQL Diagnosability functionality.

This chapter contains the following topics:

- DBMS_SQLDIAG Overview
- DBMS_SQLDIAG Security Model
- DBMS_SQLDIAG Constants
- DBMS_SQLDIAG Examples
- Summary of DBMS_SQLDIAG Subprograms

See Also:

Oracle Database Administrator's Guide for more information about "Managing Diagnostic Data"

157.1 DBMS_SQLDIAG Overview

In the rare case that a SQL statement fails with a critical error, you can run the SQL Repair Advisor to try to repair the failed statement.

This section covers the following topics:

- About the SQL Repair Advisor
- Running the SQL Repair Advisor
- Removing a SQL Patch

About the SQL Repair Advisor

You run the SQL Repair Advisor after a SQL statement fails with a critical error. The advisor analyzes the statement and in many cases recommends a patch to repair the statement. If you implement the recommendation, the applied SQL patch circumvents the failure by causing the query optimizer to choose an alternate execution plan for future executions.

Running the SQL Repair Advisor

You run the SQL Repair Advisor by creating and executing a diagnostic task using the CREATE_DIAGNOSIS_TASK and EXECUTE_DIAGNOSIS_TASK respectively. The SQL Repair Advisor first reproduces the critical error and then tries to produce a workaround in the form of SQL patch.

1. Identify the problem SQL statement

Consider the SQL statement that gives a critical error:
DELETE FROM t t1 WHERE t1.a = 'a' AND ROWID <> (SELECT MAX(ROWID) FROM t t2
WHERE t1.a = t2.a
AND t1.b = t2.b AND t1.d = t2.d)

You use the SQL Repair advisor to repair this critical error.

2. Create a diagnosis task

Invoke DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK. You can specify an optional task name, an optional time limit for the advisor task, and problem type. In the example below, we specify the SQL text, the task name as 'error_task' and a problem type as 'DBMS_SQLDIAG.PROBLEM_TYPE_COMPILATION_ERROR'.

```sql
DECLARE
    rep_out        CLOB;
    t_id            VARCHAR2(50);
BEGIN
    t_id := DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK(
        sql_text => 'DELETE FROM t t1 WHERE t1.a = ''a'' AND ROWID <> (SELECT
            MAX(ROWID) FROM t t2
            WHERE t1.a = t2.a
            AND t1.b = t2.b
            AND t1.d = t2.d)
        ',
        task_name => 'error_task',
        problem_type =>DBMS_SQLDIAG.PROBLEM_TYPE_COMPILATION_ERROR);
END;
```

3. Execute the diagnosis task

To execute the workaround generation and analysis phase of the SQL Repair Advisor, you call DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK with the task ID returned by the CREATE_DIAGNOSIS_TASK. After a short delay, the SQL Repair Advisor returns. As part of its execution, the SQL Repair Advisor keeps a record of its findings which can be accessed through the reporting facilities of SQL Repair Advisor.

```sql
DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK (t_id);
```

4. Report the diagnosis task

The analysis of the diagnosis task is accessed through dbms_sqldiag.report_diagnosis_task. If the SQL Repair Advisor was able to find a workaround, it recommends a SQL Patch. A SQL Patch is similar to a SQL profile but unlike the SQL Profile, it is used to workaround compilation or execution errors.

```sql
rep_out := DBMS_SQLDIAG.REPORT_DIAGNOSIS_TASK (t_id, DBMS_SQLDIAG.TYPE_TEXT);

   DBMS_OUTPUT.PUT_LINE ('Report : ' || rep_out);
END;
```

5. Applying the patch

If a patch recommendation is present in the report, you can run the ACCEPT_SQL_PATCH command to accept the patch by invoking DBMS_SQLDIAG.ACCEPT_SQL_PATCH. This procedure takes the task_name as an argument.

```sql
EXECUTE DBMS_SQLDIAG.ACCEPT_SQL_PATCH(task_name => 'error_task', task_owner => 'SYS', replace => TRUE);
```

6. Test the patch
Now that you have accepted the patch, you can rerun the SQL statement. This time, it will not give you the critical error. If you run 'explain plan' for this statement, you will see that a SQL patch was used to generate the plan.

```
DELETE FROM t t1 WHERE t1.a = 'a' AND ROWID <> (select max(rowid) FROM t t2 WHERE t1.a= t2.a
AND t1.b = t2.b AND t1.d=t2.d);
```

Removing a SQL Patch

In a situation where you obtained an official patch from Oracle to fix an error, or upgraded to the next patchset or release of Oracle which included the fix for the error, you call `DBMS_SQLDIAG DROP_SQL_PATCH` with the patch name to drop the SQL patch. The patch name can be obtained from the explain plan section or by querying the view `DBA_SQL_PATCHES`.

157.2 DBMS_SQLDIAG Security Model

You must have the `ADVISOR` role to execute the `DBMS_SQLDIAG` package.

157.3 DBMS_SQLDIAG Constants

`DBMS_SQLDIAG` defines constants to use when specifying parameter values.

These constants are shown in the following tables:

- Table 157-1 describes the name of SQL repair advisor as seen by the advisor framework
- Table 157-2 describes SQLDIAG advisor task scope parameter values
- Table 157-3 describes SQLDIAG advisor `time_limit` constants
- Table 157-4 describes possible formats for a report
- Table 157-5 describes possible levels of detail in the report
- Table 157-6 describes possible report sections (comma delimited)
- Table 157-7 describes possible values for the `problem_type` parameter of the `CREATE_DIAGNOSIS_TASK` Functions
- Table 157-8 describes possible values for the `_sql_findings_mode` parameter

### Table 157-1  DBMS_SQLDIAG Constants - SQLDIAG Advisor Name

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV_SQL_DIAG_NAME</td>
<td>VARCHAR2(18)</td>
<td>SQL Repair Advisor</td>
<td>Name of SQL repair advisor as seen by the advisor framework</td>
</tr>
</tbody>
</table>

### Table 157-2  DBMS_SQLDIAG Constants - SQLDIAG Advisor Task Scope Parameter Values

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOPE_COMPREHENSIVE</td>
<td>VARCHAR2(13)</td>
<td>COMPREHENSIVE</td>
<td>Detailed analysis of the problem which may take more time to execute</td>
</tr>
</tbody>
</table>
### Table 157-2  DBMS_SQLDIAG Constants - SQLDIAG Advisor Task Scope Parameter Values

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOPE_LIMITED</td>
<td>VARCHAR2(7)</td>
<td>LIMITED</td>
<td>Brief analysis of the problem</td>
</tr>
</tbody>
</table>

### Table 157-3  DBMS_SQLDIAG Constants - SQLDIAG Advisor time_limit Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_LIMIT_DEFAULT</td>
<td>NUMBER</td>
<td>1800</td>
<td>Default time limit for analysis of the problem</td>
</tr>
</tbody>
</table>

### Table 157-4  DBMS_SQLDIAG Constants - Report Type (possible values) Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_HTML</td>
<td>VARCHAR2(4)</td>
<td>HTML</td>
<td>Report from the REPORT_DIAGNOSIS_TASK Function in HTML form</td>
</tr>
<tr>
<td>TYPE_TEXT</td>
<td>VARCHAR2(4)</td>
<td>TEXT</td>
<td>Report from the REPORT_DIAGNOSIS_TASK Function in text form</td>
</tr>
<tr>
<td>TYPE_XML</td>
<td>VARCHAR2(3)</td>
<td>XML</td>
<td>Report from the REPORT_DIAGNOSIS_TASK Function in XML form</td>
</tr>
</tbody>
</table>

### Table 157-5  DBMS_SQLDIAG Constants - Report Level (possible values) Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL_ALL</td>
<td>VARCHAR2(3)</td>
<td>ALL</td>
<td>Complete report including annotations about statements skipped over</td>
</tr>
<tr>
<td>LEVEL_BASIC</td>
<td>VARCHAR2(5)</td>
<td>BASIC</td>
<td>Shows information about every statement analyzed, including recommendations not implemented</td>
</tr>
<tr>
<td>LEVEL_TYPICAL</td>
<td>VARCHAR2(7)</td>
<td>TYPICAL</td>
<td>Simple report shows only information about the actions taken by the advisor.</td>
</tr>
</tbody>
</table>
Table 157-6  DBMS_SQLDIAG Constants - Report Section (possible values) Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION_ALL</td>
<td>VARCHAR2(3)</td>
<td>ALL</td>
<td>All statements</td>
</tr>
<tr>
<td>SECTION_ERRORS</td>
<td>VARCHAR2(6)</td>
<td>ERRORS</td>
<td>Statements with errors</td>
</tr>
<tr>
<td>SECTION_FINDINGS</td>
<td>VARCHAR2(8)</td>
<td>FINDINGS</td>
<td>Tuning findings</td>
</tr>
<tr>
<td>SECTION_INFORMATION</td>
<td>VARCHAR2(11)</td>
<td>INFORMATION</td>
<td>General information</td>
</tr>
<tr>
<td>SECTION_PLANS</td>
<td>VARCHAR2(5)</td>
<td>PLANS</td>
<td>Explain plans</td>
</tr>
<tr>
<td>SECTION_SUMMARY</td>
<td>VARCHAR2(7)</td>
<td>SUMMARY</td>
<td>Summary information</td>
</tr>
</tbody>
</table>

Table 157-7  DBMS_SQLDIAG Constants - Problem Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBLEM_TYPE_PERFORMANCE</td>
<td>NUMBER</td>
<td>1</td>
<td>User suspects this is a performance problem</td>
</tr>
<tr>
<td>PROBLEM_TYPE_WRONG_RESULTS</td>
<td>NUMBER</td>
<td>2</td>
<td>User suspects the query is giving inconsistent results</td>
</tr>
<tr>
<td>PROBLEM_TYPE_COMPILATION_ERROR</td>
<td>NUMBER</td>
<td>3</td>
<td>User sees a crash in compilation</td>
</tr>
<tr>
<td>PROBLEM_TYPE_EXECUTION_ERROR</td>
<td>NUMBER</td>
<td>4</td>
<td>User sees a crash in execution</td>
</tr>
<tr>
<td>PROBLEM_TYPE_ALT_PLAN_GEN</td>
<td>NUMBER</td>
<td>5</td>
<td>User to explore all alternative plans</td>
</tr>
</tbody>
</table>

Table 157-8  DBMS_SQLDIAG Constants - Findings Filter Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLDIAG_FINDINGS_ALL</td>
<td>NUMBER</td>
<td>1</td>
<td>Show all possible findings</td>
</tr>
<tr>
<td>SQLDIAG_FINDINGS_VALIDATION</td>
<td>NUMBER</td>
<td>2</td>
<td>Show status of validation rules over structures</td>
</tr>
<tr>
<td>SQLDIAG_FINDINGS_FEATURES</td>
<td>NUMBER</td>
<td>3</td>
<td>Show only features used by the query</td>
</tr>
<tr>
<td>SQLDIAG_FINDINGS_FILTER_PLANS</td>
<td>NUMBER</td>
<td>4</td>
<td>Show the alternative plans generated by the advisor</td>
</tr>
<tr>
<td>SQLDIAG_FINDINGS_CR_DIFF</td>
<td>NUMBER</td>
<td>5</td>
<td>Show difference between two plans</td>
</tr>
<tr>
<td>SQLDIAG_FINDINGS_MASK_VARIANT</td>
<td>NUMBER</td>
<td>6</td>
<td>Mask info for testing</td>
</tr>
<tr>
<td>SQLDIAG_FINDINGS_OBJ_FEATURES</td>
<td>NUMBER</td>
<td>7</td>
<td>Show features usage history</td>
</tr>
</tbody>
</table>
157.4 DBMS_SQLDIAG Examples

Patches can be exported out of one system and imported into another by means of a staging table, provided by subprograms in this package. Like with SQL diagnosis sets, the operation of inserting into the staging table is called a "pack", and the operation of creating patches from staging table data is termed the "unpack".

Patch Pack / Unpack

DBAs should perform a pack/unpack as follows:

1. Create a staging table owned by user 'SH' through a call to CREATE_STGTAB_SQLPATCH:

   EXEC DBMS_SQLDIAG.CREATE_STGTAB_SQLPATCH(
   table_name          =>  'STAGING_TABLE',
   schema_name         =>  'SH');

2. Call PACK_STGTAB_SQLPATCH one or more times to write SQL patch data into the staging table. In this case, copy data for all SQL patches in the DEFAULT category into a staging table owned by the current schema owner:

   EXEC DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH(
   staging_table_name  =>  'STAGING_TABLE');

3. In this case, only a single SQL patch SP_FIND_EMPLOYEE is copied into a staging table owned by the current schema owner:

   EXEC DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH(
   patch_name          =>  'SP_FIND_EMPLOYEE',
   staging_table_name  =>  'STAGING_TABLE');

   The staging table can then be moved to another system using either datapump, import/export commands or through a databaselink.

4. Call UNPACK_STGTAB_SQLPATCH to create SQL patches on the new system from the patch data in the staging table. In this case, change the name in the data for the SP_FIND_EMPLOYEE patch stored in the staging table to 'SP_FIND_EMP_PROD':

   EXEC DBMS_SQLDIAG.REMAP_STGTAB_SQLPATCH(
   old_patch_name      =>  'SP_FIND_EMPLOYEE',
   new_patch_name      =>  'SP_FIND_EMP_PROD');

157.5 Summary of DBMS_SQLDIAG Subprograms

This table lists the DBMS_SQLDIAG subprograms and briefly describes them.
Table 157-9  DBMS_SQLDIAG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT_SQL_PATCH Function &amp; Procedure</td>
<td>Accepts a recommended SQL patch as recommended by the specified SQL diagnosis task</td>
</tr>
<tr>
<td>ALTER_SQL_PATCH Procedure</td>
<td>Alters specific attributes of an existing SQL patch object</td>
</tr>
<tr>
<td>CANCEL_DIAGNOSIS_TASK Procedure</td>
<td>Cancels a diagnostic task</td>
</tr>
<tr>
<td>CREATE_DIAGNOSIS_TASK Functions</td>
<td>Creates a diagnostic task in order to diagnose a single SQL statement</td>
</tr>
<tr>
<td>CREATE_SQL_PATCH Function</td>
<td>Creates an SQL patch based on a set of user specified hints for specific statements identified by SQL text.</td>
</tr>
<tr>
<td>CREATE_STGTAB_SQLPATCH Procedure</td>
<td>Creates the staging table used for transporting SQL patches from one system to another</td>
</tr>
<tr>
<td>DROP_DIAGNOSIS_TASK Procedure</td>
<td>Drops a diagnostic task</td>
</tr>
<tr>
<td>DROP_SQL_PATCH Procedure</td>
<td>Drops the named SQL patch from the database</td>
</tr>
<tr>
<td>EXECUTE_DIAGNOSIS_TASK Procedure</td>
<td>Executes a diagnostic task</td>
</tr>
<tr>
<td>EXPLAIN_SQL_TESTCASE Function</td>
<td>Explains a SQL test case</td>
</tr>
<tr>
<td>EXPORT_SQL_TESTCASE Procedures</td>
<td>Exports a SQL test case to a directory</td>
</tr>
<tr>
<td>EXPORT_SQL_TESTCASE_DIR_BY_INC Function</td>
<td>Generates a SQL Test Case corresponding to the incident ID passed as an argument.</td>
</tr>
<tr>
<td>EXPORT_SQL_TESTCASE_DIR_BY_TXT Function</td>
<td>Generates a SQL Test Case corresponding to the SQL passed as an argument</td>
</tr>
<tr>
<td>GET_FIX_CONTROL Function</td>
<td>Returns the value of fix control for a given bug number</td>
</tr>
<tr>
<td>GET_SQL Function</td>
<td>Imports a SQL test case</td>
</tr>
<tr>
<td>IMPORT_SQL_TESTCASE Procedures</td>
<td>Imports a SQL test case into a schema</td>
</tr>
<tr>
<td>INCIDENTID_2_SQL Procedure</td>
<td>Initializes a sql_setrow from an incident ID</td>
</tr>
<tr>
<td>INTERRUPT_DIAGNOSIS_TASK Procedure</td>
<td>Interrupts a diagnostic task</td>
</tr>
<tr>
<td>LOAD_SQLSET_FROM_TCB Function</td>
<td>Loads a SQLSET from Test Case Builder (TCB) file</td>
</tr>
<tr>
<td>PACK_STGTAB_SQLPATCH Procedure</td>
<td>SQL patches into the staging table created by the CREATE_STGTAB_SQLPATCH Procedure</td>
</tr>
<tr>
<td>REPLAY_SQL_TESTCASE Procedure</td>
<td>Reports on a diagnostic task</td>
</tr>
<tr>
<td>REPORT_DIAGNOSIS_TASK Function</td>
<td>Reports on a diagnostic task</td>
</tr>
<tr>
<td>RESET_DIAGNOSIS_TASK Procedure</td>
<td>Resets a diagnostic task</td>
</tr>
<tr>
<td>RESUME_DIAGNOSIS_TASK Procedure</td>
<td>Resumes a diagnostic task</td>
</tr>
<tr>
<td>SET_DIAGNOSIS_TASK_PARAMETER Procedure</td>
<td>Sets a diagnosis task parameter</td>
</tr>
</tbody>
</table>
Table 157-9  (Cont.) DBMS_SQLDIAG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNPACK_STGTAB_SQLPATCH Procedure</td>
<td>Unpacks from the staging table populated by a call to the PACK_STGTAB_SQLPATCH Procedure, using the patch data stored in the staging table to create patches on this system</td>
</tr>
</tbody>
</table>

157.5.1 ACCEPT_SQL_PATCH Function & Procedure

This procedure accepts a recommended SQL patch as recommended by the specified SQL diagnosis task.

Syntax

```sql
DBMS_SQLDIAG.ACCEPT_SQL_PATCH (  
    task_name      IN  VARCHAR2,  
    object_id      IN  NUMBER := NULL,  
    name           IN  VARCHAR2 := NULL,  
    description    IN  VARCHAR2 := NULL,  
    category       IN  VARCHAR2 := NULL,  
    task_owner     IN  VARCHAR2 := NULL,  
    replace        IN  BOOLEAN := FALSE,  
    force_match    IN  BOOLEAN := FALSE)  
RETURN VARCHAR2;
```

Parameters

Table 157-10  ACCEPT_SQL_PATCH Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Name of the SQL diagnosis task</td>
</tr>
<tr>
<td>object_id</td>
<td>Identifier of the advisor framework object representing the SQL statement associated to the diagnosis task</td>
</tr>
<tr>
<td>name</td>
<td>Name of the patch. It cannot contain double quotation marks. The name is case sensitive. If not specified, the system will generate a unique name for the SQL patch.</td>
</tr>
<tr>
<td>description</td>
<td>User specified string describing the purpose of this SQL patch. Maximum size of description is 500.</td>
</tr>
</tbody>
</table>
Table 157-10  (Cont.) ACCEPT_SQL_PATCH Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>Category name which must match the value of the SQLDIAGNOSE_CATEGORY parameter in a session for the session to use this patch. It defaults to the value DEFAULT. This is also the default of the SQLDIAGNOSE_CATEGORY parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name create a unique key for a patch. An accept will fail if this combination is duplicated.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Owner of the diagnosis task. This is an optional parameter that has to be specified to accept a SQL Patch associated to a diagnosis task owned by another user. The current user is the default value.</td>
</tr>
<tr>
<td>replace</td>
<td>If the patch already exists, it will be replaced if this argument is TRUE. It is an error to pass a name that is already being used for another signature/category pair, even with replace set to TRUE.</td>
</tr>
<tr>
<td>force_match</td>
<td>If TRUE this causes SQL Patches to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the FORCE option of the CURSOR_SHARING parameter. If FALSE, literals are not transformed. This is analogous to the matching algorithm used by the EXACT option of the CURSOR_SHARING parameter.</td>
</tr>
</tbody>
</table>

Return Values
Name of the SQL patch

Usage Notes
Requires CREATE ANY SQL PROFILE privilege

157.5.2 ALTER_SQL_PATCH Procedure

This procedure alters specific attributes of an existing SQL patch object.

Syntax

```sql
DBMS_SQLDIAG.ALTER_SQL_PATCH (
    name            IN  VARCHAR2,
    attribute_name  IN  VARCHAR2,
    value           IN  VARCHAR2);
```

Parameters

Table 157-11  ALTER_SQL_PATCH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of SQL patch to alter.</td>
</tr>
</tbody>
</table>
Table 157-11  (Cont.) ALTER_SQL_PATCH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_name</td>
<td>Name of SQL patch to alter. Possible values:</td>
</tr>
<tr>
<td></td>
<td>• STATUS -&gt; can be set to ENABLED or DISABLED</td>
</tr>
<tr>
<td></td>
<td>• NAME -&gt; can be reset to a valid name (must be a valid Oracle identifier and must be unique).</td>
</tr>
<tr>
<td></td>
<td>• DESCRIPTION -&gt; can be set to any string of size no more than 500</td>
</tr>
<tr>
<td></td>
<td>• CATEGORY -&gt; can be reset to a valid category name (must be valid Oracle identifier and must be unique when combined with normalized SQL text)</td>
</tr>
<tr>
<td></td>
<td>This parameter is mandatory and is case sensitive.</td>
</tr>
<tr>
<td>value</td>
<td>New value of the attribute. See attribute_name for valid attribute values. This parameter is mandatory.</td>
</tr>
</tbody>
</table>

Usage Notes

Requires ALTER ANY SQL PATCH privilege

157.5.3 CANCEL_DIAGNOSIS_TASK Procedure

This procedure cancels a diagnostic task.

Syntax

```sql
DBMS_SQLDIAG.CANCEL_DIAGNOSIS_TASK (taskname        IN   VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Name of task</td>
</tr>
</tbody>
</table>

157.5.4 CREATE_DIAGNOSIS_TASK Functions

This function creates a diagnostic task in order to diagnose a single SQL statement. It returns a SQL diagnosis task unique name

Syntax

Prepares the diagnosis of a single statement given its text:

```sql
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (sql_text IN CLOB, bind_list IN sql_binds := NULL, user_name IN VARCHAR2 := NULL, scope IN VARCHAR2 := SCOPE_COMPREHENSIVE, time_limit IN NUMBER := TIME_LIMIT_DEFAULT, task_name IN VARCHAR2 := NULL,
```
Prepares the diagnosis of a single statement from the Cursor Cache given its identifier:

```
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
  sql_id             IN   VARCHAR2,  
  plan_hash_value    IN   NUMBER    := NULL, 
  scope              IN   VARCHAR2  := SCOPE_COMPREHENSIVE, 
  time_limit         IN   NUMBER    := TIME_LIMIT_DEFAULT, 
  task_name          IN   VARCHAR2  := NULL, 
  description        IN   VARCHAR2  := NULL, 
  problem_type       IN   NUMBER    := PROBLEM_TYPE_PERFORMANCE) 
RETURN VARCHAR2;
```

Prepares the diagnosis of a Sqlset:

```
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
  sqlset_name       IN VARCHAR2, 
  basic_filter      IN VARCHAR2 :=  NULL, 
  object_filter     IN VARCHAR2 :=  NULL, 
  rank1             IN VARCHAR2 :=  NULL, 
  rank2             IN VARCHAR2 :=  NULL, 
  rank3             IN VARCHAR2 :=  NULL, 
  result_percentage IN NUMBER   :=  NULL, 
  result_limit      IN NUMBER   :=  NULL, 
  scope             IN VARCHAR2 :=  SCOPE_COMPREHENSIVE, 
  time_limit        IN NUMBER   :=  TIME_LIMIT_DEFAULT, 
  task_name         IN VARCHAR2 :=  NULL, 
  description       IN VARCHAR2 :=  NULL, 
  plan_filter       IN VARCHAR2 :=  'MAX_ELAPSED_TIME', 
  sqlset_owner      IN VARCHAR2 :=  NULL, 
  problem_type      IN NUMBER   :=  PROBLEM_TYPE_PERFORMANCE)  RETURN VARCHAR2;
```

Parameters

**Table 157-13   CREATE_DIAGNOSIS_TASK Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>Text of a SQL statement</td>
</tr>
<tr>
<td>bind_list</td>
<td>Set of bind values</td>
</tr>
<tr>
<td>user_name</td>
<td>Username for who the statement/sqlset will be diagnosed</td>
</tr>
<tr>
<td>scope</td>
<td>Diagnosis scope (limited/comprehensive)</td>
</tr>
<tr>
<td>time_limit</td>
<td>Maximum duration in seconds for the diagnosis session</td>
</tr>
<tr>
<td>task_name</td>
<td>Optional diagnosis task name</td>
</tr>
<tr>
<td>description</td>
<td>Maximum of 256 SQL diagnosis session description</td>
</tr>
<tr>
<td>problem_type</td>
<td>Determines the goal of the task. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· PROBLEM_TYPE_WRONG_RESULTS</td>
</tr>
<tr>
<td></td>
<td>· PROBLEM_TYPE_COMPILATION_ERROR</td>
</tr>
<tr>
<td></td>
<td>· PROBLEM_TYPE_EXECUTION_ERROR</td>
</tr>
<tr>
<td>sql_id</td>
<td>Identifier of the statement</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Hash value of the SQL execution plan</td>
</tr>
</tbody>
</table>
### Table 157-13  (Cont.) CREATE_DIAGNOSIS_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Sqlset name</td>
</tr>
<tr>
<td>basic_filter</td>
<td>SQL predicate to filter the SQL from the SQL tuning set (STS)</td>
</tr>
<tr>
<td>object_filter</td>
<td>Object filter</td>
</tr>
<tr>
<td>rank(i)</td>
<td>Order-by clause on the selected SQL</td>
</tr>
<tr>
<td>result_percentage</td>
<td>Percentage on the sum of a ranking measure</td>
</tr>
<tr>
<td>result_limit</td>
<td>Top L(imit) SQL from (filtered/ranked) SQL</td>
</tr>
</tbody>
</table>
| plan_filter    | Plan filter. It is applicable in case there are multiple plans (plan_hash_value). This filter allows selecting one plan (plan_hash_value) only. Possible values are:  
  - LAST_GENERATED: plan with most recent timestamp  
  - FIRST_GENERATED: opposite to LAST_GENERATED  
  - LAST_LOADED: plan with most recent first_load_time stat info  
  - FIRST_LOADED: opposite to LAST_LOADED  
  - MAX_ELAPSED_TIME: plan with maximum elapsed time  
  - MAX_BUFFER_GETS: plan with maximum buffer gets  
  - MAX_DISK_READS: plan with maximum disk reads  
  - MAX_DIRECT_WRITES: plan with maximum direct writes  
  - MAX_OPTIMIZER_COST: plan with maximum optimum cost |
| sqlset_owner   | Owner of the sqlset, or null for current schema owner                       |

#### 157.5.5 CREATE_SQL_PATCH Function

This function creates a SQL patch based on a set of user specified hints for specific statements identified by SQL text.

A SQL patch is usually created automatically by the SQL Repair Advisor to prevent any errors during the compilation or execution of a SQL statement. This function provides a way to manually create a SQL patch based on a set of hints that resolves the error.

**Syntax**

```sql
DBMS_SQLDIAG.CREATE_SQL_PATCH (    
  sql_text IN CLOB,    
  hint_text IN CLOB,    
  name IN VARCHAR2 := NULL,    
  description IN VARCHAR2 := NULL,    
  category IN VARCHAR2 := NULL,    
  validate IN BOOLEAN := TRUE)    
RETURN VARCHAR2;
```

```sql
DBMS_SQLDIAG.CREATE_SQL_PATCH (    
  sql_id IN VARCHAR2,    
  hint_text IN CLOB,    
  name IN VARCHAR2 := NULL,    
  description IN VARCHAR2 := NULL,    
  category IN VARCHAR2 := NULL,    
```
validate | IN BOOLEAN := TRUE)
RETURN VARCHAR2;

Parameters

Table 157-14  CREATE_SQL_PATCH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>Text of the SQL statement</td>
</tr>
<tr>
<td>sql_id</td>
<td>The SQL identifier for the SQL statement</td>
</tr>
<tr>
<td>hint_text</td>
<td>Hints to include in the SQL patch</td>
</tr>
<tr>
<td>name</td>
<td>Optional SQL patch name</td>
</tr>
<tr>
<td>description</td>
<td>Description of the SQL patch</td>
</tr>
<tr>
<td>category</td>
<td>Category name</td>
</tr>
<tr>
<td>validate</td>
<td>Whether to validate the provided hints</td>
</tr>
</tbody>
</table>

Return Values

Both functions return the SQL patch name.

157.5.6 CREATE_STGTAB_SQLPATCH Procedure

This procedure creates the staging table used for transporting SQL patches from one system to another.

Syntax

```sql
DBMS_SQLDIAG.CREATE_STGTAB_SQLPATCH (    table_name       IN  VARCHAR2,    schema_name      IN  VARCHAR2 := NULL,    tablespace_name  IN  VARCHAR2 := NULL);
```

Parameters

Table 157-15  CREATE_STGTAB_SQLPATCH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>(Mandatory) Name of the table to create (case-sensitive)</td>
</tr>
<tr>
<td>schema_name</td>
<td>Schema to create the table in, or NULL for current schema (case-sensitive)</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Tablespace to store the staging table within, or NULL for current user’s default tablespace (case-sensitive)</td>
</tr>
</tbody>
</table>
157.5.7 DROP_DIAGNOSIS_TASK Procedure

This procedure drops a diagnostic task.

Syntax

```sql
DBMS_SQLDIAG.DROP_DIAGNOSIS_TASK (
    taskname  IN  VARCHAR2);
```

Parameters

Table 157-16  DROP_DIAGNOSIS_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Name of task</td>
</tr>
</tbody>
</table>

157.5.8 DROP_SQL_PATCH Procedure

This procedure drops the named SQL patch from the database.

Syntax

```sql
DBMS_SQLDIAG.DROP_SQL_PATCH (
    name     IN  VARCHAR2,   ignore   IN  BOOLEAN := FALSE);
```

Parameters

Table 157-17  DROP_SQL_PATCH Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of patch to be dropped. The name is case sensitive.</td>
</tr>
<tr>
<td>ignore</td>
<td>Ignore errors due to object not existing.</td>
</tr>
</tbody>
</table>

Usage Notes

Requires DROP ANY SQL PATCH privilege

157.5.9 EXECUTE_DIAGNOSIS_TASK Procedure

This procedure executes a diagnostic task.

Syntax

```sql
DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK (
    taskname  IN  VARCHAR2);
```
### 157.5.10 EXPLAIN_SQL_TESTCASE Function

This procedure explains a SQL test case.

**Syntax**

```sql
DBMS_SQLDIAG.EXPLAIN_SQL_TESTCASE (
   sqlTestCase        IN   CLOB)
RETURN CLOB;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sqlTestCase</code></td>
<td>XML document describing the SQL test case</td>
</tr>
</tbody>
</table>

### 157.5.11 EXPORT_SQL_TESTCASE Procedures

This procedure exports a SQL test case to a directory.

**Syntax**

This variant has to be provided with the SQL information.

```sql
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (  
   directory                IN              VARCHAR2,  
   sql_text                 IN              CLOB,  
   user_name                IN              VARCHAR2  :=  NULL,  
   bind_list                IN              sql_binds :=  NULL,  
   exportEnvironment        IN              BOOLEAN   :=  TRUE,  
   exportMetadata           IN              BOOLEAN   :=  TRUE,  
   exportData               IN              BOOLEAN   :=  FALSE,  
   exportPkgbody            IN              BOOLEAN   :=  FALSE,  
   samplingPercent          IN              NUMBER    :=  100,  
   ctrlOptions              IN              VARCHAR2  :=  NULL,  
   timeLimit                IN              NUMBER    :=  0,  
   testcase_name            IN              VARCHAR2  :=  NULL,  
   testcase                 IN OUT NOCOPY   CLOB,  
   preserveSchemaMapping    IN              BOOLEAN   :=  FALSE,  
   version                  IN              VARCHAR2  :=  'COMPATIBLE');
```

This variant extracts the SQL information from an incident file.

```sql
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (  
   directory                IN              VARCHAR2,  
   incident_id              IN              VARCHAR2,
   incident_id              IN              VARCHAR2,
   ...
```
This variant allow the SQL Test case to be generated from a cursor present in the
cursor cache. Use V$SQL to get the SQL identifier and the SQL hash value.

Parameters

Table 157-20 EXPORT_SQL_TESTCASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Directory to store the various generated files</td>
</tr>
<tr>
<td>sql_text</td>
<td>Text of the SQL statement to export</td>
</tr>
<tr>
<td>incident_id</td>
<td>Incident ID containing the offending SQL</td>
</tr>
<tr>
<td>sql_id</td>
<td>Identifier of the statement in the cursor cache</td>
</tr>
<tr>
<td>username</td>
<td>Name of the user schema to use to parse the SQL, defaults to SYS</td>
</tr>
<tr>
<td>bind_list</td>
<td>List of bind values associated to the statement</td>
</tr>
<tr>
<td>exportEnvironment</td>
<td>TRUE if the compilation environment should be exported</td>
</tr>
<tr>
<td>exportMetadata</td>
<td>TRUE if the definition of the objects referenced in the SQL should be exported</td>
</tr>
<tr>
<td>exportData</td>
<td>TRUE if the data of the objects referenced in the SQL should be exported</td>
</tr>
<tr>
<td>exportPkgbody</td>
<td>TRUE if the body of the packages referenced in the SQL are exported</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>samplingPercent</td>
<td>If is TRUE, specify the sampling percentage to use to create the dump file</td>
</tr>
<tr>
<td>ctrlOptions</td>
<td>Opaque control parameters. For example, to execute three times, set <code>ctrlOptions</code> with the following string: <code>&lt;parameter name=&quot;mexec_count&quot;&gt;3&lt;/parameter&gt;</code>.</td>
</tr>
<tr>
<td></td>
<td>• name=&quot;capture&quot; - BASIC (default) or WITH_RUNTIME_INFO. This parameter defines the mode of TCB capture.</td>
</tr>
<tr>
<td></td>
<td>• BASIC: runs as Oracle release 11g TCB and captures all the information that is captured in that release as well as AWR reports, SQL monitor reports and parameter information.</td>
</tr>
<tr>
<td></td>
<td>• WITH_RUNTIME_INFO: TCB captures runtime information for the SQL, such as dynamic sampling data, list of binds, Dynamic Plan info, along with information captured under BASIC mode.</td>
</tr>
<tr>
<td></td>
<td>Note this must be the same value as used in the IMPORT_SQL_TESTCASE Procedures.</td>
</tr>
<tr>
<td></td>
<td>• name=&quot;mexec_count&quot; - Value is any positive number (N). This parameter tells TCB to execute the statement for N time and capture runtime info at end of each execution.</td>
</tr>
<tr>
<td></td>
<td>• name=&quot;stat_history_since&quot; - Value is date. The object statistics history is exported using this parameter. Statistics history after date specified will be exported.</td>
</tr>
<tr>
<td>timelimit</td>
<td>How much time should we spend exporting the SQL test case</td>
</tr>
<tr>
<td>testcaseName</td>
<td>An optional name for the SQL test case. This is used to prefix all the generated scripts</td>
</tr>
<tr>
<td>testcase</td>
<td>Resulting testcase</td>
</tr>
<tr>
<td>preserveSchemaMapping</td>
<td>TRUE if the schema (or schemas) are not re-mapped from the original environment to the test environment</td>
</tr>
<tr>
<td>version</td>
<td>Version of database objects to be extracted. This option is only valid for EXPORT. Database objects or attributes incompatible with the version will not be extracted.</td>
</tr>
<tr>
<td></td>
<td>• COMPATIBLE - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the V$COMPATIBILITY view). Database compatibility must be set to 9.2 or higher.</td>
</tr>
<tr>
<td></td>
<td>• LATEST - the version of the metadata that specifies the current database version.</td>
</tr>
<tr>
<td></td>
<td>• A specific database version. For example, if '10.0.0', this cannot be lower than Oracle Database release 10.0.0.</td>
</tr>
</tbody>
</table>

**Usage Notes**

- A SQL test case generates a set of files needed to help reproduce a SQL failure on a different machine. It contains:
  - a dump file containing schemas objects and statistics (.dmp)
  - the explain plan for the statements (in advanced mode)
diagnostic information gathered on the offending statement
an import script to execute to reload the objects
a SQL script to replay system statistics of the source
a table of contents file describing the SQL test case
metadata. (xxxxxmain.xml)
a README.txt file that explain the usage of the TCB
the outlines used by the statement (ol.xml)
a list of parameters set in the exporting db/env (prmimp.sql)
a SQL monitor report, if any (smrpt.html)
an AWR report, if any (awrrpt.html)
a list of binds used in this statement (bndlst.xml)

• You should not run Test Case Builder (TCB) under user SYS. Instead, use another user who can be granted the DBA role.
• The default setting for TCB is that data is not exported. However, in some cases data is required, such as to diagnose an outcome with a result that is not optimal. To export data, call EXPORT_SQL_TESTCASE with exportData=>TRUE and the data will be imported by default, unless turned OFF by importData=>FALSE.
• TCB includes PL/SQL package spec by default, but not the PL/SQL package body. However, you may need to have the package body as well, for example, to invoke the PL/SQL functions, or because you have a Virtual Private Database (VPD) function defined in a package. To export a PL/SQL package body, call EXPORT_SQL_TESTCASE with exportPkgbody=>TRUE. To import a PL/SQL package body, call IMPORT_SQL_TESTCASE Procedures with importPkgbody=>TRUE.
• To export objects statistics history, the database compatibility should be set to 12.0 or higher.
• This procedure does not export data and statistics on a Global Temporary Table (GTT).

Examples
The user can specify multiple parameters in the ctrlOptions encapsulated either by using the <parameters> parent tag or without the parent tag.

Using the <parameters> tag

<parameters>
<parameter name="capture">with_runtime_info</parameter>
<parameter name="mexec_count">1</parameter>
</parameters>

Without the <parameters> tag

<parameter name="capture">with_runtime_info</parameter>
<parameter name="mexec_count">1</parameter>"
157.5.12 EXPORT_SQL_TESTCASE_DIR_BY_INC Function

This function generates a SQL test case corresponding to the incident ID passed as an argument. It creates a set of scripts and dump file in the directory passed as an argument.

Syntax

```sql
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_INC (
    incident_id        IN   NUMBER,
    directory          IN   VARCHAR2,
    exportEnvironment  IN   VARCHAR2 := 'TRUE',
    exportMetadata     IN   VARCHAR2 := 'TRUE',
    exportData         IN   VARCHAR2 := 'FALSE',
    samplingPercent    IN   VARCHAR2 := '100',
    ctrlOptions        IN   VARCHAR2 := NULL
    version            IN   VARCHAR2 := 'COMPATIBLE')
RETURN BOOLEAN;
```

Parameters

Table 157-21  EXPORT_SQL_TESTCASE_DIR_BY_INC Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incident_id</td>
<td>Incident ID containing the offending SQL. For more information about Incidents, see Oracle Database Performance Tuning Guide.</td>
</tr>
<tr>
<td>directory</td>
<td>Directory path to the generated files</td>
</tr>
<tr>
<td>exportEnvironment</td>
<td>TRUE if the compilation environment should be exported</td>
</tr>
<tr>
<td>exportMetadata</td>
<td>TRUE if the definition of the objects referenced in the SQL should be exported</td>
</tr>
<tr>
<td>exportData</td>
<td>TRUE if the data of the objects referenced in the SQL should be exported</td>
</tr>
<tr>
<td>samplingPercent</td>
<td>If is TRUE, specify the sampling percentage to use to create the dump file</td>
</tr>
<tr>
<td>ctrlOptions</td>
<td>Opaque control parameters. For example, to execute three times, set ctrlOptions with the following string: '&lt;parameter name=&quot;mexec_count&quot;&gt;3&lt;/parameter&gt;'.</td>
</tr>
<tr>
<td>version</td>
<td></td>
</tr>
</tbody>
</table>

- **capture** - BASIC (default) or WITH_RUNTIME_INFO: This parameter defines the mode of TCB capture.
  - **BASIC**: runs as Oracle release 11g TCB and captures all the information that is captured in that release as well as AWR reports, SQL monitor reports and parameter information.
  - **WITH_RUNTIME_INFO**: TCB captures runtime information for the SQL, such as dynamic sampling data, list of binds, Dynamic Plan info, along with information captured under **BASIC** mode.
- **name=mexec_count**: Value is any positive number (N). This parameter tells TCB to execute the statement for N time and capture runtime info at end of each execution.
- **name=stat_history_since**: Value is date. The object statistics history is exported using this parameter. Statistics history after date specified will be exported.
Table 157-21    (Cont.) EXPORT_SQL_TESTCASE_DIR_BY_INC Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version of database objects to be extracted. This option is only valid for EXPORT. Database objects or attributes incompatible with the version will not be extracted.</td>
</tr>
<tr>
<td></td>
<td>• COMPATIBLE - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the V$COMPATIBILITY view). Database compatibility must be set to 9.2 or higher.</td>
</tr>
<tr>
<td></td>
<td>• LATEST - the version of the metadata that specifies the current database version.</td>
</tr>
<tr>
<td></td>
<td>• A specific database version. For example, if '10.0.0', this cannot be lower than Oracle Database release 10.0.0.</td>
</tr>
</tbody>
</table>

157.5.13 EXPORT_SQL_TESTCASE_DIR_BY_TXT Function

This function generates a SQL Test Case corresponding to the SQL passed as an argument. It creates a set of scripts and dump files in the directory passed as an argument.

Syntax

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_TXT (incident_id IN NUMBER,
    directory IN VARCHAR2,
    sql_text IN CLOB,
    user_name IN VARCHAR2 := 'SYS',
    exportEnvironment IN VARCHAR2 := 'TRUE',
    exportMetadata IN VARCHAR2 := 'TRUE',
    exportData IN VARCHAR2 := 'FALSE',
    samplingPercent IN VARCHAR2 := '100',
    ctrlOptions IN VARCHAR2 := NULL
    version IN VARCHAR2 := 'COMPATIBLE')
RETURN BOOLEAN;
```

Parameters

Table 157-22    EXPORT_SQL_TESTCASE_DIR_BY_TXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incident_id</td>
<td>Incident ID containing the offending SQL</td>
</tr>
<tr>
<td>directory</td>
<td>Directory to store the various generated files</td>
</tr>
<tr>
<td>sql_text</td>
<td>Text of the SQL statement to explain</td>
</tr>
<tr>
<td>username</td>
<td>Name of the user schema to use to parse the SQL, defaults to SYS</td>
</tr>
<tr>
<td>exportEnvironment</td>
<td>TRUE if the compilation environment should be exported</td>
</tr>
<tr>
<td>exportMetadata</td>
<td>TRUE if the definition of the objects referenced in the SQL should be exported</td>
</tr>
<tr>
<td>exportData</td>
<td>TRUE if the data of the objects referenced in the SQL should be exported</td>
</tr>
</tbody>
</table>
### Table 157-22 (Cont.) EXPORT_SQL_TESTCASE_DIR_BY.TXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>samplingPercent</td>
<td>If is TRUE, specify the sampling percentage to use to create the dump file</td>
</tr>
<tr>
<td>ctrlOptions</td>
<td>Opaque control parameters. For example, to execute three times, set ctrlOptions with the following string: '&lt;parameter name=&quot;mexec_count&quot;&gt;3&lt;/parameter&gt;'.</td>
</tr>
<tr>
<td></td>
<td>• capture - BASIC (default) or WITH_RUNTIME_INFO. This parameter defines the mode of TCB capture.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>Version of database objects to be extracted. This option is only valid for EXPORT. Database objects or attributes incompatible with the version will not be extracted.</td>
</tr>
<tr>
<td></td>
<td>• COMPATIBLE - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the V$COMPATIBILITY view). Database compatibility must be set to 9.2 or higher.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 157.5.14 GET_FIX_CONTROL Function

This function returns the value of fix control for a given bug number.

**Syntax**

```sql
DBMS_SQLDIAG.GET_FIX_CONTROL ( bug_number IN NUMBER) RETURN NUMBER;
```
157.5.15 GET_SQL Function

This function loads a sql_setrow from the trace file associated to an the given incident ID.

Syntax

```
DBMS_SQLDIAG.GET_SQL (incident_id IN VARCHAR2)
RETURN SQLSET_ROW;
```

Parameters

Table 157-24  GET_SQL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incident_id</td>
<td>Identifier of the incident</td>
</tr>
</tbody>
</table>

157.5.16 IMPORT_SQL_TESTCASE Procedures

This procedure imports a SQL test case into a schema.

Syntax

This variant requires a source directory and SQL Testcase metadata object (in XML format).

```
DBMS_SQLDIAG.IMPORT_SQL_TESTCASE (directory IN VARCHAR2,
sqlTestCase IN CLOB,
importEnvironment IN BOOLEAN := TRUE,
importMetadata IN BOOLEAN := TRUE,
importData IN BOOLEAN := TRUE,
importPkgbody IN BOOLEAN := FALSE,
importDiagnosis IN BOOLEAN := TRUE,
ignoreStorage IN BOOLEAN := TRUE,
ctrlOptions IN VARCHAR2 := NULL,
preserveSchemaMapping IN BOOLEAN := FALSE);
```

This variant requires a source directory name of SQL Testcase metadata file.

```
DBMS_SQLDIAG.IMPORT_SQL_TESTCASE (directory IN VARCHAR2,
filename IN VARCHAR2,
importEnvironment IN BOOLEAN := TRUE,
importMetadata IN BOOLEAN := TRUE,
importData IN BOOLEAN := TRUE,
```
Parameters

Table 157-25 IMPORT_SQL_TESTCASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Directory containing test case files</td>
</tr>
<tr>
<td>filename</td>
<td>Name of a file containing an XML document describing the SQL test case</td>
</tr>
<tr>
<td>importEnvironment</td>
<td>TRUE if the compilation environment should be imported</td>
</tr>
<tr>
<td>importMetadata</td>
<td>TRUE if the definition of the objects referenced in the SQL should be imported</td>
</tr>
<tr>
<td>importData</td>
<td>TRUE if the data of the objects referenced in the SQL should be imported</td>
</tr>
<tr>
<td>importPkgbody</td>
<td>TRUE if the body of the packages referenced in the SQL are imported</td>
</tr>
<tr>
<td>importDiagnosis</td>
<td>TRUE if the diagnostic information associated to the task should be imported</td>
</tr>
<tr>
<td>ignoreStorage</td>
<td>TRUE if the storage attributes should be ignored</td>
</tr>
<tr>
<td>ctrlOptions</td>
<td>Opaque control parameters, of which only capture is valid for this subprogram.</td>
</tr>
<tr>
<td>preserveSchemaMapping</td>
<td>TRUE if the schema (or schemas) are not re-mapped from the original environment to the test environment (schema mapping in the target database will be identical to the source database). Note that when an import is run with preserveSchemaMapping set to TRUE, if the objects in the schemas exist then the import will overwrite the existing objects.</td>
</tr>
</tbody>
</table>

Usage Notes

- A SQL test case generates a set of files needed to help reproduce a SQL failure on a different machine. It contains:
  - a dump file containing schemas objects and statistics (.dmp)
  - the explain plan for the statements (in advanced mode)
  - diagnostic information gathered on the offending statement
  - an import script to execute to reload the objects
  - a SQL script to replay system statistics of the source
a table of contents file describing the SQL test case
- metadata. (xxxxmain.xml)
- a README.txt file that explain the usage of the TCB
- the outlines used by the statement (ol.xml)
- a list of parameters set in the exporting db/env (prmimp.sql)
- a SQL monitor report, if any (smrpt.html)
- an AWR report, if any (awrrpt.html)
- a list of binds used in this statement (bndlst.xml)

- You should not run Test Case Builder (TCB) under user SYS. Instead, use another user who can be granted the SYSDBA privilege
- The default setting for TCB is that data is not exported. However, in some cases data is required, such as to diagnose an outcome with a result that is not optimal. To export data, call EXPORT_SQL_TESTCASE Procedures with exportData=>TRUE and the data will be imported by default, unless turned OFF by importData=>FALSE.
- TCB includes PL/SQL package spec by default, but not the PL/SQL package body. However, you may need to have the package body as well, for example, to invoke the PL/SQL functions, or because you have a Virtual Private Database (VPD) function defined in a package. To export a PL/SQL package body, call EXPORT_SQL_TESTCASE Procedures with exportPkgbody=>TRUE. To import a PL/SQL package body, call IMPORT_SQL_TESTCASE Procedures with importPkgbody=>TRUE.
- The capture value used when invoking the EXPORT_SQL_TESTCASE Procedures must be used when calling this procedure.

157.5.17 INCIDENTID_2_SQL Procedure

This procedure initializes a sql_setrow from an incident ID.

Syntax

```sql
DBMS_SQLDIAG.INCIDENTID_2_SQL (incident_id   IN     VARCHAR2,
sql_stmt      OUT    SQLSET_ROW,
problem_type  OUT    NUMBER,
err_code      OUT    BINARY_INTEGER,
err_mesg      OUT    VARCHAR2);
```

Parameters

Table 157-26 INCIDENTID_2_SQL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incident_id</td>
<td>Identifier of the incident</td>
</tr>
<tr>
<td>sql_stmt</td>
<td>Resulting SQL</td>
</tr>
</tbody>
</table>
Table 157-26  (Cont.) INCIDENTID_2_SQL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem_type</td>
<td>Tentative type of SQL problem (currently among PROBLEM_TYPE_COMPILATION_ERROR and PROBLEM_TYPE_EXECUTION_ERROR)</td>
</tr>
<tr>
<td>err_code</td>
<td>Error code if any otherwise it is set to NULL</td>
</tr>
<tr>
<td>err_msg</td>
<td>Error message if any otherwise it is set to NULL</td>
</tr>
</tbody>
</table>

157.5.18 INTERRUPT_DIAGNOSIS_TASK Procedure

This procedure interrupts a diagnostic task.

Syntax

```sql
DBMS_SQLDIAG.INTERRUPT_DIAGNOSIS_TASK (
    taskname        IN   VARCHAR2);
```

Parameters

Table 157-27  INTERRUPT_DIAGNOSIS_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Name of task</td>
</tr>
</tbody>
</table>

157.5.19 LOAD_SQLSET_FROM_TCB Function

This function loads a SQLSET from a Test Case Builder file.

Syntax

```sql
DBMS_SQLDIAG.LOAD_SQLSET_FROM_TCB (
    directory        IN     VARCHAR2,
    filename         IN     VARCHAR2,
    sqlset_name      IN     VARCHAR2 DEFAULT NULL) RETURN VARCHAR2;
```

Parameters

Table 157-28  LOAD_SQLSET_FROM_TCB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Name of directory</td>
</tr>
<tr>
<td>filename</td>
<td>Name of file</td>
</tr>
<tr>
<td>sqlset_name</td>
<td>Name of SQLSET</td>
</tr>
</tbody>
</table>
157.5.20 PACK_STGTAB_SQLPATCH Procedure

This procedure packs SQL patches into the staging table created by a call to the CREATE_STGTAB_SQLPATCH Procedure.

Syntax

```
DBMS_SQLDIAG.UPPACK_STGTAB_SQLPATCH (  
  patch_name            IN  VARCHAR2 := '%',  
  patch_category        IN  VARCHAR2 := 'DEFAULT',  
  staging_table_name    IN  VARCHAR2,  
  staging_schema_owner  IN  VARCHAR2 := NULL);
```

Parameters

Table 157-29  UPPACK_STGTAB_SQLPATCH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patch_name</td>
<td>Name of patch to pack (% wildcards acceptable, case-sensitive)</td>
</tr>
<tr>
<td>patch_category</td>
<td>Category to which to pack patches (% wildcards acceptable, case-insensitive)</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>(Mandatory) Name of the table to use (case-sensitive)</td>
</tr>
<tr>
<td>staging_schema_owner</td>
<td>Schema where the table resides, or NULL for current schema (case-sensitive)</td>
</tr>
</tbody>
</table>

Usage Notes

- Requires: ADMINISTER SQL PLAN MANAGEMENT OBJECT privilege and INSERT privilege on the staging table.

- By default, we move all SQL patches in category DEFAULT. See the Examples for details. Note that the subprogram issues a COMMIT after packing each SQL patch, so if an error is raised in mid-execution, some patches may be in the staging table.

Related Topics

- CREATE_STGTAB_SQLPATCH Procedure
  This procedure creates the staging table used for transporting SQL patches from one system to another.

157.5.21 REPLAY_SQL_TESTCASE Function

This function automates the reproduction of the SQL Test Case.

Syntax

```
DBMS_SQLDIAG.REPLAY_SQL_TESTCASE (  
  directory       IN   VARCHAR2,  
  filename        IN   VARCHAR2,  
  ctrlOptions     IN   VARCHAR2 := NULL,  
  format          IN   VARCHAR2 := 'TEXT')  
RETURN CLOB;
```

```
DBMS_SQLDIAG.REPLAY_SQL_TESTCASE (  
  directory       IN   VARCHAR2,  
```

ORACLE
Parameters

Table 157-30  REPLAY_SQL_TESTCASE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Directory containing test case files</td>
</tr>
<tr>
<td>filename</td>
<td>Name of a file containing an XML document describing the SQL test case</td>
</tr>
<tr>
<td>ctrlOptions</td>
<td>Opaque control parameters. For example, to execute three times, set ctrlOptions with the following string: '&lt;parameter name=&quot;mexec_count&quot;&gt;3&lt;/parameter&gt;'.</td>
</tr>
<tr>
<td></td>
<td>· replay - EXPLAIN (default), OUTLINE, EXECUTION or OUTLINE EXECUTION. This parameter defines TCB replay functionality.</td>
</tr>
<tr>
<td></td>
<td>· replay - EXPLAIN (default), OUTLINE, EXECUTION or OUTLINE EXECUTION. This parameter defines TCB replay functionality.</td>
</tr>
<tr>
<td></td>
<td>· replay - OUTLINE: Replay uses outlines mode and explains the statement using outlines</td>
</tr>
<tr>
<td></td>
<td>· replay - EXECUTION: Replay executes the statement without using outlines</td>
</tr>
<tr>
<td></td>
<td>· replay - OUTLINE EXECUTION: Replay executes the statement using outlines</td>
</tr>
<tr>
<td></td>
<td>Note that if the user gives an incorrect parameter value, then the replay runs in default mode and no error is thrown.</td>
</tr>
<tr>
<td></td>
<td>· name=mexec_count - Value is any positive number (N). This parameter tells TCB to execute the statement for N time and capture runtime info at end of each execution.</td>
</tr>
<tr>
<td>sqlTestCase</td>
<td>SQL test case</td>
</tr>
<tr>
<td>format</td>
<td>Format of the replay report. Possible formats are: TEXT, XML and HTML.</td>
</tr>
</tbody>
</table>

Examples

TCB Replay Mode: Execute

```
SELECT /* tcbdynpl_1 */ /*+ gather_plan_statistics */ * FROM (SELECT * FROM emp where emp.sal > 100) emp, dept WHERE emp.deptno = dept.deptno And emp.sal > 1000 /*
tcbdynpl_1 */
```

Explain Plan

```
Plan Hash Value : 2219294842
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>HASH JOIN</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NESTED LOOPS</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>STATISTICS COLLECTOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>DEPT</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>INDEX RANGE SCAN</td>
<td>EMP_IDX_DEPTNO</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>EMP</td>
<td>3</td>
</tr>
</tbody>
</table>
157.5.22 REPORT_DIAGNOSIS_TASK Function

This function reports on a diagnostic task. It returns a CLOB containing the desired report.

Syntax

```sql
DBMS_SQLDIAG.REPORT_DIAGNOSIS_TASK (  
taskname IN VARCHAR2,
type IN VARCHAR2 := TYPE_TEXT,  
level IN VARCHAR2 := LEVEL_TYPICAL,  
section IN VARCHAR2 := SECTION_ALL,  
object_id IN NUMBER := NULL,  
result_limit IN NUMBER := NULL,  
owner_name IN VARCHAR2 := NULL)  
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Name of task to report</td>
</tr>
<tr>
<td>type</td>
<td>Type of the report. Possible values are: TEXT, HTML, XML (see Table 157-4).</td>
</tr>
</tbody>
</table>
### Table 157-31 (Cont.) REPORT_DIAGNOSIS_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>Format of the recommendations. Possible values are TYPICAL, BASIC, ALL (Table 157-5).</td>
</tr>
<tr>
<td>section</td>
<td>Particular section in the report. Possible values are: SUMMARY, FINDINGS, PLAN, INFORMATION, ERROR, ALL (Table 157-6).</td>
</tr>
<tr>
<td>object_id</td>
<td>Identifier of the advisor framework object that represents a given statement in a SQL Tuning Set (STS).</td>
</tr>
<tr>
<td>result_limit</td>
<td>Number of statements in a STS for which the report is generated</td>
</tr>
<tr>
<td>owner_name</td>
<td>Name of the task execution to use. If NULL, the report will be generated for the last task execution.</td>
</tr>
</tbody>
</table>

#### 157.5.23 RESET_DIAGNOSIS_TASK Procedure

This procedure resets a diagnostic task.

**Syntax**

```sql
DBMS_SQLDIAG.RESET_DIAGNOSIS_TASK (taskname IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Name of task</td>
</tr>
</tbody>
</table>

#### 157.5.24 RESUME_DIAGNOSIS_TASK Procedure

This procedure resumes a diagnostic path.

**Syntax**

```sql
DBMS_SQLDIAG.RESUME_DIAGNOSIS_TASK (taskname IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Name of task</td>
</tr>
</tbody>
</table>
157.5.25 SET_DIAGNOSIS_TASK_PARAMETER Procedure

This procedure is called to update the value of a SQL diagnosis parameter of type VARCHAR2.

The task must be set to its initial state before calling this procedure. The diagnosis parameters that can be set by this procedure are:

- **MODE**: diag scope (comprehensive, limited)
- **_SQLDIAG_FINDING_MODE**: findings in the report (see "Table 157-8" for possible values)

Syntax

```sql
DBMS_SQLDIAG.SET_DIAGNOSIS_TASK_PARAMETER (  
    taskname           IN   VARCHAR2,  
    parameter          IN   VARCHAR2,    value              IN   NUMBER);
```

Parameters

Table 157-34  SET_DIAGNOSIS_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskname</td>
<td>Identifier of the task to execute</td>
</tr>
<tr>
<td>parameter</td>
<td>Name of the parameter to set</td>
</tr>
<tr>
<td>value</td>
<td>New value of the specified parameter</td>
</tr>
</tbody>
</table>

157.5.26 UNPACK_STGTAB_SQLPATCH Procedure

This procedure unpacks from the staging table populated by a call to the PACK_STGTAB_SQLPATCH Procedure. It uses the patch data stored in the staging table to create patches on this system. Users can opt to replace existing patches with patch data when they exist already. In this case, note that it is only possible to replace patches referring to the same statement if the names are the same (see the ACCEPT_SQL_PATCH Function & Procedure).

Syntax

```sql
DBMS_SQLDIAG.UNPACK_STGTAB_SQLPATCH (  
    patch_name            IN  VARCHAR2 := '%',  
    patch_category        IN  VARCHAR2 := '%',  
    replace               IN  BOOLEAN,  
    staging_table_name    IN  VARCHAR2,  
    staging_schema_owner  IN  VARCHAR2 := NULL);
```
Parameters

Table 157-35  UPPACK_STGTAB_SQLPATCH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patch_name</td>
<td>Name of patch to unpack (% wildcards acceptable, case-sensitive)</td>
</tr>
<tr>
<td>patch_category</td>
<td>Category from which to unpack patches (% wildcards acceptable, case-insensitive)</td>
</tr>
<tr>
<td>replace</td>
<td>Replace patches if they already exist. Note that patches cannot be replaced if there is one in the staging table with the same name as an active patch on different SQL. The subprogram raises an error if there an attempt to create a patch that already exists.</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>(Mandatory) Name of the table to use (case-sensitive)</td>
</tr>
<tr>
<td>staging_schema_owner</td>
<td>Schema where the table resides, or NULL for current schema (case-sensitive)</td>
</tr>
</tbody>
</table>

Usage Notes

- Requires: ADMINISTER SQL MANAGEMENT OBJECT privilege and SELECT or READ privilege on the staging table
- By default, all SQL patches in the staging table are moved. The function commits after successfully loading each patch. If it fails in creating an individual patch, it raises an error and does not proceed to those remaining in the staging table.

Related Topics

- **PACK_STGTAB_SQLPATCH Procedure**
  This procedure packs SQL patches into the staging table created by a call to the CREATE_STGTAB_SQLPATCH Procedure.
- **ACCEPT_SQL_PATCH Function & Procedure**
  This procedure accepts a recommended SQL patch as recommended by the specified SQL diagnosis task.
DBMS_SQLPA

The DBMS_SQLPA package provides the interface to implement the SQL Performance Analyzer.

The chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_SQLPA Subprograms

158.1 DBMS_SQLPA Overview

The DBMS_SQLPA package provides a capacity to help users predict the impact of system environment changes on the performance of a SQL workload. The interface lets users build and then compare two different versions of the workload performance, analyze the differences between the two versions, and unmask the SQL statements that might be impacted by the changes.

The package provides a task-oriented interface to implement the SQL Performance Analyzer. For example:

1. You use the CREATE_ANALYSIS_TASK Functions to create an analysis task for a single statement or a group of SQL statements.
2. The EXECUTE_ANALYSIS_TASK Function & Procedure executes a previously created analysis task.
3. The REPORT_ANALYSIS_TASK Function displays the results of an analysis task.

158.2 DBMS_SQLPA Security Model

This package is available to PUBLIC and performs its own security checking. All analysis task interfaces (XXX_ANALYSIS_TASK) require privilege ADVISOR.

158.3 Summary of DBMS_SQLPA Subprograms

This table lists the DBMS_SQLPA subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCEL_ANALYSIS_TASK Procedure</td>
<td>Cancels the currently executing task analysis of one or more SQL statements</td>
</tr>
<tr>
<td>CREATE_ANALYSIS_TASK Functions</td>
<td>Creates an advisor task to process and analyze one or more SQL statements</td>
</tr>
<tr>
<td>DROP_ANALYSIS_TASK Procedure</td>
<td>Drops a SQL analysis task</td>
</tr>
</tbody>
</table>
### Table 158-1  (Cont.) DBMS_SQLPA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE_ANALYSIS_TASK Function &amp; Procedure</td>
<td>Executes a previously created analysis task</td>
</tr>
<tr>
<td>INTERRUPT_ANALYSIS_TASK Procedure</td>
<td>Interrupts the currently executing analysis task</td>
</tr>
<tr>
<td>REPORT_ANALYSIS_TASK Function</td>
<td>Displays the results of an analysis task</td>
</tr>
<tr>
<td>RESET_ANALYSIS_TASK Procedure</td>
<td>Resets the currently executing analysis task to its initial state</td>
</tr>
<tr>
<td>RESUME_ANALYSIS_TASK Procedure</td>
<td>Resumes a previously interrupted analysis task that was created to process a SQL tuning set.</td>
</tr>
<tr>
<td>SET_ANALYSIS_TASK_PARAMETER Procedures</td>
<td>Sets the SQL analysis task parameter value</td>
</tr>
<tr>
<td>SET_ANALYSIS_DEFAULT_PARAMETER Procedures</td>
<td>Sets the SQL analysis task parameter default value</td>
</tr>
</tbody>
</table>

#### 158.3.1 CANCEL_ANALYSIS_TASK Procedure

This procedure cancels the currently executing analysis task. All intermediate result data is removed from the task.

**Syntax**

```sql
dbm_sqlpa.cancel_analysis_task(
    task_name  IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task to cancel</td>
</tr>
</tbody>
</table>

**Examples**

Canceling a task when there is a need to stop it executing and it is not required to view any already-completed results:

```sql
exec dbms_sqlpa.cancel_analysis_task(:my_task);
```

#### 158.3.2 CREATE_ANALYSIS_TASK Functions

These functions create an advisor task to process and analyze one or more SQL statements.

You can use different forms of this function to:

- Create an analysis task for a single statement given its text.
- Create an analysis task for a single statement from the cursor cache given its identifier.
• Create an analysis task for a single statement from the workload repository given a range of snapshot identifiers.

• Create an analysis task for a SQL tuning set.

In all cases, the function creates an advisor task and sets its parameters.

Syntax

SQL text format. This form of the function is called to prepare the analysis of a single statement given its text.

```sql
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    sql_text       IN CLOB,
    bind_list      IN sql_binds := NULL,
    parsing_schema IN VARCHAR2 := NULL,
    task_name      IN VARCHAR2 := NULL,
    description    IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```

SQL ID format. This form of the function is called to prepare the analysis of a single statement from the cursor cache given its identifier.

```sql
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    sql_id         IN VARCHAR2,
    plan_hash_value IN NUMBER := NULL,
    task_name      IN VARCHAR2 := NULL,
    con_name       IN VARCHAR2 DEFAULT,
    description    IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```

Workload Repository format. This form of the function is called to prepare the analysis of a single statement from the workload repository given a range of snapshot identifiers.

```sql
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    dbid           IN NUMBER DEFAULT,
    begin_snap     IN NUMBER,
    end_snap       IN NUMBER,
    sql_id         IN VARCHAR2,
    plan_hash_value IN NUMBER := NULL,
    task_name      IN VARCHAR2 := NULL,
    con_name       IN VARCHAR2 DEFAULT,
    description    IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```

SQLSET format. This form of the function is called to prepare the analysis of a SQL tuning set.

```sql
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    sqlset_name    IN VARCHAR2,
    basic_filter   IN VARCHAR2 := NULL,
    con_name       IN VARCHAR2 DEFAULT,
    order_by       IN VARCHAR2 := NULL,
    top_sql        IN VARCHAR2 := NULL,
    task_name      IN VARCHAR2 := NULL,
    description    IN VARCHAR2 := NULL,
    sqlset_owner   IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>Text of a SQL statement</td>
</tr>
<tr>
<td>bind_list</td>
<td>A set of bind values</td>
</tr>
<tr>
<td>parsing_schema</td>
<td>Name of the schema where the statement can be compiled</td>
</tr>
<tr>
<td>task_name</td>
<td>Optional analysis task name</td>
</tr>
<tr>
<td>dbid</td>
<td>The DBID for imported or PDB-level AWR data. If NULL, then the current database DBID is used</td>
</tr>
<tr>
<td>con_name</td>
<td>Container for the SPA task. The semantics depend on the function format:</td>
</tr>
<tr>
<td></td>
<td>For the SQL ID format, this parameter specifies the container from which the database fetches the SQL statement for using with SPA. SPA will analyze the statement in this container. If null, then the database uses the current PDB for SPA analysis.</td>
</tr>
<tr>
<td></td>
<td>For the AWR format, this parameter specifies the container from whose AWR data the database fetches the SQL statement for using with SPA. SPA will analyze the statement in this container. If null, then the database uses the current PDB for SPA analysis.</td>
</tr>
<tr>
<td></td>
<td>The following statements are true of all function formats:</td>
</tr>
<tr>
<td></td>
<td>· In a non-CDB, this parameter is ignored.</td>
</tr>
<tr>
<td></td>
<td>· In a PDB, this parameter must be null or match the container name of the PDB. Otherwise, error occurs.</td>
</tr>
<tr>
<td></td>
<td>· In a CDB root, this parameter must be null or match the container name of a container in this CDB. Otherwise, error occurs.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the SQL analysis task to a maximum of 256 characters</td>
</tr>
<tr>
<td>sql_id</td>
<td>Identifier of a SQL statement</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Hash value of the SQL execution plan</td>
</tr>
<tr>
<td>begin_snap</td>
<td>Begin snapshot identifier</td>
</tr>
<tr>
<td>end_snap</td>
<td>End snapshot identifier</td>
</tr>
<tr>
<td>sqlset_name</td>
<td>SQL tuning set name</td>
</tr>
<tr>
<td>basic_filter</td>
<td>SQL predicate to filter the SQL from the SQL tuning set</td>
</tr>
<tr>
<td>order_by</td>
<td>Order-by clause on the selected SQL</td>
</tr>
<tr>
<td>top_sql</td>
<td>Top N SQL after filtering and ranking</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>The owner of the SQL tuning set, or NULL for the current schema owner</td>
</tr>
</tbody>
</table>

### Return Values

A SQL analysis task name that is unique by user (two different users can give the same name to their advisor tasks).
Examples

```sql
variable stmt_task VARCHAR2(64);
variable sts_task  VARCHAR2(64);

-- Sql text format
EXEC :stmt_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    sql_text => 'select quantity_sold from sales s, times t where s.time_id =
    t.time_id and s.time_id = TO_DATE(''24-NOV-00'')');

-- Sql id format (cursor cache)
EXEC :stmt_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    sql_id       => 'ay1m3ssvtrh24');

-- Workload repository format
exec :stmt_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    begin_snap  => 1,
    end_snap    => 2,
    sql_id      => 'ay1m3ssvtrh24');

-- Sql tuning set format (first we need to load an STS, then analyze it)
EXEC :sts_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK( -
    sqlset_name    =>   'my_workload', -
    order_by       =>   'BUFFER_GETS', -
    description    =>   'process workload ordered by buffer gets');
```

### 158.3.3 DROP_ANALYSIS_TASK Procedure

This procedure drops a SQL analysis task. The task and all its result data are deleted.

**Syntax**

```sql
DBMS_SQLPA.DROP_ANALYSIS_TASK(
    task_name         IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the analysis task to drop</td>
</tr>
</tbody>
</table>

### 158.3.4 EXECUTE_ANALYSIS_TASK Function & Procedure

This function and procedure executes a previously created analysis task, the function version returning the new execution name.

**Syntax**

```sql
DBMS_SQLPA.EXECUTE_ANALYSIS_TASK( -
    task_name         IN VARCHAR2, -
    execution_type    IN VARCHAR2               := 'test execute', -
    execution_name    IN VARCHAR2               := NULL, -
    execution_params  IN dbms_advisor.argList   := NULL, -
    execution_desc    IN VARCHAR2               := NULL)
```

RETURN VARCHAR2;
DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(
  task_name         IN VARCHAR2,
  execution_type    IN VARCHAR2               := 'test execute',
  execution_name    IN VARCHAR2               := NULL,
  execution_params  IN dbms_advisor.argList   := NULL,
  execution_desc    IN VARCHAR2               := NULL);

Parameters

Table 158-5  EXECUTE_ANALYSIS_TASK Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of the task to execute</td>
</tr>
<tr>
<td>execution_type</td>
<td>Type of the action to perform by the function. If NULL it will default to the value of the DEFAULT_EXECUTION_TYPE parameter. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- [TEST] EXECUTE - test-execute every SQL statement and collect its execution plans and execution statistics. The resulting plans and statistics will be stored in the advisor framework. This is default.</td>
</tr>
<tr>
<td></td>
<td>- EXPLAIN PLAN - generate explain plan for every statement in the SQL workload. This is similar to the EXPLAIN PLAN command. The resulting plans will be stored in the advisor framework in association with the task.</td>
</tr>
<tr>
<td></td>
<td>- COMPARE [PERFORMANCE] - analyze and compare two versions of SQL performance data. The performance data is generated by test-executing or generating explain plan of the SQL statements. Use this option when two executions of type EXPLAIN_PLAN or TEST_EXECUTE already exist in the task</td>
</tr>
<tr>
<td></td>
<td>- CONVERT SQLSET - used to read the statistics captured in a SQL Tuning Set and model them as a task execution. This can be used when you wish to avoid executing the SQL statements because valid data for the experiment already exists in the SQL Tuning Set.</td>
</tr>
<tr>
<td>execution_name</td>
<td>A name to qualify and identify an execution. If not specified, it will be generated by the advisor and returned by function.</td>
</tr>
<tr>
<td>execution_params</td>
<td>List of parameters (name, value) for the specified execution. The execution parameters have effect only on the execution for which they are specified. They will override the values for the parameters stored in the task (set through the SET_ANALYSIS_DEFAULT_PARAMETER Procedures).</td>
</tr>
<tr>
<td>execution_desc</td>
<td>A 256-length string describing the execution</td>
</tr>
</tbody>
</table>

Usage Notes

SQL performance analyzer task can be executed multiples times without having to reset it. For example, when a task is created to perform a change impact analysis on a SQL workload, the created task has to be executed before making any change in the system environment to build a version of the workload that will be used as a reference for performance analysis. Once the change has been made, a second execution is required to build the post-change version of the workload. Finally, the task has to be executed a third time to let the advisor analyze and compare the performance of the workload in both versions.
Examples

1. Create a task with a purpose of change impact analysis

EXEC :tname := DBMS_SQLPA.CREATE_ANALYSIS_TASK(  
    sqlset_name => 'my_sts');

2. Make baseline or the before change execution

EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(  
    task_name => :tname,  
    execution_type => 'test execute',  
    execution_name => 'before_change');

3. Make change

...

4. Make the after change version of the workload performance

EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(  
    task_name => :tname,  
    execution_type => 'test execute',  
    execution_name => 'after_change');

5. Compare the two versions of the workload

By default we always compare the results of the two last executions. The SQL Performance Analyzer uses the elapsed_time as a default metric for comparison. Here we are changing it to buffer_gets instead.

EXEC DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(  
    :tname, 'comparison_metric', 'buffer_gets');
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(  
    task_name => :tname,  
    execution_type => 'compare performance',  
    execution_name => 'after_change');

Use the following call if you would like to explicitly specify the two executions to compare as well as the comparison metric to use.

EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(  
    task_name => :tname,  
    execution_type => 'compare performance',  
    execution_params => dbms_advisor.arglist(  
        'execution_name1',  
        'before_change',  
        'execution_name2',  
        'after_change',  
        'comparison_metric',  
        'buffer_gets'));
158.3.5  INTERRUPT_ANALYSIS_TASK  Procedure

This procedure interrupts the currently executing analysis task. All intermediate result data will not be removed from the task.

Syntax

DBMS_SQLPA.INTERRUPT_ANALYSIS_TASK(
    task_name         IN VARCHAR2);

Parameters

Table 158-6  INTERRUPT_ANALYSIS_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of the analysis task to interrupt</td>
</tr>
</tbody>
</table>

Examples

EXEC DBMS_SQLPA.INTERRUPT_ANALYSIS_TASK(:my_task);

158.3.6  REPORT_ANALYSIS_TASK  Function

This procedure displays the results of an analysis task.

Syntax

DBMS_SQLPA.REPORT_ANALYSIS_TASK(
    task_name       IN   VARCHAR2,
    type            IN   VARCHAR2   := 'TEXT',
    level           IN   VARCHAR2   := 'TYPICAL',
    section         IN   VARCHAR2   := 'SUMMARY',
    object_id       IN   NUMBER     := NULL,
    top_sql         IN   NUMBER     := 100,
    execution_name  IN   VARCHAR2   := NULL,
    task_owner      IN   VARCHAR2   := NULL,
    order_by        IN   VARCHAR2   := NULL)
RETURN CLOB;

Parameters

Table 158-7  REPORT_ANALYSIS_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task to report</td>
</tr>
<tr>
<td>type</td>
<td>Type of the report to produce. Possible values are TEXT (default), HTML, XML and ACTIVE (see Usage Notes).</td>
</tr>
</tbody>
</table>
### Table 158-7  (Cont.) REPORT_ANALYSIS_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>Level of detail in the report:</td>
</tr>
<tr>
<td></td>
<td>- ALL - details of all SQL</td>
</tr>
<tr>
<td></td>
<td>- BASIC - currently the same as typical</td>
</tr>
<tr>
<td></td>
<td>- CHANGED - only SQL with changed performance</td>
</tr>
<tr>
<td></td>
<td>- CHANGED_PLANS - only SQL with plan changes</td>
</tr>
<tr>
<td></td>
<td>- ERRORS - SQL with errors only</td>
</tr>
<tr>
<td></td>
<td>- IMPROVED - only improved SQL</td>
</tr>
<tr>
<td></td>
<td>- REGRESSED - only regressed SQL</td>
</tr>
<tr>
<td></td>
<td>- TIMEOUT - only SQL which timed-out during execution</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL (default) - show information about every statement analyzed,</td>
</tr>
<tr>
<td></td>
<td>including changing and errors</td>
</tr>
<tr>
<td></td>
<td>- UNCHANGED - only SQL with unchanged performance</td>
</tr>
<tr>
<td></td>
<td>- UNCHANGED_PLANS - only SQL with unchanged plans</td>
</tr>
<tr>
<td></td>
<td>- UNSUPPORTED - only SQL not supported by SPAs</td>
</tr>
<tr>
<td>section</td>
<td>Optionally limit the report to a single section (ALL for all sections):</td>
</tr>
<tr>
<td></td>
<td>- SUMMARY (default) - workload summary only</td>
</tr>
<tr>
<td></td>
<td>- ALL - summary and details on SQL</td>
</tr>
<tr>
<td>object_id</td>
<td>Identifier of the advisor framework object that represents a given SQL in</td>
</tr>
<tr>
<td></td>
<td>a tuning set (STS)</td>
</tr>
<tr>
<td>top_sql</td>
<td>Number of SQL statements in a STS for which the report is generated</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name of the task execution to use. If NULL, the report will be generated</td>
</tr>
<tr>
<td></td>
<td>for the last task execution.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Owner of the relevant analysis task. Defaults to the current schema owner.</td>
</tr>
<tr>
<td>order_by</td>
<td>How to sort SQL statements in the report (summary and body). Possible</td>
</tr>
<tr>
<td></td>
<td>values:</td>
</tr>
<tr>
<td></td>
<td>- CHANGE_DIFF - sort SQL statements by change difference in SQL</td>
</tr>
<tr>
<td></td>
<td>performance in terms of the comparison Metric</td>
</tr>
<tr>
<td></td>
<td>- NULL (default) - order SQL statement by impact on workload</td>
</tr>
<tr>
<td></td>
<td>- SQL_IMPACT - order SQL statement by change impact on SQL</td>
</tr>
<tr>
<td></td>
<td>- WORKLOAD_IMPACT - same as NULL</td>
</tr>
<tr>
<td></td>
<td>- METRIC_DELTA - same as CHANGE_DIFF</td>
</tr>
</tbody>
</table>

### Return Values

A CLOB containing the desired report.

### Usage Notes

**ACTIVE** reports have a rich, interactive user interface similar to Enterprise Manager while not requiring any EM installation. The report file built is in HTML format so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.
Examples

-- Get the whole report for the single statement case.
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(:stmt_task) from dual;

-- Show me the summary for the sts case.
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(:sts_task, 'TEXT', 'TYPICAL', 'SUMMARY') FROM DUAL;

-- Show me the findings for the statement I'm interested in.
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(:sts_task, 'TEXT', 'TYPICAL', 'ALL', 5) from dual;

158.3.7 RESET_ANALYSIS_TASK Procedure

This procedure is called on an analysis task that is not currently executing to prepare it for re-execution.

All intermediate result data will be deleted.

Syntax

DBMS_SQLPA.RESET_ANALYSIS_TASK(
  task_name IN VARCHAR2);

Parameters

Table 158-8  RESET_ANALYSIS_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of the analysis task to reset</td>
</tr>
</tbody>
</table>

Examples

-- reset and re-execute a task
EXEC DBMS_SQLPA.RESET_ANALYSIS_TASK(:sts_task);

-- re-execute the task
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(:sts_task);

158.3.8 RESUME_ANALYSIS_TASK Procedure

This procedure resumes a previously interrupted or FAILED (with a fatal error) task execution.

Syntax

DBMS_SQLPA.RESUME_ANALYSIS_TASK(
  task_name IN VARCHAR2,
  basic_filter IN VARCHAR2 := NULL);
Parameters

Table 158-9  RESUME_ANALYSIS_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of the analysis task to resume</td>
</tr>
<tr>
<td>basic_filter</td>
<td>A SQL predicate to filter the SQL from the SQL tuning set. Note that this filter will be applied in conjunction with the basic filter (parameter basic_filter) that was specified when calling the CREATE_ANALYSIS_TASK Functions.</td>
</tr>
</tbody>
</table>

Usage Notes

Resuming a single SQL analysis task (a task that was created to analyze a single SQL statement as compared to a SQL Tuning Set) is not supported.

Examples

-- Interrupt the task
EXEC DBMS_SQLPA.INTERRUPT_ANALYSIS_TASK(:conc_task);

-- Once a task is interrupted, we can elect to reset it, resume it, or check -- out its results and then decide. For this example we will just resume.
EXEC DBMS_SQLPA.RESUME_ANALYSIS_TASK(:conc_task);

158.3.9 SET_ANALYSIS_TASK_PARAMETER Procedures

This procedure sets the SQL analysis task parameter value.

Syntax

This form of the procedure updates the value of a SQL analysis parameter of type VARCHAR2.

DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(
    task_name         IN VARCHAR2,
    parameter         IN VARCHAR2,
    value             IN VARCHAR2,
    test_execute_dop  IN NUMBER DEFAULT 0,
    compare_resultset IN BOOLEAN DEFAULT TRUE);

This form of the procedure updates the value of a SQL analysis parameter of type NUMBER.

DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(
    task_name         IN VARCHAR2,
    parameter         IN VARCHAR2,
    value             IN NUMBER
    test_execute_dop  IN NUMBER DEFAULT 0,
    compare_resultset IN BOOLEAN DEFAULT TRUE);
Parameters

Table 158-10  SET_ANALYSIS_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of the task to execute</td>
</tr>
<tr>
<td>parameter</td>
<td>Name of the parameter to set. The possible analysis parameters that can be set by this procedure are:</td>
</tr>
<tr>
<td></td>
<td>• APPLY_CAPTURED_COMPILENV: indicates whether the advisor could use the compilation environment captured with the SQL statements. The default is 0 (that is, NO).</td>
</tr>
<tr>
<td></td>
<td>• BASIC_FILTER: basic filter for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>• CELL_SIMULATION_ENABLED: for more details, see the helper script tcellsim.sql in the ADMIN directory.</td>
</tr>
<tr>
<td></td>
<td>• COMPARISON_METRIC: specify an expression of execution statistics to use in performance comparison (Example: buffer_gets, cpu_time + buffer_gets * 10)</td>
</tr>
<tr>
<td></td>
<td>• DATABASE_LINK: can be set to the global name of a PUBLIC database link. When it is set, SQL Performance Analyzer will use the database link for all TEST EXECUTE and EXPLAIN PLAN operations by sending the SQL statements to the remote database to be processed remotely. The analysis results will still be stored on the local database.</td>
</tr>
<tr>
<td></td>
<td>• DAYS_TO_EXPIRE: number of days until the task is deleted</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT_EXECUTION_TYPE: the task will default to this type of execution when none is specified by the EXECUTE_ANALYSIS_TASK Function &amp; Procedure.</td>
</tr>
</tbody>
</table>
Table 158-10  (Cont.) SET_ANALYSIS_TASK_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter (contd.)</td>
<td>• DISABLE_MULTI_EXEC: SQL statements are executed multiple times and runtime statistics are then averaged. Set this parameter to TRUE to disable this capability. In this case, each SQL in the SQL tuning set is executed only once.</td>
</tr>
<tr>
<td></td>
<td>• EXECUTE_TRIGGERS: Set this parameter to TRUE to execute all statement-level triggers in the FULLDML mode. If the parameter is set to FALSE, then the triggers will not be executed even in FULLDML mode of test execution. Any changes incurred due to potential execution of triggers are always rolled back by SPA. The default value of this parameter is FALSE.</td>
</tr>
<tr>
<td></td>
<td>• EXECUTION_DAYS_TO_EXPIRE: number of days until the tasks's executions will be deleted (without deleting the task)</td>
</tr>
<tr>
<td></td>
<td>• EXECUTE_FULLDML: TRUE to execute DML statement fully, including acquiring row locks and modifying rows; FALSE (default) to execute only the query part of the DML without modifying data. When TRUE, SQL Performance Analyzer will issue a rollback following DML execution to prevent persistent changes from being made by the DML.</td>
</tr>
<tr>
<td></td>
<td>• EXECUTION_NAME1: name of the first task execution to analyze</td>
</tr>
<tr>
<td></td>
<td>• EXECUTION_NAME2: name of the second task execution to analyze</td>
</tr>
<tr>
<td></td>
<td>• LOCAL_TIME_LIMIT: per-statement time out (seconds)</td>
</tr>
<tr>
<td></td>
<td>• METRIC_DELTA_THRESHOLD: threshold of the difference between the SQL performance metric before and after the change. The default value is zero.</td>
</tr>
<tr>
<td></td>
<td>• NUM_ROWS_TO_FETCH: specifies the number of rows to be fetched for an SQL query. You can use one of the following values:</td>
</tr>
<tr>
<td></td>
<td>- ALL_ROWS: Fetches all the rows for an SQL query</td>
</tr>
<tr>
<td></td>
<td>- AVERAGE: Number of result rows is calculated as the ratio of total rows processed and total executions for each SQL in the STS</td>
</tr>
<tr>
<td></td>
<td>- AUTO: Number of result rows is determined using the value of optimizer_mode parameter of the optimizer environment captured in the STS. If the value of optimizer_mode is ALL_ROWS, then all result rows will be fetched. If its value is FIRST_ROWS_n, then n result rows will be fetched by the SPA.</td>
</tr>
<tr>
<td></td>
<td>- A valid number: Fetches the exact number of rows specified by in the SQL query</td>
</tr>
<tr>
<td></td>
<td>The default value is ALL_ROWS.</td>
</tr>
<tr>
<td></td>
<td>• PLAN_FILTER: plan filter for SQL tuning set (see SELECT_SQLSET for possible values)</td>
</tr>
<tr>
<td></td>
<td>• PLAN_LINES_COMPARISON:</td>
</tr>
<tr>
<td></td>
<td>- ALWAYS --line by line comparison of plans in all scenarios.</td>
</tr>
</tbody>
</table>
|                 |   - AUTO -Line by Line comparison of plans only if phv2 is not available and phv1 is different- NONE (default) - line by line comparison of plans only if phv is unknown
Table 158-10  (Cont.) SET_ANALYSIS_TASK_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANK_MEASURE1</td>
<td>first ranking measure for SQL tuning set</td>
</tr>
<tr>
<td>RANK_MEASURE2</td>
<td>second possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td>RANK_MEASURE3</td>
<td>third possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td>REPLACE_SYSDATE_WITH</td>
<td>Returns a fixed date for all calls to SYSDATE within the SPA task execution. You can use one of the following values:</td>
</tr>
<tr>
<td>- CURRENT_SYSDATE</td>
<td>SYSDATE calls return the current date.</td>
</tr>
<tr>
<td>- SQLSET_SYSDATE</td>
<td>SYSDATE calls return the value of the column LAST_EXEC_START_TIME in the STS. The default value is CURRENT_SYSDATE.</td>
</tr>
<tr>
<td>RESUME_FILTER</td>
<td>a extra filter for SQL tuning sets besides BASIC_FILTER</td>
</tr>
<tr>
<td>SQL_IMPACT_THRESHOLD</td>
<td>threshold of a change impact on a SQL statement. Same as the previous parameter, but at the level of the SQL statement.</td>
</tr>
<tr>
<td>SQL_LIMIT</td>
<td>maximum number of SQL statements to process</td>
</tr>
<tr>
<td>SQL_PERCENTAGE</td>
<td>percentage filter of SQL tuning set statements</td>
</tr>
<tr>
<td>SQLSET_NAME</td>
<td>name of the SQL tuning set to associate to the specified task or task execution. This parameter is mainly using in comparing two SQL tuning sets using SPA.</td>
</tr>
<tr>
<td>SQLSET_OWNER</td>
<td>owner of the SQL tuning set specified using task parameter SQLSET_NAME.</td>
</tr>
<tr>
<td>TIME_LIMIT</td>
<td>global time out (seconds)</td>
</tr>
<tr>
<td>WORKLOAD_IMPACT_THRESHOLD</td>
<td>threshold of a SQL statement impact on a workload. Statements which workload change impact is below the absolute value of this threshold will be ignored and not considered for improvement or regression.</td>
</tr>
<tr>
<td>CON_DBID_MAPPING</td>
<td>provide a mapping of multitenant container database (CDB) IDs. When it is set, SQL Performance Analyzer uses the new CDB ID when it finds a match for the old CDB ID and executes the SQL in that container.</td>
</tr>
</tbody>
</table>

value

New value of the specified parameter

test_execute_dop

Specifies the requested level of concurrency with which a SPA task should be executed.

Values 0 or 1 indicate that the SPA task will run with no additional processes as it used to run in releases prior to Oracle Database 18c Release. A value of n (higher than 1) means that n background SPA processes are being requested to concurrently process the input workload.
Table 158-10  (Cont.) SET_ANALYSIS_TASK_PARAMETER Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| compare_resultset    | Directs SPA to detect if the result-sets between the two trials being compared are different. If differences are seen in the result-sets of any SQL statement between the two trials being compared, the SPA comparison report will indicate this for every such SQL statement.  
  • If set to `TRUE` the result set comparison will be performed  
  • If set to `FALSE` result set comparison will not be performed. |

Usage Notes

The actual number of processes granted might be equal to or lower than the number requested using the `test_execute_dop` parameter. This parameter applies only to `test-execute` or `explain plan` type of trials that process a SQL Tuning set.

Examples

To request two concurrent processes to execute the SPA task:

```sql
DBMS_SQLPA.set_analysis_task_parameter(:tname,'TEST_EXECUTE_DOP',2)
```

To enable result-set validation:

```sql
exec
DBMS_SQLPA.set_analysis_task_parameter(:atname,'COMPARE_RESULTSET','TRUE')
```

To disable result-set validation:

```sql
exec
DBMS_SQLPA.set_analysis_task_parameter(:atname,'COMPARE_RESULTSET','FALSE')
```

158.3.10 SET_ANALYSIS_DEFAULT_PARAMETER Procedures

This procedure sets the SQL analysis task parameter default value.

Syntax

This form of the procedure updates the default value of an analyzer parameter of type VARCHAR2.

```sql
DBMS_SQLPA.SET_ANALYSIS_DEFAULT_PARAMETER(  
  parameter    IN  VARCHAR2,  
  value        IN  VARCHAR2);
```

This form of the procedure updates the default value of an analyzer parameter of type NUMBER.
DBMS_SQLPA.SET_ANALYSIS_DEFAULT_PARAMETER(
  parameter    IN  VARCHAR2,
  value        IN  NUMBER);

Parameters

Table 158-11  SET_ANALYSIS_DEFAULT_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>Name of the parameter to set. The possible analysis parameters that can be set by this procedure are:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- APPLY_CAPTURED_COMPILEENV: indicates whether the advisor could use the compilation environment captured with the SQL statements. The default is 0 (that is, NO).</td>
</tr>
<tr>
<td></td>
<td>- BASIC_FILTER: basic filter for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>- COMPARISON_METRIC: specify an expression of execution statistics to use in performance comparison (Example: buffer_gets, cpu_time + buffer_gets * 10)</td>
</tr>
<tr>
<td></td>
<td>- DATABASE_LINK: can be set to the global name of a PUBLIC database link. When it is set, SQL Performance Analyzer will use the database link for all TEST EXECUTE and EXPLAIN PLAN operations by sending the SQL statements to the remote database to be processed remotely. The analysis results will still be stored on the local database.</td>
</tr>
<tr>
<td></td>
<td>- DAYS_TO_EXPIRE: number of days until the task is deleted</td>
</tr>
<tr>
<td></td>
<td>- DEFAULT_EXECUTION_TYPE: the task will default to this type of execution when none is specified by the EXECUTE_ANALYSIS_TASK Function &amp; Procedure.</td>
</tr>
<tr>
<td></td>
<td>- EXECUTE_FULLDML: TRUE to execute DML statement fully, including acquiring row locks and modifying rows; FALSE (default) to execute only the query part of the DML without modifying data. When TRUE, SQL Performance Analyzer will issue a rollback following DML execution to prevent persistent changes from being made by the DML.</td>
</tr>
<tr>
<td></td>
<td>- EXECUTION_DAYS_TO_EXPIRE: number of days until the tasks's executions will be deleted (without deleting the task)</td>
</tr>
<tr>
<td></td>
<td>- EXECUTION_NAME1: name of the first task execution to analyze</td>
</tr>
<tr>
<td></td>
<td>- EXECUTION_NAME2: name of the second task execution to analyze</td>
</tr>
<tr>
<td></td>
<td>- LOCAL_TIME_LIMIT: per-statement time out (seconds)</td>
</tr>
<tr>
<td>Parameter (contd.)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PLAN_FILTER:</td>
<td>plan filter for SQL tuning set (see SELECT_SQLSET for possible values)</td>
</tr>
<tr>
<td>RANK_MEASURE1:</td>
<td>first ranking measure for SQL tuning set</td>
</tr>
<tr>
<td>RANK_MEASURE2:</td>
<td>second possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td>RANK_MEASURE3:</td>
<td>third possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td>RESUME_FILTER:</td>
<td>an extra filter for SQL tuning sets besides BASIC_FILTER</td>
</tr>
<tr>
<td>SQL_IMPACT_THRESHOLD:</td>
<td>threshold of a change impact on a SQL statement. Same as the previous parameter, but at the level of the SQL statement.</td>
</tr>
<tr>
<td>SQL_LIMIT:</td>
<td>maximum number of SQL statements to process</td>
</tr>
<tr>
<td>SQL_PERCENTAGE:</td>
<td>percentage filter of SQL tuning set statements</td>
</tr>
<tr>
<td>TIME_LIMIT:</td>
<td>global timeout (seconds)</td>
</tr>
<tr>
<td>WORKLOAD_IMPACT_THRESHOLD:</td>
<td>threshold of a SQL statement impact on a workload. Statements which workload change impact is below the absolute value of this threshold will be ignored and not considered for improvement or regression.</td>
</tr>
</tbody>
</table>

value | New value of the specified parameter |
The **DBMS_SQLTUNE** package provides an interface to manage SQL tuning sets.

This package provides the same subprograms, although in some cases with slightly different names, as the SQL tuning set subprograms in **DBMS_SQLTUNE**. The difference is that **DBMS_SQLSET** does not require the Oracle Tuning Pack.

This chapter contains the following topics:

- DBMS_SQLSET Overview
- DBMS_SQLSET Security Model
- DBMS_SQLSET Data Structures
- Summary of DBMS_SQLSET Subprograms

### 159.1 DBMS_SQLSET Overview

Use this package to manage SQL tuning sets.

> **Note:**
> All **DBMS_SQLSET** subprograms have equivalents in the **DBMS_SQLTUNE** package.

SQL tuning sets store SQL statements along with the following information:

- The execution context, such as the parsing schema name and bind values
- Execution statistics such as average elapsed time and execution count
- Execution plans, which are the sequence of operations that the database performs to run SQL statements
- Row source statistics such as the number of rows processed for each operation executed within the plan

You can create SQL tuning sets by filtering or ranking SQL statements from several sources:

- The shared SQL area using the **SELECT_CURSOR_CACHE** Function
- Top SQL statements from the Automatic Workload Repository using the **SELECT_WORKLOAD_REPOSITORY** Function
- Other SQL tuning sets using the **SELECT_SQLSET** Function
- SQL Performance Analyzer task comparison results using the **SELECT_SQLPA_TASK** Function
- SQL Trace files using the **SELECT_SQL_TRACE** Function
- A user-defined workload
The complete group of subprograms listed in **Summary of DBMS_SQLSET Subprograms** facilitates this functionality. As examples:

- The **CREATE_SQLSET Procedure and Function** creates a SQL tuning set object in the database.
- The **LOAD_SQLSET Procedure** populates the SQL tuning set with a set of selected SQL.
- The **CAPTURE_CURSOR_CACHE Procedure** collects SQL statements from the shared SQL area over a specified time interval, attempting to build a realistic picture of database workload.

### 159.2 DBMS_SQLSET Security Model

This package is available to PUBLIC and performs its own security checking.

SQL tuning set subprograms require either the **ADMINISTER SQL TUNING SET** or the **ADMINISTER ANY SQL TUNING SET** privilege. Users having the **ADMINISTER SQL TUNING SET** privilege can only create and modify a SQL tuning set that they own. The **ADMINISTER ANY SQL TUNING SET** privilege allows users to manipulate all SQL tuning sets, even those owned by other users. For example, you can use **CREATE_SQLSET Procedure and Function** to create a SQL tuning set to be owned by a different user. In this case, the different user need not have the **ADMINISTER SQL TUNING SET** privilege to manipulate the SQL tuning set.

### 159.3 DBMS_SQLSET Data Structures

The **SELECT_*** subprograms in the **DBMS_SQLTUNE** package return objects of the **SQLSET_ROW** type.

#### Object Types

- **SQLSET_ROW Object Type**

#### 159.3.1 SQLSET_ROW Object Type

The **SQLSET_ROW** object models the content of a SQL tuning set for the user.

Logically, a SQL tuning set is a collection of **SQLSET_ROW** objects. Each **SQLSET_ROW** contains a single SQL statement along with its execution context, statistics, binds, and plan. The **SELECT_*** subprograms each model a data source as a collection of **SQLSET_ROW** objects, with each object uniquely identified by (\texttt{sql_id}, \texttt{plan_hash_value}). Similarly, the **LOAD_SQLSET** procedure takes as input a cursor whose row type is **SQLSET_Row**, treating each **SQLSET_ROW** in isolation according to the policies requested by the user.

Several subprograms package accept basic filters on the content of a SQL tuning set or data source. These filters are expressed in terms of the attributes within the **SQLSET_ROW** as defined.

#### Syntax

```sql
CREATE TYPE sqlset_row AS object (
    sql_id VARCHAR(13),
```

---

**ORACLE**

**Chapter 159**

**DBMS_SQLSET Security Model**

159-2
Attributes

Table 159-1  SQLSET_ROW Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>Unique SQL ID.</td>
</tr>
<tr>
<td>forcing_matching_signature</td>
<td>Signature with literals, case, and whitespace removed.</td>
</tr>
<tr>
<td>sql_text</td>
<td>Full text for the SQL statement.</td>
</tr>
<tr>
<td>object_list</td>
<td>Currently not implemented.</td>
</tr>
<tr>
<td>bind_data</td>
<td>Bind data as captured for this SQL. Note that you cannot stipulate an argument for this parameter and also for bind_list - they are mutually exclusive.</td>
</tr>
<tr>
<td>parsing_schema_name</td>
<td>Schema where the SQL is parsed.</td>
</tr>
<tr>
<td>module</td>
<td>Last application module for the SQL.</td>
</tr>
<tr>
<td>action</td>
<td>Last application action for the SQL.</td>
</tr>
<tr>
<td>elapsed_time</td>
<td>Sum total elapsed time for this SQL statement.</td>
</tr>
<tr>
<td>cpu_time</td>
<td>Sum total CPU time for this SQL statement.</td>
</tr>
<tr>
<td>buffer_gets</td>
<td>Sum total number of buffer gets.</td>
</tr>
</tbody>
</table>

Chapter 159  DBMS_SQLSET Data Structures
Table 159-1 (Cont.) SQLSET_ROW Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk_reads</td>
<td>Sum total number of disk reads.</td>
</tr>
<tr>
<td>direct_writes</td>
<td>Sum total number of direct path writes.</td>
</tr>
<tr>
<td>rows_processed</td>
<td>Sum total number of rows processed by this SQL.</td>
</tr>
<tr>
<td>fetches</td>
<td>Sum total number of fetches.</td>
</tr>
<tr>
<td>executions</td>
<td>Total executions of this SQL statement.</td>
</tr>
<tr>
<td>end_of_fetch_count</td>
<td>Number of times the SQL statement was fully executed with all of its rows</td>
</tr>
<tr>
<td></td>
<td>fetched.</td>
</tr>
<tr>
<td>optimizer_cost</td>
<td>Optimizer cost for this SQL.</td>
</tr>
<tr>
<td>optimizer_env</td>
<td>Optimizer environment for this SQL statement.</td>
</tr>
<tr>
<td>priority</td>
<td>User-defined priority (1,2,3).</td>
</tr>
<tr>
<td>command_type</td>
<td>Statement type, such as INSERT or SELECT.</td>
</tr>
<tr>
<td>first_load_time</td>
<td>Load time of the parent cursor.</td>
</tr>
<tr>
<td>stat_period</td>
<td>Period of time (seconds) when the statistics of this SQL statement were</td>
</tr>
<tr>
<td></td>
<td>collected.</td>
</tr>
<tr>
<td>active_stat_period</td>
<td>Effective period of time (in seconds) during which the SQL statement was</td>
</tr>
<tr>
<td></td>
<td>active.</td>
</tr>
<tr>
<td>other</td>
<td>Other column for user-defined attributes.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Plan hash value of the plan.</td>
</tr>
<tr>
<td>sql_plan</td>
<td>Execution plan for the SQL statement.</td>
</tr>
<tr>
<td>bind_list</td>
<td>List of user-specified binds for the SQL statement. This is used for</td>
</tr>
<tr>
<td></td>
<td>user-specified workloads. Note that you cannot stipulate an argument for</td>
</tr>
<tr>
<td></td>
<td>this parameter and also for bind_data: they are mutually exclusive.</td>
</tr>
<tr>
<td>con_dbid</td>
<td>DBID of the PDB or CDB root.</td>
</tr>
<tr>
<td>last_exec_start_time</td>
<td>Most recent execution start time of this SQL statement.</td>
</tr>
</tbody>
</table>

159.4 Summary of DBMS_SQLSET Subprograms

This table lists the DBMS_SQLSET subprograms and briefly describes them.

Table 159-2 DBMS_SQLSET Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_REFERENCE Function</td>
<td>This procedure adds a new reference to an existing SQL tuning set to indicate</td>
</tr>
<tr>
<td></td>
<td>its use by a client.</td>
</tr>
<tr>
<td>CAPTURE_CURSOR_CACHE Procedure</td>
<td>This procedure captures a workload from the shared SQL area into a SQL</td>
</tr>
<tr>
<td></td>
<td>tuning set.</td>
</tr>
<tr>
<td>CREATE_SQLSET Procedure and Function</td>
<td>This procedure or function creates a SQL tuning set object in the database.</td>
</tr>
</tbody>
</table>
### Table 159-2  (Cont.) DBMS_SQLSET Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_STGTAB Procedure</td>
<td>This procedure creates a staging table through which SQL tuning sets are imported and exported.</td>
</tr>
<tr>
<td>DELETE_SQLSET Procedure</td>
<td>This procedure deletes a set of SQL statements from a SQL tuning set.</td>
</tr>
<tr>
<td>DROP_SQLSET Procedure</td>
<td>This procedure drops a SQL tuning set if it is not active.</td>
</tr>
<tr>
<td>LOAD_SQLSET Procedure</td>
<td>This procedure populates the SQL tuning set with a set of selected SQL statements. You can call the procedure multiple times to add new SQL statements or replace attributes of existing statements.</td>
</tr>
<tr>
<td>PACK_STGTAB Procedure</td>
<td>This procedure copies one or more SQL tuning sets from their location in the SYS schema to a staging table created by the CREATE_STGTAB procedure.</td>
</tr>
<tr>
<td>REMAP_STGTAB Procedure</td>
<td>This procedure changes the tuning set names and owners in the staging table so that they can be unpacked with different values.</td>
</tr>
<tr>
<td>REMOVE_REFERENCE Procedure</td>
<td>This procedure deactivates a SQL tuning set to indicate that it is no longer used by the client.</td>
</tr>
<tr>
<td>SELECT_CURSOR_CACHE Function</td>
<td>This function collects SQL statements from the workload repository.</td>
</tr>
<tr>
<td>SELECT_SQL_TRACE Function</td>
<td>This table function reads the content of one or more trace files and returns the SQL statements it finds in the format of sqlset_row.</td>
</tr>
<tr>
<td>SELECT_SQLPA_TASK Function</td>
<td>This function collects SQL statements from a SQL Performance Analyzer comparison task.</td>
</tr>
<tr>
<td>SELECT_SQLSET Function</td>
<td>This is a table function that reads the contents of a SQL tuning set.</td>
</tr>
<tr>
<td>SELECT_WORKLOAD_REPOSITORY Function</td>
<td>This function collects SQL statements from the workload repository.</td>
</tr>
<tr>
<td>UNPACK_STGTAB Procedure</td>
<td>This procedure copies one or more SQL tuning sets from their location in the staging table into the SQL tuning sets schema, making them proper SQL tuning sets.</td>
</tr>
<tr>
<td>UPDATE_SQLSET Procedures</td>
<td>This overloaded procedure updates selected fields for SQL statements in a SQL tuning set.</td>
</tr>
</tbody>
</table>
159.4.1 ADD_REFERENCE Function

This procedure adds a new reference to an existing SQL tuning set to indicate its use by a client.

Syntax

```sql
DBMS_SQLSET.ADD_REFERENCE (
    sqlset_name  IN  VARCHAR2,
    description  IN  VARCHAR2 := NULL,
    sqlset_owner IN  VARCHAR2 :=NULL
) RETURN NUMBER;
```

Parameters

The parameters are identical for `DBMS_SQLTUNE.ADD_SQLSET_REFERENCE` and `DBMS_SQLSET.ADD_REFERENCE`.

Table 159-3  ADD_SQLSET_REFERENCE and ADD_REFERENCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>description</td>
<td>Provides an optional description of the usage of SQL tuning set.</td>
</tr>
<tr>
<td></td>
<td>The description is truncated if longer than 256 characters.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for the current schema owner.</td>
</tr>
</tbody>
</table>

Return Values

The identifier of the added reference.

Usage Notes

Adding a reference to a SQL tuning set prevents the tuning set from being modified while it is being used. Invoking SQL Tuning Advisor on the SQL tuning set adds a reference automatically, so use `ADD_REFERENCE` only when the automatically generated reference is not sufficient. The `ADD_REFERENCE` function returns a reference ID that you can later supply to the `REMOVE_SQLSET_REFERENCE` procedure. Query the `DBA_SQLSET_REFERENCES` view to find all references to a specified SQL tuning set.

Examples

This example generates a reference to the SQL tuning set named `my_workload` and stores it in the `b_rid` variable.

```sql
VARIABLE b_rid NUMBER;
EXEC :b_rid := DBMS_SQLSET.ADD_REFERENCE(sqlset_name => 'my_workload',
description => 'my sts ref');
```
159.4.2 CAPTURE_CURSOR_CACHE Procedure

This procedure captures a workload from the shared SQL area into a SQL tuning set.

The procedure polls the cache multiple times over a time period, and updates the workload data stored there. It can execute over as long a period as required to capture an entire system workload.

Syntax

```sql
DBMS_SQLSET.CAPTURE_CURSOR_CACHE(
    sqlset_name         IN VARCHAR2,
    time_limit          IN POSITIVE := 1800,
    repeat_interval     IN POSITIVE := 300,
    capture_option      IN VARCHAR2 := 'MERGE',
    capture_mode        IN NUMBER   := MODE_REPLACE_OLD_STATS,
    basic_filter        IN VARCHAR2 := NULL,
    sqlset_owner        IN VARCHAR2 := NULL,
    recursive_sql       IN VARCHAR2 := HAS_RECURSIVE_SQL);
```

Parameters

The parameters are the same for both `DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET` and `DBMS_SQLSET.CAPTURE_CURSOR_CACHE`.

Table 159-4  CAPTURE_CURSOR_CACHE_SQLSET and CAPTURE_CURSOR_CACHE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the SQL tuning set name</td>
</tr>
<tr>
<td>time_limit</td>
<td>Defines the total amount of time, in seconds, to execute.</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>Defines the amount of time, in seconds, to pause between sampling.</td>
</tr>
<tr>
<td>capture_option</td>
<td>Specifies whether to insert new statements, update existing statements, or both.</td>
</tr>
<tr>
<td>capture_mode</td>
<td>Specifies the capture mode (UPDATE and MERGE capture options). Possible values:</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Defines a filter to apply to the shared SQL area for each sample. If basic_filter is not set by the caller, then the subprogram captures only statements of type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set or NULL for current schema owner</td>
</tr>
</tbody>
</table>
Table 159-4  (Cont.) CAPTURE_CURSOR_CACHE_SQLSET and CAPTURE_CURSOR_CACHE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive_sql</td>
<td>Defines a filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).</td>
</tr>
</tbody>
</table>

Examples

In this example capture takes place over a 30-second period, polling the cache once every five seconds. This captures all statements run during that period but not before or after. If the same statement appears a second time, the process replaces the stored statement with the new occurrence.

Note that in production systems the time limit and repeat interval would be set much higher. You should tune the time_limit and repeat_interval parameters based on the workload time and shared SQL area turnover properties of your system.

EXEC DBMS_SQLSET.CAPTURE_CURSOR_CACHE( -
  sqlset_name     => 'my_workload', -
  time_limit      =>  30, -
  repeat_interval =>  5);

In the following call you accumulate execution statistics as you go. This option produces an accurate picture of the cumulative activity of each cursor, even across age-outs, but it is more expensive than the previous example.

EXEC DBMS_SQLSET.CAPTURE_CURSOR_CACHE( -
  sqlset_name     => 'my_workload', -
  time_limit      =>  30, -
  repeat_interval =>  5, -
  capture_mode    => DBMS_SQLSET.MODE_ACCUMULATE_STATS);

This call performs a very inexpensive capture where you only insert new statements and do not update their statistics once they have been inserted into the SQL tuning set.

EXEC DBMS_SQLSET.CAPTURE_CURSOR_CACHE( -
  sqlset_name     => 'my_workload', -
  time_limit      =>  30, -
  repeat_interval =>  5, -
  capture_option  => 'INSERT');
159.4.3 CREATE_SQLSET Procedure and Function

This procedure or function creates a SQL tuning set object in the database.

Syntax

```
DBMS_SQLSET.CREATE_SQLSET (
    sqlset_name  IN  VARCHAR2,
    description  IN  VARCHAR2 := NULL,
    sqlset_owner IN  VARCHAR2 := NULL);
```

```
DBMS_SQLSET.CREATE_SQLSET (    
    sqlset_name  IN  VARCHAR2 := NULL,
    description  IN  VARCHAR2 := NULL,
    sqlset_owner IN  VARCHAR2 := NULL)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the created SQL tuning set. The name is the name</td>
</tr>
<tr>
<td></td>
<td>passed to the function. If no name is passed to the function, then the</td>
</tr>
<tr>
<td></td>
<td>function generates an automatic name.</td>
</tr>
<tr>
<td>description</td>
<td>Provides an optional description of the SQL tuning set.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for the current schema</td>
</tr>
<tr>
<td></td>
<td>owner.</td>
</tr>
</tbody>
</table>

Return Values

Specifies the name of the created SQL tuning set. The name is the name passed to the function. If no name is passed to the function, then the function generates an automatic name.

Examples

```
EXEC DBMS_SQLSET.CREATE_SQLSET(-
    sqlset_name => 'my_workload', -
    description => 'complete application workload');
```
159.4.4 CREATE_STGTAB Procedure

This procedure creates a staging table through which SQL tuning sets are imported and exported.

Syntax

```
DBMS_SQLSET.CREATE_STGTAB ( 
  table_name        IN VARCHAR2, 
  schema_name       IN VARCHAR2 := NULL, 
  tablespace_name   IN VARCHAR2 := NULL, 
  db_version        IN NUMBER   := NULL);
```

Parameters

Table 159-6  CREATE_STGTAB_SQLSET and CREATE_STGTAB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Specifies the name of the table to create. The name is case sensitive.</td>
</tr>
<tr>
<td>schema_name</td>
<td>Defines the schema in which to create the table, or NULL for the current</td>
</tr>
<tr>
<td></td>
<td>schema. The name is case sensitive.</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Specifies the tablespace in which to store the staging table, or NULL for</td>
</tr>
<tr>
<td></td>
<td>the default tablespace of the current user. The name is case sensitive.</td>
</tr>
<tr>
<td>db_version</td>
<td>Specifies the database version that determines the format of the staging</td>
</tr>
<tr>
<td></td>
<td>table.</td>
</tr>
</tbody>
</table>

You can also create an older database version staging table to export an STS to an older database version. Use one of the following values:

- **NULL** (default) — Specifies the current database version.
- **STS_STGTAB_10_2_VERSION** — Specifies the 10.2 database version.
- **STS_STGTAB_11_1_VERSION** — Specifies the 11.1 database version.
- **STS_STGTAB_11_2_VERSION** — Specifies the 11.2 database version.
- **STS_STGTAB_12_1_VERSION** — Specifies the 12.1 database version.
- **STS_STGTAB_12_2_VERSION** — Specifies the 12.2 database version.

Examples

Create a staging table for packing and eventually exporting a SQL tuning sets:

```
EXEC DBMS_SQLSET.CREATE_STGTAB(table_name => 'STGTAB_SQLSET');
```

Create a staging table to pack a SQL tuning set in Oracle Database 12c Release 1 (12.1.0.2) format:

```
BEGIN
  DBMS_SQLSET.CREATE_STGTAB(
    table_name => 'STGTAB_SQLSET_121',
    db_version => DBMS_SQLSET.STS_STGTAB_12_1_VERSION);
END;
```
159.4.5 DELETE_SQLSET Procedure

This procedure deletes a set of SQL statements from a SQL tuning set.

Syntax

```
DBMS_SQLSET.DELETE_SQLSET(
    sqlset_name   IN  VARCHAR2,
    basic_filter  IN  VARCHAR2 := NULL,
    sqlset_owner  IN  VARCHAR2 := NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate to filter the SQL from the SQL tuning set.</td>
</tr>
<tr>
<td></td>
<td>This basic filter is used as a where clause on the SQL tuning set content</td>
</tr>
<tr>
<td></td>
<td>to select a desired subset of SQL from the SQL tuning set.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for current schema owner.</td>
</tr>
</tbody>
</table>

Examples

```
-- Delete all statements in a sql tuning set.
EXEC DBMS_SQLSET.DELETE_SQLSET(sqlset_name   => 'my_workload');

-- Delete all statements in a sql tuning set which ran for less than a second
EXEC DBMS_SQLSET.DELETE_SQLSET(sqlset_name   => 'my_workload', -
                                basic_filter  => 'elapsed_time < 1000000');
```

159.4.6 DROP_SQLSET Procedure

This procedure drops a SQL tuning set if it is not active.

Syntax

```
DBMS_SQLSET.DROP_SQLSET(
    sqlset_name   IN  VARCHAR2,
    sqlset_owner  IN  VARCHAR2 := NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
</tbody>
</table>
Table 159-8  (Cont.) DROP_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for current schema owner.</td>
</tr>
</tbody>
</table>

Usage Notes

You cannot drop a SQL tuning set when it is referenced by one or more clients.

Examples

```sql
-- Drop the sqlset.
EXEC DBMS_SQLSET.DROP_SQLSET ('my_workload');
```

159.4.7 LOAD_SQLSET Procedure

This procedure populates the SQL tuning set with a set of selected SQL statements. You can call the procedure multiple times to add new SQL statements or replace attributes of existing statements.

Syntax

```sql
DBMS_SQLSET.LOAD_SQLSET (
    sqlset_name       IN  VARCHAR2,
    populate_cursor   IN  sqlset_cursor,
    load_option       IN VARCHAR2 := 'INSERT',
    update_option     IN VARCHAR2 := 'REPLACE',
    update_condition  IN VARCHAR2 :=  NULL,
    update_attributes IN VARCHAR2 :=  NULL,
    ignore_null       IN BOOLEAN  :=  TRUE,
    commit_rows       IN POSITIVE :=  NULL,
    sqlset_owner      IN VARCHAR2 := NULL);
```

Parameters

Table 159-9  LOAD_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of SQL tuning set to be loaded.</td>
</tr>
<tr>
<td>populate_cursor</td>
<td>Specifies the cursor reference to the SQL tuning set to be loaded.</td>
</tr>
</tbody>
</table>
| load_option    | Specifies which statements are loaded into the SQL tuning set. The possible values are:
  - INSERT (default) — Adds only new statements.
  - UPDATE — Updates existing the SQL statements and ignores any new statements.
  - MERGE — Inserts new statements and updates the information of the existing ones. |
Table 159-9  (Cont.) LOAD_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update_option</td>
<td>Specifies how existing SQL statements are updated. This parameter is considered only if load_option is specified with UPDATE or MERGE as an option. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>REPLACE (default) — Updates the statement using the new statistics, bind list, object list, and so on.</td>
</tr>
<tr>
<td></td>
<td>ACCUMULATE — Combines attributes when possible (for example, statistics such as elapsed_time), and otherwise replaces the existing values (for example, module and action) with the provided values. The SQL statement attributes that can be accumulated are: elapsed_time, buffer_gets, direct_writes, disk_reads, row_processed, fetches, executions, end_of_fetch_count, stat_period, and active_stat_period.</td>
</tr>
<tr>
<td>update_condition</td>
<td>Specifies when to perform the update. The procedure only performs the update when the specified condition is satisfied. The condition can refer to either the data source or destination. The condition must use the following prefixes to refer to attributes from the source or the destination:</td>
</tr>
<tr>
<td></td>
<td>OLD — Refers to statement attributes from the SQL tuning set (destination).</td>
</tr>
<tr>
<td></td>
<td>NEW — Refers to statement attributes from the input statements (source).</td>
</tr>
<tr>
<td>update_attributes</td>
<td>Specifies the list of SQL statement attributes to update during a merge or update. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>NULL (default) — Specifies the content of the input cursor except the execution context. On other terms, it is equivalent to ALL without execution contexts such as module and action.</td>
</tr>
<tr>
<td></td>
<td>BASIC — Specifies statistics and binds only.</td>
</tr>
<tr>
<td></td>
<td>TYPICAL — Specifies BASIC with SQL plans (without row source statistics) and without an object reference list.</td>
</tr>
<tr>
<td></td>
<td>ALL — Specifies all attributes, including the execution context attributes such as module and action.</td>
</tr>
<tr>
<td></td>
<td>List of comma separated attribute names to update:</td>
</tr>
<tr>
<td></td>
<td>EXECUTION_CONTEXT</td>
</tr>
<tr>
<td></td>
<td>EXECUTION_STATISTICS</td>
</tr>
<tr>
<td></td>
<td>SQL_BINDS</td>
</tr>
<tr>
<td></td>
<td>SQL_PLAN</td>
</tr>
<tr>
<td></td>
<td>SQL_PLAN_STATISTICS (similar to SQL_PLAN with added row source statistics)</td>
</tr>
<tr>
<td>ignore_null</td>
<td>Specifies whether to update attributes when the new value is NULL. If TRUE, then the procedure does not update an attribute when the new value is NULL. That is, do not override with NULL values unless intentional.</td>
</tr>
</tbody>
</table>
### Table 159-9  (Cont.) LOAD_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit_rows</td>
<td>Specifies whether to commit statements after DML. If a value is provided, then the load commits after each specified number of statements is inserted. If NULL is provided, then the load commits only once, at the end of the operation. Providing a value for this argument enables you to monitor the progress of a SQL tuning set load operation in the <code>DBA_SQLSET</code> views. The <code>STATEMENT_COUNT</code> value increases as new SQL statements are loaded.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Defines the owner of the SQL tuning set, or the current schema owner (or NULL for the current owner).</td>
</tr>
</tbody>
</table>

#### Exceptions

- This procedure returns an error when `sqlset_name` is invalid, or a corresponding SQL tuning set does not exist, or the `populate_cursor` is incorrect and cannot be executed.
- Exceptions are also raised when invalid filters are provided. Filters can be invalid either because they don't parse (for example, they refer to attributes not in `sqlset_row`), or because they violate the user's privileges.

#### Usage Notes

Rows in the input `populate_cursor` must be of type `SQLSET_ROW`.

#### Examples

In this example, you create and populate a SQL tuning set with all shared SQL area statements with an elapsed time of 5 seconds or more, excluding statements that belong to `SYS` schema. You select all attributes of the SQL statements and load them in the tuning set using the default mode. The default mode loads only new statements because the SQL tuning set is empty.

```sql
-- create the tuning set
EXEC DBMS_SQLSET.CREATE_SQLSET('my_workload');

-- populate the tuning set from the shared SQL area
DECLARE
cur DBMS_SQLSET.SQLSET_CURSOR;
BEGIN
OPEN cur FOR SELECT VALUE(P) FROM table(DBMS_SQLSET.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS'' AND elapsed_time > 5000000', NULL, NULL, NULL, NULL, 1, NULL, 'ALL')) P;
DBMS_SQLSET.LOAD_SQLSET(sqlset_name => 'my_workload',
populate_cursor => cur);
```
Now you want to augment this information with what is stored in the workload repository (AWR). You populate the tuning set with 'ACCUMULATE' as your update_option because it is assumed the cursors currently in the cache have aged out since the snapshot was taken.

You omit the elapsed_time filter because it is assumed that any statement captured in AWR is important, but still you throw away the SYS-parsed cursors to avoid recursive SQL.

```sql
DECLARE
  cur DBMS_SQLSET.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLSET.SELECT_WORKLOAD_REPOSITORY(1,2, 'parsing_schema_name <> ''SYS''', NULL, NULL, NULL, 1, NULL, 'ALL'))) P;

  DBMS_SQLSET.LOAD_SQLSET(sqlset_name => 'my_workload', populate_cursor => cur, load_option => 'MERGE', update_option => 'ACCUMULATE');
END;
```

The following example is a simple load that only inserts new statements from the workload repository, skipping existing ones (in the SQL tuning set). Note that 'INSERT' is the default value for the load_option argument of the LOAD_SQLSET procedure.

```sql
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLSET.SELECT_WORKLOAD_REPOSITORY(1,2)) P;
  DBMS_SQLSET.LOAD_SQLSET(sqlset_name => 'my_workload', populate_cursor => cur);
END;
```

The next example demonstrates a load with UPDATE option. This updates statements that already exist in the SQL tuning set but does not add new ones. By default, old statistics are replaced by their new values.

```sql
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
```
159.4.8 PACK_STGTAB Procedure

This procedure copies one or more SQL tuning sets from their location in the SYS schema to a staging table created by the CREATE_STGTAB procedure.

Syntax

\[
\text{DBMS\_SQLSET\_PACK\_STGTAB}\ (\ \\
\quad \text{sqlset\_name} \quad \text{IN VARCHAR2}, \ \\
\quad \text{sqlset\_owner} \quad \text{IN VARCHAR2} \quad := \text{NULL}, \ \\
\quad \text{staging\_table\_name} \quad \text{IN VARCHAR2}, \ \\
\quad \text{staging\_schema\_owner} \quad \text{IN VARCHAR2} \quad := \text{NULL}, \ \\
\quad \text{db\_version} \quad \text{IN NUMBER} \quad := \text{NULL});
\]

Parameters

The parameters are identical for the DBMS_SQLTUNE.PACK_STGTAB_SQLSET and DBMS_SQLSET.PACK_STGTAB procedures.

Table 159-10  PACK_STGTAB_SQLSET and PACK_STGTAB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set to pack. The name is case sensitive. Wildcard characters (%) are permitted.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the category from which to pack SQL tuning sets. The name is case sensitive. Wildcard characters (%) are permitted.</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>Specifies the name of the table to use. The value is case sensitive.</td>
</tr>
<tr>
<td>staging_schema_owner</td>
<td>Specifies the schema where the table resides, or NULL for the current schema. The value is case sensitive.</td>
</tr>
<tr>
<td>db_version</td>
<td>Specifies the database version that determines the format of the staging table. You can also create an older database version staging table to export an STS to an older database version. Use any of the following values:</td>
</tr>
<tr>
<td></td>
<td>- NULL (default) — Specifies the current database version.</td>
</tr>
<tr>
<td></td>
<td>- STS_STGTAB_10_2_VERSION — Specifies the 10.2 database version.</td>
</tr>
<tr>
<td></td>
<td>- STS_STGTAB_11_1_VERSION — Specifies the 11.1 database version.</td>
</tr>
<tr>
<td></td>
<td>- STS_STGTAB_11_2_VERSION — Specifies the 11.2 database version.</td>
</tr>
</tbody>
</table>
Usage Notes

- To move more than one SQL tuning set, call this procedure multiple times. You can then move the populated staging table to a destination database using any method, such as a database link or Oracle Data Pump, and then unpack the SQL tuning set in the destination database.
- This function issues a COMMIT after packing each SQL tuning set. If an error is raised mid-execution, then clear the staging table by deleting its rows.

Examples

Put all SQL tuning sets on the database in the staging table:

BEGIN
    DBMS_SQLSET.PACK_STGTAB(
        sqlset_name        => '%',
        sqlset_owner       => '%',
        staging_table_name => 'STGTAB_SQLSET');
END;

Put only those SQL tuning sets owned by the current user in the staging table:

BEGIN
    DBMS_SQLSET.PACK_STGTAB(
        sqlset_name        => '%',
        staging_table_name => 'STGTAB_SQLSET');
END;

Pack a specific SQL tuning set:

BEGIN
    DBMS_SQLSET.PACK_STGTAB(
        sqlset_name         => 'my_workload',
        staging_table_name  => 'STGTAB_SQLSET');
END;

Pack a second SQL tuning set:

BEGIN
    DBMS_SQLSET.PACK_STGTAB(
        sqlset_name         => 'workload_subset',
        staging_table_name  => 'STGTAB_SQLSET');
END;

Pack the STS my_workload_subset into a staging table stgtab_sqlset created for Oracle Database 11g Release 2 (11.2):

BEGIN
    DBMS_SQLSET.PACK_STGTAB(
        sqlset_name          => 'workload_subset',
        staging_table_name   => 'STGTAB_SQLSET');
END;
159.4.9 REMAP_STGTAB Procedure

This procedure changes the tuning set names and owners in the staging table so that they can be unpacked with different values.

Syntax

```sql
DBMS_SQLSET.REMAP_STGTAB(
    old_sqlset_name        IN VARCHAR2,
    old_sqlset_owner       IN VARCHAR2 := NULL,
    new_sqlset_name        IN VARCHAR2 := NULL,
    new_sqlset_owner       IN VARCHAR2 := NULL,
    staging_table_name     IN VARCHAR2,
    staging_schema_owner   IN VARCHAR2 := NULL,
    old_con_dbid           IN NUMBER   := NULL,
    new_con_dbid           IN NUMBER   := NULL);
```

Parameters

The parameters are identical for the DBMS_SQLTUNE.REMAP_STGTAB_SQLSET and DBMS_SQLSET.REMAP_SQLSET procedures.

**Parameter** | **Description**
--- | ---
old_sqlset_name | Specifies the name of the tuning set to target for a remap operation. Wildcard characters (%) are not supported.
old_sqlset_owner | Specifies the new name of the tuning set owner to target for a remap operation. NULL for current schema owner
new_sqlset_name | Specifies the new name for the tuning set, or NULL to keep the same tuning set name.
new_sqlset_owner | Specifies the new owner for the tuning set, or NULL to keep the same owner name.
staging_table_name | Specifies the name of the table on which to perform the remap operation. The value is case sensitive.
staging_schema_owner | Specifies the name of staging table owner, or NULL for the current schema owner. The value is case sensitive.
old_con_dbid | Specifies the old container DBID to be remapped to a new container DBID. Specify NULL to use the same container DBID. You must provide both old_con_dbid and new_con_dbid for the remap to succeed.
new_con_dbid | Specifies the new container DBID to replace with the old container DBID. Specify NULL to use the same container DBID. You must provide both old_con_dbid and new_con_dbid for the remap to succeed.
Usage Notes

Call this procedure multiple times to remap more than one tuning set name or owner. This procedure only handles one tuning set per call.

Examples

-- Change the name of an STS in the staging table before unpacking it.
BEGIN
   DBMS_SQLSET.REMAP_STGTAB(
      old_sqlset_name => 'my_workload',
      old_sqlset_owner => 'SH',
      new_sqlset_name => 'imp_workload',
      staging_table_name => 'STGTAB_SQLSET');
END;

-- Change the owner of an STS in the staging table before unpacking it.
BEGIN
   DBMS_SQLSET.REMAP_STGTAB(
      old_sqlset_name => 'imp_workload',
      old_sqlset_owner => 'SH',
      new_sqlset_owner => 'SYS',
      staging_table_name => 'STGTAB_SQLSET');
END;

159.4.10 REMOVE_REFERENCE Procedure

This procedure deactivates a SQL tuning set to indicate that it is no longer used by the client.

Syntax

```
DBMS_SQLSET.REMOVE_REFERENCE (
   sqlset_name   IN  VARCHAR2,
   reference_id  IN  NUMBER,
   sqlset_owner  IN  VARCHAR2 := NULL,
   force_remove  IN  NUMBER   := 0);
```

Parameters

The parameters are identical for the DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE and DBMS_SQLSET.REMOVE_REFERENCE procedures.

Table 159-12    REMOVE_SQLSET_REFERENCE and REMOVE_REFERENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>reference_id</td>
<td>Specifies the identifier of the reference to remove.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set (or NULL for the current schema owner).</td>
</tr>
</tbody>
</table>
Table 159-12  (Cont.) REMOVE_SQLSET_REFERENCE and REMOVE_REFERENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force_remove</td>
<td>Specifies whether references can be removed for other users (1) or whether they cannot be removed (0). Setting this parameter to 1 only takes effect when the user has the ADMINISTER ANY SQL TUNING SET privilege. Otherwise, the database only removes references owned by the user.</td>
</tr>
</tbody>
</table>

Examples

You can remove references on a given SQL tuning set when you finish using it and want to make it writable again. The following example removes the reference to my_workload:

```
EXEC DBMS_SQLSET.REMOVE_REFERENCE(sqlset_name   => 'my_workload', -
                                 reference_id  => :rid);
```

Use the DBA_SQLSET_REFERENCES view to find all references to a given SQL tuning set.

159.4.11 SELECT_CURSOR_CACHE Function

This function collects SQL statements from the shared SQL area.

Syntax

```
DBMS_SQLSET.SELECT_CURSOR_CACHE (   
  basic_filter        IN   VARCHAR2 := NULL,   
  object_filter       IN   VARCHAR2 := NULL,   
  ranking_measure1    IN   VARCHAR2 := NULL,   
  ranking_measure2    IN   VARCHAR2 := NULL,   
  ranking_measure3    IN   VARCHAR2 := NULL,   
  result_percentage   IN   NUMBER   := 1,   
  result_limit        IN   NUMBER   := NULL,   
  attribute_list      IN   VARCHAR2 := 'TYPICAL',   
  recursive_sql       IN   VARCHAR2 := HAS_RECURSIVE_SQL)
RETURN sys.sqlset PIPELINED;
```

Parameters

Table 159-13  SELECT_CURSOR_CACHE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate that filters the SQL from the shared SQL area defined on attributes of the SQLSET_ROW. If basic_filter is not set by the caller, then the subprogram captures only statements of the type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.</td>
</tr>
</tbody>
</table>
Table 159-13 (Cont.) SELECT_CURSOR_CACHE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_filter</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>ranking_measure(n)</td>
<td>Defines an ORDER BY clause on the selected SQL.</td>
</tr>
<tr>
<td>result_percentage</td>
<td>Specifies a filter that picks the top n% according to the supplied ranking measure. The value applies only if one ranking measure is supplied.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Defines the top limit SQL from the filtered source ranked by the ranking measure.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>Specifies the list of SQL statement attributes to return in the result. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL — Specifies BASIC plus SQL plan (without row source statistics) and without object reference list (default).</td>
</tr>
<tr>
<td></td>
<td>- BASIC — Specifies all attributes (such as execution statistics and binds) except the plans. The execution context is always part of the result.</td>
</tr>
<tr>
<td></td>
<td>- ALL — Specifies all attributes.</td>
</tr>
<tr>
<td></td>
<td>- Comma-separated list of attribute names. This values returns only a subset of SQL attributes:</td>
</tr>
<tr>
<td></td>
<td>- EXECUTION_STATISTICS</td>
</tr>
<tr>
<td></td>
<td>- BIND_LIST</td>
</tr>
<tr>
<td></td>
<td>- OBJECT_LIST</td>
</tr>
<tr>
<td></td>
<td>- SQL_PLAN</td>
</tr>
<tr>
<td></td>
<td>- SQL_PLAN_STATISTICS — Similar to SQL_PLAN plus row source statistics</td>
</tr>
<tr>
<td>recursive_sql</td>
<td>Specifies that the filter must include recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL, which is the default) or exclude it (NO_RECURSIVE_SQL).</td>
</tr>
</tbody>
</table>

Return Values
This function returns a one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Usage Notes
- Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.
- Users need privileges on the shared SQL area views.

Example 159-1 Statements with 500 or More Buffer Gets
This query obtains the SQL IDs and SQL text for statements with 500 buffer gets:

```sql
SELECT SQL_ID, SQL_TEXT
FROM   TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE('buffer_gets > 500'))
ORDER BY sql_id;
```
Example 159-2  All Information About a Statement

The following query obtains all information about the SQL statement with the SQL ID 4rm4183czbs7j:

```
SELECT * FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE('sql_id = ''4rm4183czbs7j'''));
```

Example 159-3  Multiple Plans for a SQL Statement

A data source may store multiple plans for each SQL statement. The output of the SELECT_CURSOR_CACHE function is a SQL row set object that is uniquely identified by SQL ID and plan hash value. This example queries the plan hash values for the statement with the SQL ID aylm3ssvtrh24:

```
SELECT sql_id, plan_hash_value
FROM table(DBMS_SQLSET.select_cursor_cache('sql_id = ''aylm3ssvtrh24'''))
ORDER BY sql_id, plan_hash_value;
```

Example 159-4  Processing All Statements in the Shared SQL Area

This example processes all statements in the shared SQL area:

```
DECLARE
    cur sys_refcursor;
BEGIN
    OPEN cur FOR
        SELECT VALUE(p)
        FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE) p;
    -- Process each statement in cursor (or pass cursor to load_sqlset).
    CLOSE cur;
END;
/
```

Example 159-5  Process Statements NotParsed by SYS

This example processes all statements not parsed in the SYS schema:

```
DECLARE
    cur sys_refcursor;
BEGIN
    OPEN cur for
        SELECT VALUE(p)
        FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS''')) p;
    -- Process each statement (or pass cursor to load_sqlset).
    CLOSE cur;
```
Example 159-6  All Statements from an Application Module and Action

This example processes all statements from a specified application module and action:

DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(
      DBMS_SQLSET.SELECT_CURSOR_CACHE(
        'module = ''MY_APPLICATION'' and action = ''MY_ACTION''')) p;
  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;/

Example 159-7  All Statements Whose Elapsed Time Is At Least Five Seconds

This example processes all statements that ran for at least five seconds:

DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLSET.SELECT_CURSOR_CACHE('elapsed_time > 5000000')) P;
  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;/

Example 159-8  Statements Parsed in the APPS Schema

This example processes all SQL statements that were parsed in the APPS schema and had more than 100 buffer gets:

DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(
      DBMS_SQLSET.SELECT_CURSOR_CACHE(
        'buffer_gets > 100 and parsing_schema_name = ''APPS'''))p;
  -- Process each statement (or pass cursor to load_sqlset)
CLOSE cur;
END;
/

Example 159-9  Plans and SQL Statements

This example processes all SQL statements exceeding 5 seconds. It also selects the plans for these statements. For performance reasons, the example selects execution statistics and SQL binds. The SQL_PLAN attribute of sqlset_row is NULL.

-- select all statements exceeding 5 seconds in elapsed time, but also
-- select the plans (by default we only select execution stats and
-- binds
-- for performance reasons - in this case the SQL_PLAN attribute of
sqlset_row
-- is NULL)
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE(
      basic_filter => 'elapsed_time > 5000000',
      object_filter => NULL,
      ranking_measure1 => NULL,
      ranking_measure2 => NULL,
      ranking_measure3 => NULL,
      result_percentage => 1,
      result_limit => NULL,
      attribute_list => 'EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN',
      recursive_sql => HAS_RECURSIVE_SQL)) p;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;/

Example 159-10  Top 100 Statements Ordered by Elapsed Time

This example selects the top 100 statements in the shared SQL area, ordered by elapsed time:

DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE(
      basic_filter => NULL,
      object_filter => NULL,
      ranking_measure_1 => 'ELAPSED_TIME',
      ranking_measure_2 => NULL,
      ranking_measure_3 => NULL,
      result_percentage => 1,
      result_limit => NULL,
      attribute_list => 'EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN',
      recursive_sql => HAS_RECURSIVE_SQL)) p;
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLSET.SELECT_CURSOR_CACHE(
      basic_filter => NULL,
      object_filter => NULL,
      ranking_measure_1 => 'BUFFER_GETS',
      ranking_measure_2 => NULL,
      ranking_measure_3 => NULL,
      result_percentage => .9,
      result_limit => NULL,
      attribute_list => 'TYPICAL',
      recursive_sql => HAS_RECURSIVE_SQL)) p;

  -- Process each statement (or pass cursor to load_sqlset).
  CLOSE cur;
END;
/

Example 159-11  Statements Responsible for Most Buffer Gets

This example processes statements that cumulatively account for 90% of the buffer gets in the shared SQL area. The buffer gets of all statements added together is approximately 90% of the sum of all statements currently in the shared SQL area.

DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLSET.SELECT_CURSOR_CACHE(
      basic_filter => NULL,
      object_filter => NULL,
      ranking_measure_1 => 'BUFFER_GETS',
      ranking_measure_2 => NULL,
      ranking_measure_3 => NULL,
      result_percentage => .9,
      result_limit => NULL,
      attribute_list => 'TYPICAL',
      recursive_sql => HAS_RECURSIVE_SQL)) p;

  -- Process each statement (or pass cursor to load_sqlset).
  CLOSE cur;
END;
/

159.4.12 SELECT_SQL_TRACE Function

This table function reads the content of one or more trace files and returns the SQL statements it finds in the format of sqlset_row.

Syntax

DBMS_SQLSET.SELECT_SQL_TRACE (
  directory          IN VARCHAR2,
  file_name           IN VARCHAR2 := NULL,
  mapping_table_name  IN VARCHAR2 := NULL,
  mapping_table_owner IN VARCHAR2 := NULL,
  select_mode         IN POSITIVE := SINGLE_EXECUTION,
  options             IN BINARY_INTEGER := LIMITED_COMMAND_TYPE,
  pattern_start       IN VARCHAR2 := NULL,
Parameters

Table 159-14  SELECT_SQL_TRACE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Defines the directory object containing the trace files. This field is mandatory.</td>
</tr>
<tr>
<td>file_name</td>
<td>Specifies all or part of the name of the trace files. If NULL, then the function uses the current or most recent file in the specified location or path. '％' wildcards are supported for matching trace file names.</td>
</tr>
<tr>
<td>mapping_table_name</td>
<td>Specifies the mapping table name. Note that the mapping table name is case insensitive. If the mapping table name is NULL, then the function uses the mappings in the current database.</td>
</tr>
<tr>
<td>mapping_table_owner</td>
<td>Specifies the mapping table owner. If it is NULL, then the function uses the current user.</td>
</tr>
<tr>
<td>select_mode</td>
<td>Specifies the mode for selecting SQL from the trace. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· SINGLE_EXECUTION — Returns one execution of a SQL. This is the default.</td>
</tr>
<tr>
<td></td>
<td>· ALL_EXECUTIONS — Returns all executions.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies which types of SQL statements are returned.</td>
</tr>
<tr>
<td></td>
<td>· LIMITED_COMMAND_TYPE — Returns the SQL statements with the command types CREATE, INSERT, SELECT, UPDATE, DELETE, and MERGE. This value is the default.</td>
</tr>
<tr>
<td></td>
<td>· ALL_COMMAND_TYPE — Returns the SQL statements with all command types.</td>
</tr>
<tr>
<td>pattern_start</td>
<td>Specifies the delimiting pattern of the trace file sections to consider. CURRENTLY INOPERABLE.</td>
</tr>
<tr>
<td>pattern_end</td>
<td>Specifies the closing delimiting pattern of the trace file sections to process. CURRENTLY INOPERABLE.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Specifies the top SQL from the filtered source. Default to MAXSB4 if NULL.</td>
</tr>
</tbody>
</table>

Return Values

This function returns a SQLSET_ROW object.

Usage Notes

The ability to create a directory object for the system directory creates a potential security issue. For example, in a CDB, all containers write trace files to the same directory. A local user with SELECT privileges on this directory can read the contents of trace files belonging to any container.
To prevent this type of unauthorized access, copy the files from the default SQL trace directory into a different directory, and then create a directory object. Use the `PATH_PREFIX` clause of the `CREATE PLUGGABLE DATABASE` statement to ensure that all directory object paths associated with the PDB are restricted to the specified directory or its subdirectories.

**Examples**

The following code shows how to enable SQL trace for a few SQL statements and load the results into a SQL tuning set:

```sql
-- turn on the SQL trace in the capture database
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 4'

-- run sql statements
SELECT 1 FROM DUAL;
SELECT COUNT(*) FROM dba_tables WHERE table_name = :mytab;

ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF';

-- create mapping table from the capture database
CREATE TABLE mapping AS
SELECT object_id id, owner, substr(object_name, 1, 30) name
FROM dba_objects
WHERE object_type NOT IN ('CONSUMER GROUP', 'EVALUATION CONTEXT',
    'FUNCTION', 'INDEXTYPE', 'JAVA CLASS',
    'JAVA DATA', 'JAVA RESOURCE', 'LIBRARY',
    'LOB', 'OPERATOR', 'PACKAGE',
    'PACKAGE BODY', 'PROCEDURE', 'QUEUE',
    'RESOURCE PLAN', 'TRIGGER', 'TYPE',
    'TYPE BODY')
UNION ALL
SELECT user_id id, username owner, NULL name
FROM dba_users;

-- create the directory object where the SQL traces are stored
CREATE DIRECTORY SQL_TRACE_DIR as '/home/foo/trace';

-- create the STS
EXEC DBMS_SQLSET.CREATE_SQLSET('my_sts', 'test purpose');

-- load the SQL statements into STS from SQL TRACE
DECLARE
    cur sys_refcursor;
BEGIN
    OPEN cur FOR
        SELECT value(p)
        FROM TABLE(
            DBMS_SQLSET.SELECT_SQL_TRACE(  
                directory=>'SQL_TRACE_DIR',
                file_name=>'%trc',
                mapping_table_name=>'mapping')) p;
    DBMS_SQLSET.LOAD_SQLSET('my_sts', cur);
    CLOSE cur;
END;
/
159.4.13 SELECT_SQLPA_TASK Function

This function collects SQL statements from a SQL Performance Analyzer comparison task.

Syntax

```sql
DBMS_SQLSET.SELECT_SQLPA_TASK(
    task_name         IN VARCHAR2,
    task_owner        IN VARCHAR2 := NULL,
    execution_name    IN VARCHAR2 := NULL,
    level_filter      IN VARCHAR2 := 'REGRESSED',
    basic_filter      IN VARCHAR2 := NULL,
    object_filter     IN VARCHAR2 := NULL,
    attribute_list    IN VARCHAR2 := 'TYPICAL')
RETURN sys.sqlset PIPELINED;
```

Parameters

**Table 159-15  SELECT_SQLPA_TASK Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Specifies the name of the SQL Performance Analyzer task.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Specifies the owner of the SQL Performance Analyzer task. If NULL, then assume the current user.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Specifies the name of the SQL Performance Analyzer task execution (type COMPARE PERFORMANCE) from which the provided filters will be applied. If NULL, then assume the most recent COMPARE PERFORMANCE execution.</td>
</tr>
</tbody>
</table>
Table 159-15  (Cont.) SELECT_SQLPA_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level_filter</td>
<td>Specifies which subset of SQL statements to include. Same format as DBMS_SQLPA.REPORT_ANALYSIS_TASK.LEVEL, with some possible strings removed.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate to filter the SQL in addition to the level filters.</td>
</tr>
<tr>
<td>object_filter</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>Defines the SQL statement attributes to return in the result. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL — Returns BASIC plus the SQL plan (without row source statistics) and without an object reference list. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• BASIC — Returns all attributes (such as execution statistics and binds) except the plans. The execution context is always part of the result.</td>
</tr>
<tr>
<td></td>
<td>• ALL — Returns all attributes.</td>
</tr>
<tr>
<td></td>
<td>• Comma-separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics).</td>
</tr>
</tbody>
</table>

Return Values

This function returns a SQL tuning set object.

Usage Notes

For example, you can use this function to create a SQL tuning set containing the subset of SQL statements that regressed during a SQL Performance Analyzer (SPA) experiment. You can also specify other arbitrary filters.

159.4.14 SELECT_SQLSET Function

This is a table function that reads the contents of a SQL tuning set.

Syntax

```sql
DBMS_SQLSET.SELECT_SQLSET (  
    sqlset_name IN VARCHAR2,  
    basic_filter IN VARCHAR2 := NULL,  
    object_filter IN VARCHAR2 := NULL,  
    ranking_measure1 IN VARCHAR2 := NULL,  
    ranking_measure2 IN VARCHAR2 := NULL,  
    ranking_measure3 IN VARCHAR2 := NULL,  
)  ```
result_percentage  IN  NUMBER   :=  1,
result_limit        IN  NUMBER   :=  NULL)
attribute_list      IN  VARCHAR2 := 'TYPICAL',
plan_filter         IN  VARCHAR2 := NULL,
sqlset_owner        IN  VARCHAR2 := NULL,
recursive_sql       IN  VARCHAR2 := HAS_RECURSIVE_SQL)
RETURN sys.sqlset PIPELINED;

Parameters

Table 159-16  SELECT_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set to query.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate to filter the SQL from the SQL tuning set defined on attributes of the SQLSET_ROW.</td>
</tr>
<tr>
<td>object_filter</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>ranking_measure(n)</td>
<td>Specifies an ORDER BY clause on the selected SQL.</td>
</tr>
<tr>
<td>result_percentage</td>
<td>Specifies a filter that picks the top n% according to the supplied ranking measure. Note that this parameter applies only if one ranking measure is supplied.</td>
</tr>
<tr>
<td>result_limit</td>
<td>The top limit SQL from the filtered source, ranked by the ranking measure.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>Defines the SQL statement attributes to return in the result. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>• BASIC — Returns all attributes (such as execution statistics and binds) except the plans. The execution context is included in the result.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL — Returns BASIC plus the SQL plan, but without row source statistics and without the object reference list. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• ALL — Returns all attributes.</td>
</tr>
<tr>
<td></td>
<td>• Comma-separated list of attribute names. This value enables the function to return only a subset of SQL attributes:</td>
</tr>
<tr>
<td></td>
<td>• EXECUTION_STATISTICS</td>
</tr>
<tr>
<td></td>
<td>• SQL_BINDS</td>
</tr>
<tr>
<td></td>
<td>• SQL_PLAN_STATISTICS (similar to SQL_PLAN plus row source statistics)</td>
</tr>
</tbody>
</table>
Table 159-16  (Cont.) SELECT_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_filter</td>
<td>Specifies the plan filter. This parameter enables you to select a single plan when a statement has multiple plans. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• LAST_GENERATED — Returns the plan with the most recent timestamp.</td>
</tr>
<tr>
<td></td>
<td>• FIRST_GENERATED — Returns the plan with the least recent timestamp.</td>
</tr>
<tr>
<td></td>
<td>• LAST_LOADED — Returns the plan with the most recent FIRST_LOAD_TIME statistical information.</td>
</tr>
<tr>
<td></td>
<td>• FIRST_LOADED — Returns the plan with the least recent FIRST_LOAD_TIME statistical information.</td>
</tr>
<tr>
<td></td>
<td>• MAX_ELAPSED_TIME — Returns the plan with the maximum elapsed time.</td>
</tr>
<tr>
<td></td>
<td>• MAX_BUFFER_GETS — Returns the plan with the maximum buffer gets.</td>
</tr>
<tr>
<td></td>
<td>• MAX_DISK_READS — Returns the plan with the maximum disk reads.</td>
</tr>
<tr>
<td></td>
<td>• MAX_DIRECT_WRITES — Returns the plan with the maximum direct writes.</td>
</tr>
<tr>
<td></td>
<td>• MAX_OPTIMIZER_COST — Returns the plan with the maximum optimizer cost value.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for the current schema owner.</td>
</tr>
<tr>
<td>recursive_sql</td>
<td>Specifies that the filter must include recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL, which is the default) or exclude it (NO_RECURSIVE_SQL).</td>
</tr>
</tbody>
</table>

**Return Values**

This function returns one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

**Usage Notes**

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

**Examples**

```sql
-- select from a sql tuning set
DECLARE
    cur sys_refcursor;
BEGIN
    OPEN cur FOR
        SELECT VALUE (P)
        FROM table(DBMS_SQLSET.SELECT_SQLSET('my_workload')) P;
    -- Process each statement (or pass cursor to load_sqlset)
```

**Chapter 159**

Summary of DBMS_SQLSET Subprograms

159-31
159.4.15 SELECT_WORKLOAD_REPOSITORY Function

This function collects SQL statements from the workload repository.

The overloaded forms enable you to collect SQL statements from the following sources:

- Snapshots between begin_snap and end_snap
- A workload repository baseline

Syntax

\[
\text{DBMS_SQLSET.SELECT_WORKLOAD_REPOSITORY (}
\begin{align*}
\text{begin_snap} & \quad \text{IN NUMBER,} \\
\text{end_snap} & \quad \text{IN NUMBER,} \\
\text{basic_filter} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{object_filter} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{ranking_measure1} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{ranking_measure2} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{ranking_measure3} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{result_percentage} & \quad \text{IN NUMBER := 1,} \\
\text{result_limit} & \quad \text{IN NUMBER := NULL,} \\
\text{attribute_list} & \quad \text{IN VARCHAR2 := 'TYPICAL',} \\
\text{recursive_sql} & \quad \text{IN VARCHAR2 := HAS_RECURSIVE_SQL,} \\
\text{dbid} & \quad \text{IN NUMBER := NULL)} \\
\text{RETURN sys.sqlset PIPELINED;}
\end{align*}
\]

\[
\text{DBMS_SQLSET.SELECT_WORKLOAD_REPOSITORY (}
\begin{align*}
\text{baseline_name} & \quad \text{IN VARCHAR2,} \\
\text{basic_filter} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{object_filter} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{ranking_measure1} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{ranking_measure2} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{ranking_measure3} & \quad \text{IN VARCHAR2 := NULL,} \\
\text{result_percentage} & \quad \text{IN NUMBER := 1,} \\
\text{result_limit} & \quad \text{IN NUMBER := NULL,} \\
\text{attribute_list} & \quad \text{IN VARCHAR2 := 'TYPICAL',} \\
\text{recursive_sql} & \quad \text{IN VARCHAR2 := HAS_RECURSIVE_SQL,} \\
\text{dbid} & \quad \text{IN NUMBER := NULL)} \\
\text{RETURN sys.sqlset PIPELINED;}
\end{align*}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin_snap</td>
<td>Defines the beginning AWR snapshot (non-inclusive).</td>
</tr>
<tr>
<td>end_snap</td>
<td>Defines the ending AWR snapshot (inclusive).</td>
</tr>
</tbody>
</table>
Table 159-17  (Cont.) SELECT_WORKLOAD_REPOSITORY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline_name</td>
<td>Specifies the name of the AWR baseline period.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate to filter the SQL from the workload repository. The filter is defined on attributes of the SQLSET_ROW. If basic_filter is not set by the caller, then the subprogram captures only statements of type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.</td>
</tr>
<tr>
<td>object_filter</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>ranking_measure(n)</td>
<td>Defines an ORDER BY clause on the selected SQL.</td>
</tr>
<tr>
<td>result_percentage</td>
<td>Specifies a filter that picks the top n% according to the supplied ranking measure. Note that this percentage applies only if one ranking measure is given.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Specifies the top limit SQL from the source according to the supplied ranking measure.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>Specifies the SQL statement attributes to return in the result. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>· TYPICAL — Returns BASIC plus SQL plan (without row source statistics) and without object reference list. This is the default.</td>
</tr>
<tr>
<td></td>
<td>· BASIC — Returns all attributes (such as execution statistics and binds) are returned except the plans. The execution context is always part of the result.</td>
</tr>
<tr>
<td></td>
<td>· ALL — Returns all attributes</td>
</tr>
<tr>
<td></td>
<td>· Comma-separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN_STATISTICS (similar to SQL_PLAN plus row source statistics).</td>
</tr>
<tr>
<td>recursive_sql</td>
<td>Specifies the filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).</td>
</tr>
<tr>
<td>dbid</td>
<td>Specifies the DBID for imported or PDB-level AWR data. If NULL, then the function uses the current database DBID.</td>
</tr>
</tbody>
</table>

Return Values

This function returns one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Usage Notes

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user’s security privileges and can contain any constructs and subqueries that user can access, but no more.

Examples

```sql
-- select statements from snapshots 1-2
DECLARE
  cur sys_refcursor;
BEGIN
```
OPEN cur FOR
    SELECT VALUE (P)
    FROM table(DBMS_SQLSET.SELECT_WORKLOAD_REPOSITORY(1,2)) P;

-- Process each statement (or pass cursor to load_sqlset)

CLOSE cur;
END;
/

159.4.16 UNPACK_STGTAB Procedure

This procedure copies one or more SQL tuning sets from their location in the staging table into the SQL tuning sets schema, making them proper SQL tuning sets.

Syntax

DBMS_SQLSET.UNPACK_STGTAB (
    sqlset_name          IN VARCHAR2 := '%',
    sqlset_owner         IN VARCHAR2 := NULL,
    replace              IN BOOLEAN,
    staging_table_name   IN VARCHAR2,
    staging_schema_owner IN VARCHAR2 := NULL);

Parameters

The parameters are identical for DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET and DBMS_SQLSET.UNPACK_STGTAB.

Table 159-18  UNPACK_STGTAB_SQLSET and UNPACK_STGTAB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the tuning set to unpack (not null). Wildcard characters (%) are supported to unpack multiple tuning sets in a single call. For example, specify % to unpack all tuning sets from the staging table.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the name of tuning set owner, or NULL for the current schema owner. Wildcard characters (%) are supported.</td>
</tr>
<tr>
<td>replace</td>
<td>Specifies whether to replace an existing SQL tuning set. If FALSE, then this procedure raises errors when you try to create a tuning set that already exists.</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>Specifies the name of the staging table, moved after a call to the DBMS_SQLTUNE.PACK_STGTAB_SQLSET or DBMS_SQLSET.PACK_STGTAB procedure (case-sensitive).</td>
</tr>
<tr>
<td>staging_schema_owner</td>
<td>Specifies the name of staging table owner, or NULL for the current schema owner (case-sensitive).</td>
</tr>
</tbody>
</table>
Examples

-- unpack all STS in the staging table
EXEC DBMS_SQLSET.UNPACK_STGTAB(sqlset_name => '%', -
   sqlset_owner => '%', -
   replace       => FALSE, -
   staging_table_name => 'STGTAB_SQLSET');

-- errors can arise during STS unpack when a STS in the staging table has the
-- same name/owner as STS on the system. In this case, users should call
-- remap_stgtab_sqlset to patch the staging table and with which to call
-- unpack
-- Replace set to TRUE.
EXEC DBMS_SQLSET.UNPACK_STGTAB(sqlset_name => '%', -
   sqlset_owner => '%', -
   replace       => TRUE, -
   staging_table_name => 'STGTAB_SQLSET');

159.4.17 UPDATE_SQLSET Procedures

This overloaded procedure updates selected fields for SQL statements in a SQL tuning set.

Syntax

```sql
DBMS_SQLSET.UPDATE_SQLSET (
   sqlset_name      IN  VARCHAR2,
   sql_id           IN  VARCHAR2,
   plan_hash_value  IN  NUMBER := NULL,
   attribute_name   IN  VARCHAR2,
   attribute_value  IN  VARCHAR2 := NULL,
   sqlset_owner     IN  VARCHAR2 := NULL);
```

```sql
DBMS_SQLSET.UPDATE_SQLSET (
   sqlset_name      IN  VARCHAR2,
   sql_id           IN  VARCHAR2,
   plan_hash_value  IN  NUMBER := NULL,
   attribute_name   IN  VARCHAR2,
   attribute_value  IN  NUMBER := NULL,
   sqlset_owner     IN  VARCHAR2 := NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>sql_id</td>
<td>Specifies the identifier of the SQL statement to be updated.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Specifies the hash value of the execution plan for a SQL statement. Use this parameter when you want to update the attribute for a specific plan for a statement, but not all plans for the statement.</td>
</tr>
</tbody>
</table>

Chapter 159
Summary of DBMS_SQLSET Subprograms
159-35
### Table 159-19  (Cont.) UPDATE_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_name</td>
<td>Specifies the name of the attribute to be modified. You can update the text field for MODULE, ACTION, PARSING_SCHEMA_NAME, and OTHER. The only numerical field that you can update is PRIORITY. If a statement has multiple plans, then the procedure changes the attribute value for all plans.</td>
</tr>
<tr>
<td>attribute_value</td>
<td>Specifies the new value of the attribute.</td>
</tr>
</tbody>
</table>
160 DBMS_SQLTUNE

The **DBMS_SQLTUNE** package is the interface for tuning SQL on demand. The related package **DBMS_AUTO_SQLTUNE** package provides the interface for SQL Tuning Advisor run as an automated task.

The chapter contains the following topics:

- **DBMS_SQLTUNE Overview**
- **DBMS_SQLTUNE Security Model**
- **DBMS_SQLTUNE Data Structures**
- **DBMS_SQLTUNE Subprogram Groups**
- **Summary of DBMS_SQLTUNE Subprograms**

See Also:

"DBMS_AUTO_SQLTUNE Overview"

160.1 DBMS_SQLTUNE Overview

The **DBMS_SQLTUNE** package provides a number of interrelated areas of functionality.

This section contains the following topics:

- **DBMS_SQLTUNE SQL Tuning Advisor Subprograms**
- **DBMS_SQLTUNE SQL Profile Subprograms**
- **DBMS_SQLTUNE SQL Tuning Set Subprograms**

SQL Tuning Advisor

SQL Tuning Advisor is one of a suite of advisors, a set of expert systems that identifies and helps resolve database performance problems. Specifically, SQL Tuning Advisor automates tuning of problematic SQL statements. It takes one or more SQL statements as input and gives precise advice on how to tune the statements. The advisor provides the advice in the form of SQL actions for tuning the SQL along with their expected performance benefit.

The group of **DBMS_SQLTUNE SQL Tuning Advisor Subprograms** provide a task-oriented interface that enables you to access the advisor. You can call the following subprograms in the order given to use some of SQL Tuning Advisor’s features:

1. **CREATE_TUNING_TASK Functions** creates a tuning task for tuning one or more SQL statements.
2. The **EXECUTE_TUNING_TASK Function and Procedure** executes a previously created tuning task.
3. The **REPORT_TUNING_TASK Function** displays the results of a tuning task.

4. You use the **SCRIPT_TUNING_TASK Function** to create a SQL*Plus script which can then be executed to implement a set of Advisor recommendations.

**SQL Profile Subprograms**

SQL Tuning Advisor may recommend the creation of a SQL profile to improve the performance of a statement. SQL profiles consist of auxiliary statistics specific to the statement. The query optimizer makes estimates about cardinality, selectivity, and cost that can sometimes be off by a significant amount, resulting in poor execution plans. The SQL profile addresses this problem by collecting additional information using sampling and partial execution techniques to adjust these estimates.

The group of **DBMS_SQLTUNE SQL Profile Subprograms** provides a mechanism for delivering statistics to the optimizer that targets one particular SQL statement, and helps the optimizer make good decisions for that statement by giving it the most accurate statistical information possible. For example:

- You can use the **ACCEPT_SQL_PROFILE Procedure and Function** to accept a SQL profile recommended by SQL Tuning Advisor.
- You can alter the **STATUS**, **NAME**, **DESCRIPTION**, and **CATEGORY** attributes of an existing SQL profile with the **ALTER_SQL_PROFILE Procedure**.
- You can drop a SQL profile with the **DROP_SQL_PROFILE Procedure**.

**SQL Tuning Sets**

SQL tuning sets store SQL statements along with the following information:

- The execution context, such as the parsing schema name and bind values
- Execution statistics such as average elapsed time and execution count
- Execution plans, which are the sequence of operations that the database performs to run SQL statements
- Row source statistics such as the number of rows processed for each operation executed within the plan

You can create SQL tuning sets by filtering or ranking SQL statements from several sources:

- The shared SQL area using the **SELECT_CURSOR_CACHE Function**
- Top SQL statements from the Automatic Workload Repository using the **SELECT_WORKLOAD_REPOSITORY Function**
- Other SQL tuning sets using the **SELECT_SQLSET Function**
- SQL Performance Analyzer task comparison results using the **SELECT_SQLPA_TASK Function**
- SQL Trace files using the **SELECT_SQL_TRACE Function**
- A user-defined workload

The complete group of **DBMS_SQLTUNE SQL Tuning Set Subprograms** facilitates this functionality. As examples:

- The **CREATE_SQLSET Procedure and Function** creates a SQL tuning set object in the database.
• The `LOAD_SQLSET Procedure` populates the SQL tuning set with a set of selected SQL.
• The `CAPTURE_CURSOR_CACHE_SQLSET Procedure` collects SQL statements from the shared SQL area over a specified time interval, attempting to build a realistic picture of database workload.

**Note:**
When manipulating SQL tuning sets, you can use `DBMS_SQLSET` as an alternative to `DBMS_SQLTUNE`.

### Import and Export of SQL Tuning Sets and SQL Profiles

Use `DBMS_SQLTUNE` subprograms to move SQL profiles and SQL tuning sets from one system to another using a common programmatic model. In both cases, you create a staging table on the source database and populate this staging table with the relevant data. You then move that staging table to the destination system following the method of your choice (such as Oracle Data Pump, or a database link), where it is used to reconstitute the objects in their original form. The following steps are implemented by means of subprograms included in this package:

1. To create the staging table on the source system, call the `CREATE_STGTAB_SQLPROF Procedure` or the `CREATE_STGTAB_SQLSET Procedure`.
2. To populate the staging table with information from the source system, call the `PACK_STGTAB_SQLPROF Procedure` or `PACK_STGTAB_SQLSET Procedure`.
3. Move the staging table to the destination system.
4. To re-create the object on the new system, call the `UNPACK_STGTAB_SQLPROF Procedure` or the `UNPACK_STGTAB_SQLSET Procedure`.

**See Also:**
*Oracle Database SQL Tuning Guide* for more information about programmatic flow

### Automatic Tuning Task Functions

The automated system task `SYS_AUTO_SQL_TUNING_TASK` is created by the database as part of the catalog scripts. This task automatically chooses a set of high-load SQL from AWR and runs SQL Tuning Advisor on this SQL. The automated task performs the same comprehensive analysis as any other SQL Tuning task.

You can obtain a report on the activity of the Automatic SQL Tuning task through the `DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK API`.

**See Also:**
`DBMS_AUTO_SQLTUNE` for the list of subprograms that you can use to manage the automated SQL tuning task.
Real-Time SQL Monitoring

Real-time SQL Monitoring enables DBAs or performance analysts to monitor the execution of long-running SQL statements while they are executing. Both cursor statistics (such as CPU times and IO times) and execution plan statistics (such as number of output rows, memory and temp space used) are updated in almost real time during statement execution. The \$SQL_MONITOR and \$SQL_PLAN_MONITOR views expose these statistics. In addition, DBMS_SQLTUNE provides the REPORT_SQL_MONITOR and REPORT_SQL_MONITOR_LIST functions to report monitoring information.

Note:

DBMS_SQL_MONITOR also contains the REPORT_SQL_MONITOR and REPORT_SQL_MONITOR_LIST functions.

Tuning a Standby Database Workload

In some cases, a standby database can assume a reporting role in addition to its data protection role. The standby database can have its own workload of queries, some of which may require tuning. You can issue SQL Tuning Advisor statements on a standby database, which is read-only. A standby-to-primary database link enables DBMS_SQLTUNE to write data to and read data from the primary database. The procedures that are eligible for tuning standby workloads include the database_link_to parameter.

160.2 DBMS_SQLTUNE Security Model

This package is available to PUBLIC and performs its own security checking.

Note the following:

- Because SQL Tuning Advisor relies on the advisor framework, all tuning task interfaces (*_TUNING_TASK) require the ADVISOR privilege.
- SQL tuning set subprograms (*_SQLSET) require either of the following privileges:
  - ADMINISTER SQL TUNING SET
    You can only create and modify a SQL tuning set that you own.
  - ADMINISTER ANY SQL TUNING SET
    You can operate on all SQL tuning sets, even those owned by other users.
- In earlier releases, three different privileges were needed to invoke subprograms involving SQL profiles:
  - CREATE ANY SQL PROFILE
  - ALTER ANY SQL PROFILE
  - DROP ANY SQL PROFILE

The preceding privileges have been deprecated in favor of ADMINISTER SQL MANAGEMENT OBJECT.
160.3 DBMS_SQLTUNE Data Structures

The SELECT_ * subprograms in the DBMS_SQLTUNE package return objects of the SQLSET_ROW type.

Object Types

- SQLSET_ROW Object Type

160.3.1 SQLSET_ROW Object Type

The SQLSET_ROW object models the content of a SQL tuning set for the user.

Logically, a SQL tuning set is a collection of SQLSET_ROW objects. Each SQLSET_ROW contains a single SQL statement along with its execution context, statistics, binds, and plan. The SELECT_ * subprograms each model a data source as a collection of SQLSET_ROW objects, with each object uniquely identified by (sql_id, plan_hash_value). Similarly, the LOAD_SQLSET procedure takes as input a cursor whose row type is SQLSET_ROW, treating each SQLSET_ROW in isolation according to the policies requested by the user.

Several subprograms package accept basic filters on the content of a SQL tuning set or data source. These filters are expressed in terms of the attributes within the SQLSET_ROW as defined.

Syntax

```
CREATE TYPE sqlset_row AS object (  
    sql_id                   VARCHAR(13),  
    force_matching_signature NUMBER,  
    sql_text                 CLOB,  
    object_list              sql_objects,  
    bind_data                RAW(2000),  
    parsing_schema_name      VARCHAR2(30),  
    module                   VARCHAR2(48),  
    action                   VARCHAR2(32),  
    elapsed_time             NUMBER,  
    cpu_time                 NUMBER,  
    buffer_gets              NUMBER,  
    disk_reads               NUMBER,  
    direct_writes            NUMBER,  
    rows_processed           NUMBER,  
    fetches                  NUMBER,  
    executions               NUMBER,  
    end_of_fetch_count       NUMBER,  
    optimizer_cost           NUMBER,  
    optimizer_env            RAW(2000),  
    priority                 NUMBER,  
    command_type             NUMBER,  
    first_load_time          VARCHAR2(19),  
    stat_period              NUMBER,  
    active_stat_period       NUMBER,  
    other                    CLOB,  
    plan_hash_value          NUMBER,  
);```

sql_plan                   sql_plan_table_type,
bind_list                  sql_binds,
con_dbid                   NUMBER,
last_exec_start_time      VARCHAR2(19))

Attributes

Table 160-1   SQLSET_ROW Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>Unique SQL ID.</td>
</tr>
<tr>
<td>forcing_matching_signature</td>
<td>Signature with literals, case, and whitespace removed.</td>
</tr>
<tr>
<td>sql_text</td>
<td>Full text for the SQL statement.</td>
</tr>
<tr>
<td>object_list</td>
<td>Currently not implemented.</td>
</tr>
<tr>
<td>bind_data</td>
<td>Bind data as captured for this SQL. Note that you cannot stipulate an</td>
</tr>
<tr>
<td></td>
<td>argument for this parameter and also for bind_list - they are</td>
</tr>
<tr>
<td></td>
<td>mutually exclusive.</td>
</tr>
<tr>
<td>parsing_schema_name</td>
<td>Schema where the SQL is parsed.</td>
</tr>
<tr>
<td>module</td>
<td>Last application module for the SQL.</td>
</tr>
<tr>
<td>action</td>
<td>Last application action for the SQL.</td>
</tr>
<tr>
<td>elapsed_time</td>
<td>Sum total elapsed time for this SQL statement.</td>
</tr>
<tr>
<td>cpu_time</td>
<td>Sum total CPU time for this SQL statement.</td>
</tr>
<tr>
<td>buffer_gets</td>
<td>Sum total number of buffer gets.</td>
</tr>
<tr>
<td>disk_reads</td>
<td>Sum total number of disk reads.</td>
</tr>
<tr>
<td>direct_writes</td>
<td>Sum total number of direct path writes.</td>
</tr>
<tr>
<td>rows_processed</td>
<td>Sum total number of rows processed by this SQL.</td>
</tr>
<tr>
<td>fetches</td>
<td>Sum total number of fetches.</td>
</tr>
<tr>
<td>executions</td>
<td>Total executions of this SQL statement.</td>
</tr>
<tr>
<td>end_of_fetch_count</td>
<td>Number of times the SQL statement was fully executed with all of its</td>
</tr>
<tr>
<td></td>
<td>rows fetched.</td>
</tr>
<tr>
<td>optimizer_cost</td>
<td>Optimizer cost for this SQL.</td>
</tr>
<tr>
<td>optimizer_env</td>
<td>Optimizer environment for this SQL statement.</td>
</tr>
<tr>
<td>priority</td>
<td>User-defined priority (1,2,3).</td>
</tr>
<tr>
<td>command_type</td>
<td>Statement type, such as INSERT or SELECT.</td>
</tr>
<tr>
<td>first_load_time</td>
<td>Load time of the parent cursor.</td>
</tr>
<tr>
<td>stat_period</td>
<td>Period of time (seconds) when the statistics of this SQL statement were</td>
</tr>
<tr>
<td></td>
<td>collected.</td>
</tr>
<tr>
<td>active_stat_period</td>
<td>Effective period of time (in seconds) during which the SQL statement was</td>
</tr>
<tr>
<td></td>
<td>active.</td>
</tr>
<tr>
<td>other</td>
<td>Other column for user-defined attributes.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Plan hash value of the plan.</td>
</tr>
<tr>
<td>sql_plan</td>
<td>Execution plan for the SQL statement.</td>
</tr>
</tbody>
</table>
Table 160-1  (Cont.) SQLSET_ROW Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind_list</td>
<td>List of user-specified binds for the SQL statement. This is used for user-specified workloads. Note that you cannot stipulate an argument for this parameter and also for bind_data; they are mutually exclusive.</td>
</tr>
<tr>
<td>con_dbid</td>
<td>DBID of the PDB or CDB root.</td>
</tr>
<tr>
<td>last_exec_start_time</td>
<td>Most recent execution start time of this SQL statement.</td>
</tr>
</tbody>
</table>

160.4 DBMS_SQLTUNE Subprogram Groups

DBMS_SQLTUNE subprograms are grouped by function.
- DBMS_SQLTUNE SQL Tuning Advisor Subprograms
- DBMS_SQLTUNE SQL Profile Subprograms
- DBMS_SQLTUNE SQL Tuning Set Subprograms
- DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms
- DBMS_SQLTUNE SQL Performance Reporting Subprograms

160.4.1 DBMS_SQLTUNE SQL Tuning Advisor Subprograms

This subprogram group provides an interface to manage SQL tuning tasks.

Table 160-2  SQL Tuning Task Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CANCEL_TUNING_TASK Procedure&quot;</td>
<td>Cancels the currently executing tuning task</td>
</tr>
<tr>
<td>&quot;CREATE_SQL_PLAN_BASELINE Procedure&quot;</td>
<td>Creates a SQL plan baseline for an existing plan</td>
</tr>
<tr>
<td>&quot;CREATE_TUNING_TASK Functions&quot;</td>
<td>Creates a tuning of a single statement or SQL tuning set for either SQL Tuning Advisor</td>
</tr>
<tr>
<td>&quot;DROP_TUNING_TASK Procedure&quot;</td>
<td>Drops a SQL tuning task</td>
</tr>
<tr>
<td>&quot;EXECUTE_TUNING_TASK Function and Procedure&quot;</td>
<td>Executes a previously created tuning task</td>
</tr>
<tr>
<td>&quot;IMPLEMENT_TUNING_TASK Procedure&quot;</td>
<td>Implements a set of SQL profile recommendations made by SQL Tuning Advisor</td>
</tr>
<tr>
<td>&quot;INTERRUPT_TUNING_TASK Procedure&quot;</td>
<td>Intermittently executes tuning task</td>
</tr>
<tr>
<td>&quot;REPORT_AUTO_TUNING_TASK Function&quot;</td>
<td>Displays a report from the automatic tuning task, reporting on a range of executions</td>
</tr>
<tr>
<td>&quot;REPORT_TUNING_TASK Function&quot;</td>
<td>Displays the results of a tuning task</td>
</tr>
<tr>
<td>&quot;RESET_TUNING_TASK Procedure&quot;</td>
<td>Resets the currently executing tuning task to its initial state</td>
</tr>
<tr>
<td>&quot;RESUME_TUNING_TASK Procedure&quot;</td>
<td>Resumes a previously interrupted task that was created to process a SQL tuning set</td>
</tr>
</tbody>
</table>
Table 160-2  (Cont.) SQL Tuning Task Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SCHEDULE_TUNING_TASK Function&quot;</td>
<td>Creates a tuning task and schedules its execution as a scheduler job</td>
</tr>
<tr>
<td>&quot;SCRIPT_TUNING_TASK Function&quot;</td>
<td>Creates a SQL*Plus script which can then be executed to implement a set of SQL Tuning Advisor recommendations</td>
</tr>
<tr>
<td>&quot;SET_TUNING_TASK_PARAMETER Procedures&quot;</td>
<td>Updates the value of a SQL tuning parameter of type VARCHAR2 or NUMBER</td>
</tr>
</tbody>
</table>

"Summary of DBMS_SQLTUNE Subprograms" contains a complete listing of all subprograms in the package.

160.4.2 DBMS_SQLTUNE SQL Profile Subprograms

This subprogram group provides an interface to manage SQL profiles.

Table 160-3  SQL Profile Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT_ALL_SQL_PROFILES Procedure</td>
<td>Accepts all SQL profiles recommended by a specific execution of a tuning task</td>
</tr>
<tr>
<td>ACCEPT_SQLPROFILE Procedure and Function</td>
<td>Creates a SQL profile for the specified tuning task</td>
</tr>
<tr>
<td>ALTER_SQLPROFILE Procedure</td>
<td>Alters specific attributes of an existing SQL profile object</td>
</tr>
<tr>
<td>CREATE_STGTAB_SQLPROF Procedure</td>
<td>Creates the staging table used for copying SQL profiles from one system to another</td>
</tr>
<tr>
<td>DROP_SQLPROFILE Procedure</td>
<td>Drops the named SQL profile from the database</td>
</tr>
<tr>
<td>PACK_STGTAB_SQLPROF Procedure</td>
<td>Moves profile data out of the SYS schema into the staging table</td>
</tr>
<tr>
<td>REMAP_STGTAB_SQLPROF Procedure</td>
<td>Changes the profile data values kept in the staging table prior to performing an unpack operation</td>
</tr>
<tr>
<td>SQLTEXT_TO_SIGNATURE Function</td>
<td>Returns a SQL text's signature</td>
</tr>
<tr>
<td>UNPACK_STGTAB_SQLPROF Procedure</td>
<td>Uses the profile data stored in the staging table to create profiles on this system</td>
</tr>
</tbody>
</table>

"Summary of DBMS_SQLTUNE Subprograms" contains a complete listing of all subprograms in the package.

160.4.3 DBMS_SQLTUNE SQL Tuning Set Subprograms

This subprogram group provides an interface to manage SQL tuning sets.
### Table 160-4  SQL Tuning Set Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_SQLSET_REFERENCE Function</td>
<td>Adds a new reference to an existing SQL tuning set to indicate its use by a client</td>
</tr>
<tr>
<td>CAPTURE_CURSOR_CACHE_SQLSET Procedure</td>
<td>Over a specified time interval incrementally captures a workload from the shared SQL area into a SQL tuning set</td>
</tr>
<tr>
<td>CREATE_SQLSET Procedure and Function</td>
<td>Creates a SQL tuning set object in the database</td>
</tr>
<tr>
<td>CREATE_STGTAB_SQLSET Procedure</td>
<td>Creates a staging table through which SQL Tuning Sets are imported and exported</td>
</tr>
<tr>
<td>DELETE_SQLSET Procedure</td>
<td>Deletes a set of SQL statements from a SQL tuning set</td>
</tr>
<tr>
<td>DROP_SQLSET Procedure</td>
<td>Drops a SQL tuning set if it is not active</td>
</tr>
<tr>
<td>LOAD_SQLSET Procedure</td>
<td>Populates the SQL tuning set with a set of selected SQL</td>
</tr>
<tr>
<td>PACK_STGTAB_SQLSET Procedure</td>
<td>Copies tuning sets out of the SYS schema into the staging table</td>
</tr>
<tr>
<td>REMAP_STGTAB_SQLSET Procedure</td>
<td>Changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system</td>
</tr>
<tr>
<td>REMOVE_SQLSET_REFERENCE Procedure</td>
<td>Deactivates a SQL tuning set to indicate it is no longer used by the client</td>
</tr>
<tr>
<td>SELECT_CURSOR_CACHE Function</td>
<td>Collects SQL statements from the shared SQL area</td>
</tr>
<tr>
<td>SELECT_SQL_TRACE Function</td>
<td>Reads the content of one or more trace files and returns the SQL statements it finds in the format of sqlset_row</td>
</tr>
<tr>
<td>SELECT_SQLPA_TASK Function</td>
<td>Collects SQL statements from a SQL performance analyzer comparison task</td>
</tr>
<tr>
<td>SELECT_SQLSET Function</td>
<td>Collects SQL statements from an existing SQL tuning set</td>
</tr>
<tr>
<td>SELECT_WORKLOAD_REPOSITORY Function</td>
<td>Collects SQL statements from the workload repository</td>
</tr>
<tr>
<td>UNPACK_STGTAB_SQLSET Procedure</td>
<td>Copies one or more SQL tuning sets from the staging table</td>
</tr>
<tr>
<td>UPDATE_SQLSET Procedures</td>
<td>Updates whether selected string fields for a SQL statement in a SQL tuning set or the set numerical attributes of a SQL in a SQL tuning set</td>
</tr>
</tbody>
</table>

The Summary of DBMS_SQLTUNE Subprograms contains a complete listing of all subprograms in the package.

### 160.4.4 DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms

This subprogram group provides function to report on monitoring data collected in V$SQL_MONITOR and V$SQL_PLAN_MONITOR.
### Table 160-5  Real-Time SQL Monitoring Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT_SQL_MONITOR Function</td>
<td>Reports on Real-Time SQL Monitoring</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_LIST Function</td>
<td>Builds a report for all or a subset of statements monitored by Oracle Database</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_LIST_XML Function</td>
<td>Builds an XML report for all or a subset of statements monitored by Oracle Database</td>
</tr>
</tbody>
</table>

#### 160.4.5 DBMS_SQLTUNE SQL Performance Reporting Subprograms

This subprogram group provides detailed reports on SQL performance using statistics from the shared SQL area and automatic workload repository (AWR).

### Table 160-6  SQL Performance Reporting Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT_SQL_DETAIL Function</td>
<td>This function reports on a specific SQL ID.</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR Function</td>
<td>This function builds a report (text, simple HTML, active HTML, XML) for the monitoring information collected on behalf of the targeted statement execution.</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_LIST Function</td>
<td>This function builds a report for all or a sub-set of statements monitored by Oracle. For each statement, the subprogram gives key information and associated global statistics.</td>
</tr>
<tr>
<td>REPORT_TUNING_TASK Function</td>
<td>This function displays the results of a tuning task.</td>
</tr>
<tr>
<td>REPORT_TUNING_TASK_XML Function</td>
<td>This function displays an XML report of a tuning task.</td>
</tr>
</tbody>
</table>

#### 160.5 Summary of DBMS_SQLTUNE Subprograms

This table lists the DBMS_SQLTUNE subprograms and briefly describes them.

### Table 160-7  DBMS_SQLTUNE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT_ALL_SQL_PROFILE Procedure</td>
<td>Accepts all SQL profiles recommended by a particular execution of a particular tuning task</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>ACCEPT_SQL_PROFILE Procedure and Function</td>
<td>Creates a SQL profile for the specified tuning task</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>ADD_SQLSET_REFERENCE Function</td>
<td>Adds a new reference to an existing SQL tuning set to indicate its use by a client</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>ALTER_SQL_PROFILE Procedure</td>
<td>Alters specific attributes of an existing SQL profile object</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>CANCEL_TUNING_TASK Procedure</td>
<td>Cancels the currently executing tuning task</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>CAPTURE_CURSOR_CACHE_SQLSET Procedure</td>
<td>Over a specified time interval incrementally captures a workload from the shared SQL area into a SQL tuning set</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>CREATE_SQL_PLAN_BASELINE Procedure</td>
<td>Creates a SQL plan baseline for an existing plan</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>CREATE_SQLSET Procedure and Function</td>
<td>Creates a SQL tuning set object in the database</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>CREATE_STGTAB_SQLPROFILE Procedure</td>
<td>Creates the staging table used for copying SQL profiles from one system to another</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>CREATE_STGTAB_SQLSET Procedure</td>
<td>Creates a staging table through which SQL tuning sets are imported and exported</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>CREATE_TUNING_TASK Functions</td>
<td>Creates a tuning of a single statement or SQL tuning set for either SQL Tuning Advisor</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>DELETE_SQLSET Procedure</td>
<td>Deletes a set of SQL statements from a SQL tuning set</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>DROP_SQL_PROFILE Procedure</td>
<td>Drops the named SQL profile from the database</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>DROP_SQLSET Procedure</td>
<td>Drops a SQL tuning set if it is not active</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>DROP_TUNING_TASK Procedure</td>
<td>Drops a SQL tuning task</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>EXECUTE_TUNING_TASK Function and Procedure</td>
<td>Executes a previously created tuning task</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>IMPLEMENT_TUNING_TASK Procedure</td>
<td>Implements a set of SQL profile recommendations made by SQL Tuning Advisor</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>INTERRUPT_TUNING_TASK Procedure</td>
<td>Interrupts the currently executing tuning task</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>LOAD_SQLSET Procedure</td>
<td>Populates the SQL tuning set with a set of selected SQL</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>PACK_STGTAB_SQLPROFILE Procedure</td>
<td>Moves profile data out of the SYS schema into the staging table</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>PACK_STGTAB_SQLSET Procedure</td>
<td>Moves tuning sets out of the SYS schema into the staging table</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>REMAP_STGTAB_SQLPROF</td>
<td>Changes the profile data values kept in the staging table prior to performing an unpack operation</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>REMAP_STGTAB_SQLSET</td>
<td>Changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>REMOVE_SQLSET_REFERENCE</td>
<td>Deactivates a SQL tuning set to indicate it is no longer used by the client</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>REPORT_AUTO_TUNING_TASK</td>
<td>Displays a report from the automatic tuning task, reporting on a range of subtasks</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>REPORT_SQL_DETAIL</td>
<td>Reports on a specific SQL ID</td>
<td>DBMS_SQLTUNE SQL Performance Reporting Subprograms</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR</td>
<td>Builds a report (text, simple HTML, active HTML, XML) for the monitoring information collected on behalf of the targeted statement execution</td>
<td>DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_LIST</td>
<td>Builds a report for all or a subset of statements monitored by Oracle Database. For each statement, the subprogram gives key information and associated global statistics</td>
<td>DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms</td>
</tr>
<tr>
<td>REPORT_SQL_MONITOR_LIST_XML</td>
<td>Equivalent to the REPORT_SQL_MONITOR_LIST function, except that it returns XMLType</td>
<td>DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms</td>
</tr>
<tr>
<td>REPORT_TUNING_TASK</td>
<td>Displays the results of a tuning task</td>
<td>DBMS_SQLTUNE SQL Performance Reporting Subprograms</td>
</tr>
<tr>
<td>REPORT_TUNING_TASK_XML</td>
<td>Displays an XML report of a tuning task</td>
<td>DBMS_SQLTUNE SQL Performance Reporting Subprograms</td>
</tr>
<tr>
<td>RESET_TUNING_TASK</td>
<td>Resets the currently executing tuning task to its initial state</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>RESUME_TUNING_TASK</td>
<td>Resumes a previously interrupted task that was created to process a SQL tuning set</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>SCHEDULE_TUNING_TASK</td>
<td>Creates a SQL tuning task and schedule its execution as a scheduler job</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>SCRIPT_TUNING_TASK</td>
<td>Creates a SQL*Plus script which can then be executed to implement a set of SQL Tuning Advisor recommendations</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>SELECT_CURSOR_CACHE</td>
<td>Collects SQL statements from the shared SQL area</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>SELECT_SQL_TRACE</td>
<td>Reads the content of one or more trace files and returns the SQL statements it finds in the format of sqlset_row</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>SELECT_SQLPA_TASK</td>
<td>Collects SQL statements from a SQL Performance Analyzer comparison task</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>SELECT_SQLSET</td>
<td>Collects SQL statements from an existing SQL tuning set</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>SELECT_WORKLOAD_REPOSITORY</td>
<td>Collects SQL statements from the workload repository</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>SET_TUNING_TASK_PARAMETER</td>
<td>Updates the value of a SQL tuning parameter of type VARCHAR2 or NUMBER</td>
<td>DBMS_SQLTUNE SQL Tuning Advisor Subprograms</td>
</tr>
<tr>
<td>SQLTEXT_TO_SIGNATURE</td>
<td>Returns a SQL text's signature</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>UNPACK_STGTAB_SQLPROF</td>
<td>Uses the profile data stored in the staging table to create profiles on this system</td>
<td>DBMS_SQLTUNE SQL Profile Subprograms</td>
</tr>
<tr>
<td>UNPACK_STGTAB_SQLSET</td>
<td>Moves one or more SQL tuning sets from the staging table</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
<tr>
<td>UPDATE_SQLSET</td>
<td>Updates selected fields for a SQL statement in a SQL tuning set</td>
<td>DBMS_SQLTUNE SQL Tuning Set Subprograms</td>
</tr>
</tbody>
</table>

160.5.1 ACCEPT_ALL_SQL_PROFILES Procedure

This procedure accepts all SQL profiles recommended by a specific execution of a tuning task, and sets the attributes of the SQL profiles according to the parameter values passed by the user.

See Also:

DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group
Syntax

```
DBMS_SQLTUNE.ACCEPT_ALL_SQL_PROFILES (
    task_name         IN VARCHAR2,
    category          IN VARCHAR2 := NULL,
    replace           IN BOOLEAN  := FALSE,
    force_match       IN BOOLEAN  := FALSE,
    profile_type      IN VARCHAR2 := REGULAR_PROFILE,
    autotune_period   IN NUMBER   := NULL,
    execution_name    IN VARCHAR2 := NULL,
    task_owner        IN VARCHAR2 := NULL,
    description       IN VARCHAR2 := NULL,
    database_link_to  IN VARCHAR2 := NULL);
```

Parameters

**Table 160-8  ACCEPT_ALL_SQL_PROFILES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The (mandatory) name of the SQL tuning task</td>
</tr>
<tr>
<td>category</td>
<td>This is the category name which must match the value of the SQLTUNE_CATEGORY parameter in a session for the session to use this SQL profile. It defaults to the value &quot;DEFAULT&quot;. This is also the default of the SQLTUNE_CATEGORY parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name creates a unique key for a SQL profile. An ACCEPT_SQL_PROFILE fails if this combination is duplicated.</td>
</tr>
<tr>
<td>replace</td>
<td>If the profile already exists, it is replaced if this argument is TRUE. It is an error to pass a name that is already being used for another signature/category pair, even with replace set to TRUE.</td>
</tr>
<tr>
<td>force_match</td>
<td>If TRUE this causes SQL profiles to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the FORCE option of the cursor_sharing parameter. If FALSE, literals are not transformed. This is analogous to the matching algorithm used by the EXACT option of the cursor_sharing parameter.</td>
</tr>
<tr>
<td>profile_type</td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td>• REGULAR_PROFILE - profile without a change to parallel execution (Default, equivalent to NULL). Note that if the SQL statement currently has a parallel execution plan, the regular profile will cause the optimizer to choose a different, but still parallel, execution plan.</td>
</tr>
<tr>
<td></td>
<td>• PX_PROFILE - regular profile with a change to parallel execution</td>
</tr>
</tbody>
</table>
**Table 160-8  (Cont.) ACCEPT_ALL_SQL_PROFILES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autotune_period</td>
<td>The time period for the automatic SQL tuning. This setting applies only to the automatic SQL Tuning Advisor task. Possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td>- null or negative value (default) - all or full. The result includes all task executions.</td>
</tr>
<tr>
<td></td>
<td>- 0 - result of the current or most recent task execution.</td>
</tr>
<tr>
<td></td>
<td>- 1 - result for the most recent 24-hour period.</td>
</tr>
<tr>
<td></td>
<td>- 7 - result for the most recent 7-day period.</td>
</tr>
<tr>
<td></td>
<td>The procedure interprets any other value as the time of the most recent task execution minus the value of this argument.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name of the task execution to use. If null, then the procedure generates the report for the most recent task execution.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Owner of the tuning task. This is an optional parameter that must be specified to accept a SQL profile associated to a tuning task owned by another user. The current user is the default value.</td>
</tr>
<tr>
<td>description</td>
<td>A user specified string describing the purpose of the SQL profile. The description is truncated if longer than 256 characters. The maximum size is 500 characters.</td>
</tr>
<tr>
<td>database_link_to</td>
<td>Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local.</td>
</tr>
</tbody>
</table>

Use `DBMS_SQLTUNE` to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute `REPORT_TUNING_TASK` locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The `database_link_to` parameter must specify a private database link. This link must be owned by `SYS` and accessed by the default privileged user `SYS$UMP`. The following sample statement creates a link named `lnk_to_pri`:

```
CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS USER SYS$UMP IDENTIFIED BY password USING 'inst1';
```

**Security Model**

The `ADMINISTER SQL MANAGEMENT OBJECT` privilege is required. The `CREATE ANY SQL PROFILE` privilege is deprecated.

**160.5.2 ACCEPT_SQL_PROFILE Procedure and Function**

This subprogram creates a SQL profile recommended by SQL Tuning Advisor.

The SQL text is normalized for matching purposes although it is stored in the data dictionary in denormalized form for readability. SQL text is provided through a reference to the SQL
Tuning task. If the referenced SQL statement does not exist, then the database reports an error.

See Also:

DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```sql
DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
    task_name    IN  VARCHAR2,
    object_id    IN  NUMBER   := NULL,
    name         IN  VARCHAR2 := NULL,
    description  IN  VARCHAR2 := NULL,
    category     IN  VARCHAR2 := NULL,
    task_owner   IN VARCHAR2  := NULL,
    replace      IN BOOLEAN   := FALSE,
    force_match  IN BOOLEAN   := FALSE,
    profile_type IN VARCHAR2  := REGULAR_PROFILE);
```

Parameters

**Table 160-9  ACCEPT_SQL_PROFILE Procedure and Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The (mandatory) name of the SQL tuning task</td>
</tr>
<tr>
<td>object_id</td>
<td>The identifier of the advisor framework object representing the SQL statement associated with the tuning task</td>
</tr>
<tr>
<td>name</td>
<td>The name of the SQL profile. It cannot contain double quotation marks. The name is case sensitive. If not specified, the system generates a unique name for the SQL profile.</td>
</tr>
<tr>
<td>description</td>
<td>A user specified string describing the purpose of the SQL profile. The description is truncated if longer than 256 characters. The maximum size is 500 characters.</td>
</tr>
</tbody>
</table>
Table 160-9  (Cont.) ACCEPT_SQL_PROFILE Procedure and Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>The category name. This name must match the value of the SQLTUNE_CATEGORY parameter in a session for the session to use this SQL profile. It defaults to the value &quot;DEFAULT&quot;. This is also the default of the SQLTUNE_CATEGORY parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name creates a unique key for a SQL profile. An ACCEPT_SQL_PROFILE fails if this combination is duplicated.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Owner of the tuning task. This is an optional parameter that has to be specified to accept a SQL profile associated to a tuning task owned by another user. The current user is the default value.</td>
</tr>
<tr>
<td>replace</td>
<td>If the profile already exists, it is replaced if this argument is TRUE. It is an error to pass a name that is already being used for another signature/category pair, even with replace set to TRUE.</td>
</tr>
<tr>
<td>force_match</td>
<td>If TRUE this causes SQL profiles to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the FORCE option of the cursor_sharing parameter. If FALSE, literals are not transformed. This is analogous to the matching algorithm used by the EXACT option of the cursor_sharing parameter.</td>
</tr>
<tr>
<td>profile_type</td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td>- REGULAR_PROFILE - profile without a change to parallel execution (Default, equivalent to NULL). Note that if the SQL statement currently has a parallel execution plan, the regular profile will cause the optimizer to choose a different, but still parallel, execution plan.</td>
</tr>
<tr>
<td></td>
<td>- PX_PROFILE - regular profile with a change to parallel execution</td>
</tr>
</tbody>
</table>
Table 160-9  (Cont.) ACCEPT_SQL_PROFILE Procedure and Function
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| database_link_to | Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri:

CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1'; |

Return Values

The name of the SQL profile.

Usage Notes

The ADMINISTER SQL MANAGEMENT OBJECT privilege is required. The CREATE ANY SQL PROFILE privilege is deprecated.

Examples

You use both the procedure and the function versions of the subprogram in the same way except you must specify a return value to invoke the function. Here we give examples of the procedure only.

In this example, you tune a single SQL statement form the workload repository and you create the SQL profile recommended by SQL Tuning Advisor.

```sql
VARIABLE stmt_task VARCHAR2(64);
VARIABLE sts_task VARCHAR2(64);

-- create a tuning task tune the statement
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(
    begin_snap => 1, -
    end_snap   => 2, -
    sql_id     => 'aylm3ssvtrh24');

-- execute the resulting task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);
```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE(:stmt_task);

Note that you do not have to specify the ID (that is, object_id) for the advisor framework object created by SQL Tuning Advisor to represent the tuned SQL statement.

You might also want to accept the recommended SQL profile in a different category, (for example, TEST), so that it is not used by default.

EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (  
task_name => :stmt_task, -  
category  => 'TEST');

You can use command ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST' to see how this profile behaves.

The following call creates a SQL profile that targets any SQL statement with the same force_matching_signature as the tuned statement.

EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (task_name => :stmt_task, -  
force_match => TRUE);

In the following example, you tune a SQL tuning set, and you create a SQL profile for only one of the SQL statements in the SQL tuning set. The SQL statement is represented by an advisor framework object with ID equal to 5. You must pass an object ID to the ACCEPT_SQL_PROFILE procedure because there are potentially many SQL profiles for the tuning task. This object ID is given along with the report.

EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK ( -  
sqlset_name  => 'my_workload', -  
rank1         => 'ELAPSED_TIME', -  
time_limit    => 3600,  
description   => 'my workload ordered by elapsed time');

-- execute the resulting task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);

-- create the profile for the sql statement corresponding to object_id = 5.
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (  
task_name => :sts_task, -  
object_id => 5);
### 160.5.3 ADD_SQLSET_REFERENCE Function

This procedure adds a new reference to an existing SQL tuning set to indicate its use by a client.

**See Also:**

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

#### Syntax

```
DBMS_SQLTUNE.ADD_SQLSET_REFERENCE (
    sqlset_name IN VARCHAR2,
    description IN VARCHAR2 := NULL)
RETURN NUMBER;
```

#### Parameters

The parameters are identical for `DBMS_SQLTUNE.ADD_SQLSET_REFERENCE` and `DBMS_SQLSET.ADD_REFERENCE`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>description</td>
<td>Provides an optional description of the usage of SQL tuning set.</td>
</tr>
<tr>
<td></td>
<td>The description is truncated if longer than 256 characters.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for the current schema owner.</td>
</tr>
</tbody>
</table>

#### Return Values

The identifier of the added reference.

#### Examples

You can add reference to a SQL tuning set. This prevents the tuning set from being modified while it is being used. References are automatically added when you invoke SQL Tuning Advisor on the SQL tuning set, so you should use this function for custom purposes only. The function returns a reference ID that is used to remove it later. You use the `REMOVE_SQLSET_REFERENCE` procedure to delete references to a SQL tuning set.

```sql
VARIABLE rid NUMBER;
EXEC :rid := DBMS_SQLTUNE.ADD_SQLSET_REFERENCE( -
```

---

**ORACLE**

---

160-20
You can use the `DBA_SQLSET_REFERENCES` view to find all references on a given SQL tuning set.

### 160.5.4 ALTER_SQL_PROFILE Procedure

This procedure alters specific attributes of an existing SQL profile object.

The following attributes can be altered (using these attribute names):

- **STATUS** can be set to `ENABLED` or `DISABLED`.
- **NAME** can be reset to a valid name which must be a valid Oracle identifier and must be unique.
- **DESCRIPTION** can be set to any string of size no more than 500 characters.
- **CATEGORY** can be reset to a valid category name which must be a valid Oracle identifier and must be unique when combined with normalized SQL text).

#### Syntax

```sql
DBMS_SQLTUNE.ALTER_SQL_PROFILE (name                 IN  VARCHAR2,
                                   attribute_name       IN  VARCHAR2,
                                   value                IN  VARCHAR2);
```

#### Parameters

**Table 160-11  ** ALTER_SQL_PROFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The (mandatory) name of the existing SQL profile to alter</td>
</tr>
<tr>
<td>attribute_name</td>
<td>The (mandatory) attribute name to alter (case insensitive) using valid attribute names</td>
</tr>
<tr>
<td>value</td>
<td>The (mandatory) new value of the attribute using valid attribute values</td>
</tr>
</tbody>
</table>

#### Usage Notes

Requires the `ALTER ANY SQL PROFILE` privilege.

#### Examples

```sql
-- Disable a profile, so it is not be used by any sessions.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name => :pname, -
```
attribute_name  =>  'STATUS', -
value           =>  'DISABLED');

-- Enable it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>   :pname,   -
attribute_name  =>   'STATUS', -
value           =>   'ENABLED');

-- Change the category of the profile so it is used only by sessions
-- with category set to TEST.
-- Use ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST' to see how this
profile
-- behaves.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>  :pname,   -
attribute_name  =>  'CATEGORY', -
value           =>  'TEST');

-- Change it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>  :pname,   -
attribute_name  =>  'CATEGORY', -
value           =>  'DEFAULT');

160.5.5 CANCEL_TUNING_TASK Procedure

This procedure cancels the currently executing tuning task. All intermediate result data
is deleted.

See Also:
DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms
in this group

Syntax

DBMS_SQLTUNE.CANCEL_TUNING_TASK ( task_name         IN VARCHAR2);

Parameters

Table 160-12  CANCEL_TUNING_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Specifies the name of the task to cancel</td>
</tr>
</tbody>
</table>
Examples

You cancel a task when you need to stop it executing and do not require to view any already-completed results.

EXEC DBMS_SQLTUNE.CANCEL_TUNING_TASK(:my_task);

160.5.6 CAPTURE_CURSOR_CACHE_SQLSET Procedure

This procedure captures a workload from the shared SQL area into a SQL tuning set. The procedure polls the cache multiple times over a time period, and updates the workload data stored there. It can execute over as long a period as required to capture an entire system workload.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET (  
    sqlset_name         IN VARCHAR2,  
    time_limit          IN POSITIVE := 1800,  
    repeat_interval     IN POSITIVE := 300,  
    capture_option      IN VARCHAR2 := 'MERGE',  
    capture_mode        IN NUMBER   := MODE_REPLACE_OLD_STATS,  
    basic_filter        IN VARCHAR2 := NULL,  
    sqlset_owner        IN VARCHAR2 := NULL,  
    recursive_sql       IN VARCHAR2 := HAS_RECURSIVE_SQL);

Parameters

The parameters are the same for both DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET and DBMS_SQLSET.CAPTURE_CURSOR_CACHE.

Table 160-13  CAPTURE_CURSOR_CACHE_SQLSET and CAPTURE_CURSOR_CACHE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the SQL tuning set name</td>
</tr>
<tr>
<td>time_limit</td>
<td>Defines the total amount of time, in seconds, to execute.</td>
</tr>
<tr>
<td>repeat_interval</td>
<td>Defines the amount of time, in seconds, to pause between sampling.</td>
</tr>
<tr>
<td>capture_option</td>
<td>Specifies whether to insert new statements, update existing statements, or both. Values are INSERT, UPDATE, or MERGE. The values are the same as for load_option in load_sqlset.</td>
</tr>
</tbody>
</table>
Table 160-13 (Cont.) CAPTURE_CURSOR_CACHE_SQLSET and CAPTURE_CURSOR_CACHE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_mode</td>
<td>Specifies the capture mode (UPDATE and MERGE capture options). Possible values:</td>
</tr>
<tr>
<td></td>
<td>· MODE_REPLACE_OLD_STATS — Replaces statistics when the number of executions is greater than the number stored in the SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>· MODE_ACCUMULATE_STATS — Adds new values to current values for SQL that is already stored. Note that this mode detects if a statement has been aged out, so the final value for a statistics is the sum of the statistics of all cursors that statement existed under.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Defines a filter to apply to the shared SQL area for each sample.</td>
</tr>
<tr>
<td></td>
<td>If basic_filter is not set by the caller, then the subprogram captures only statements of type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set or NULL for current schema owner</td>
</tr>
<tr>
<td>recursive_sql</td>
<td>Defines a filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).</td>
</tr>
</tbody>
</table>

Examples

In this example capture takes place over a 30-second period, polling the cache once every five seconds. This captures all statements run during that period but not before or after. If the same statement appears a second time, the process replaces the stored statement with the new occurrence.

Note that in production systems the time limit and repeat interval would be set much higher. You should tune the time_limit and repeat_interval parameters based on the workload time and shared SQL area turnover properties of your system.

EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
  sqlset_name     => 'my_workload', -
  time_limit      => 30, -
  repeat_interval => 5);

In the following call you accumulate execution statistics as you go. This option produces an accurate picture of the cumulative activity of each cursor, even across age-outs, but it is more expensive than the previous example.

EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
  sqlset_name     => 'my_workload', -
  time_limit      => 30, -
  repeat_interval => 5, -
  capture_mode    =>
  dbms_sqltune.MODE_ACCUMULATE_STATS);
This call performs a very inexpensive capture where you only insert new statements and do not update their statistics once they have been inserted into the SQL tuning set.

```sql
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
    sqlset_name => 'my_workload', -
    time_limit => 30, -
    repeat_interval => 5, -
    capture_option => 'INSERT');
```

### 160.5.7 CREATE_SQL_PLAN_BASELINE Procedure

This procedure creates a SQL plan baseline for an execution plan. It can be used in the context of an Alternative Plan Finding made by SQL Tuning Advisor.

**See Also:**

- DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

**Syntax**

```sql
DBMS_SQLTUNE.CREATE_SQL_PLAN_BASELINE (  
    task_name            IN VARCHAR2,
    object_id            IN NUMBER   := NULL,
    plan_hash_value      IN NUMBER,
    owner_name           IN VARCHAR2 := NULL,
    database_link_to     IN VARCHAR2 := NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the task for which to get a script</td>
</tr>
<tr>
<td>object_id</td>
<td>Object ID to which the SQL corresponds</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Plan to create plan baseline</td>
</tr>
<tr>
<td>owner_name</td>
<td>Owner of the relevant tuning task. Defaults to the current schema owner.</td>
</tr>
<tr>
<td>database_link_to</td>
<td></td>
</tr>
</tbody>
</table>
Table 160-14 (Cont.) CREATE_SQL_PLAN_BASELINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| database_link_to| Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use `DBMS_SQLTUNE` to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute `REPORT_TUNING_TASK` locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The `database_link_to` parameter must specify a private database link. This link must be owned by `SYS` and accessed by the default privileged user `SYS$UMF`. The following sample statement creates a link named `lnk_to_pri`:

```sql
CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS
$UMF IDENTIFIED BY password USING 'inst1';
```

160.5.8 CREATE_SQLSET Procedure and Function

This procedure or function creates a SQL tuning set object in the database.

**Syntax**

```sql
DBMS_SQLTUNE.CREATE_SQLSET (  
    sqlset_name    IN    VARCHAR2,  
    description    IN    VARCHAR2 := NULL  
    sqlset_owner   IN    VARCHAR2 := NULL);

DBMS_SQLTUNE.CREATE_SQLSET (  
    sqlset_name    IN    VARCHAR2 := NULL,  
    description    IN    VARCHAR2 := NULL,  
    sqlset_owner   IN    VARCHAR2 := NULL)  
RETURN VARCHAR2;
```

**Parameters**

Table 160-15 CREATE_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the created SQL tuning set. The name is the name passed to the function. If no name is passed to the function, then the function generates an automatic name.</td>
</tr>
<tr>
<td>description</td>
<td>Provides an optional description of the SQL tuning set.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for the current schema owner.</td>
</tr>
</tbody>
</table>
Examples

EXEC DBMS_SQLTUNE.CREATE_SQLSET(-
    sqlset_name => 'my_workload', -
    description => 'complete application workload');

160.5.9 CREATE_STGTAB_SQLPROF Procedure

This procedure creates the staging table used for copying SQL profiles from one system to another.

See Also:

DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (  
    table_name            IN VARCHAR2,  
    schema_name           IN VARCHAR2 := NULL,  
    tablespace_name       IN VARCHAR2 := NULL);  

Parameters

Table 160-16  CREATE_STGTAB_SQLPROF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table to create (case-insensitive unless double quoted).</td>
</tr>
<tr>
<td>schema_name</td>
<td>The schema to create the table in, or NULL for the current schema (case-</td>
</tr>
<tr>
<td></td>
<td>insensitive unless double quoted).</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>The tablespace to store the staging table within, or NULL for the default</td>
</tr>
<tr>
<td></td>
<td>tablespace of the current user (case-insensitive unless double quoted).</td>
</tr>
</tbody>
</table>

Usage Notes

- Call this procedure once before issuing a call to the PACK_STGTAB_SQLPROF Procedure.
- To put different SQL profiles in different staging tables, you can call this procedure multiple times.
- This is a DDL operation, so it does not occur within a transaction.

Examples

Create a staging table to store profile data that can be moved to another system.

EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (table_name => 'PROFILE_STGTAB');
160.5.10 CREATE_STGTAB_SQLSET Procedure

This procedure creates a staging table through which SQL tuning sets are imported and exported.

See Also:
DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.CREATE_STGTAB_SQLSET (  
    table_name           IN VARCHAR2,  
    schema_name          IN VARCHAR2 := NULL,  
    tablespace_name      IN VARCHAR2 := NULL,  
    db_version           IN NUMBER   := NULL);

Parameters

Table 160-17  CREATE_STGTAB_SQLSET and CREATE_STGTAB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Specifies the name of the table to create. The name is case sensitive.</td>
</tr>
<tr>
<td>schema_name</td>
<td>Defines the schema in which to create the table, or NULL for the current</td>
</tr>
<tr>
<td></td>
<td>schema. The name is case sensitive.</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Specifies the tablespace in which to store the staging table, or NULL for</td>
</tr>
<tr>
<td></td>
<td>the default tablespace of the current user. The name is case sensitive.</td>
</tr>
<tr>
<td>db_version</td>
<td>Specifies the database version that determines the format of the staging</td>
</tr>
<tr>
<td></td>
<td>table. You can also create an older database version staging table to export</td>
</tr>
<tr>
<td></td>
<td>an STS to an older database version. Use one of the following values:</td>
</tr>
<tr>
<td></td>
<td>•   NULL (default) — Specifies the current database version.</td>
</tr>
<tr>
<td></td>
<td>•   STS_STGTAB_10_2_VERSION — Specifies the 10.2 database version.</td>
</tr>
<tr>
<td></td>
<td>•   STS_STGTAB_11_1_VERSION — Specifies the 11.1 database version.</td>
</tr>
<tr>
<td></td>
<td>•   STS_STGTAB_11_2_VERSION — Specifies the 11.2 database version.</td>
</tr>
<tr>
<td></td>
<td>•   STS_STGTAB_12_1_VERSION — Specifies the 12.1 database version.</td>
</tr>
<tr>
<td></td>
<td>•   STS_STGTAB_12_2_VERSION — Specifies the 12.2 database version.</td>
</tr>
</tbody>
</table>

Security Model

You must have CREATE TABLE permissions in the specified schema and tablespace.

Usage Notes

• Call this procedure once before packing the SQL set.
• To have different tuning sets in different staging tables, you can call this procedure multiple times.
• This is a DDL operation, so it does not occur within a transaction.
• The staging table contains nested table columns and indexes, so it should not be renamed.

Examples
Create a staging table for packing and eventually exporting a SQL tuning sets

EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(table_name => 'STGTAB_SQLSET');

Create a staging table to pack a SQL tuning set in Oracle Database 11g Release 2 (11.2) format

BEGIN
  DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(
    table_name => 'STGTAB_SQLSET',
    db_version => DBMS_SQLTUNE.STS_STGTAB_11_2_VERSION );
END;

160.5.11 CREATE_TUNING_TASK Functions

This function creates a SQL Tuning Advisor task.

You can use different forms of this function to:
• Create a tuning task for a single statement given its text.
• Create a tuning task for a single statement from the shared SQL area given its identifier.
• Create a tuning task for a single statement from the workload repository given a range of snapshot identifiers.
• Create a tuning task for a SQL tuning set.
• Create a tuning task for SQL Performance Analyzer.

In all cases, the function mainly creates a SQL Tuning Advisor task and sets its parameters.

See Also:
DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

SQL text format:

DBMS_SQLTUNE.CREATE_TUNING_TASK ( sql_text IN CLOB,
                                      bind_list IN sql_binds := NULL,
                                      user_name IN VARCHAR2 := NULL,
scope IN VARCHAR2 := SCOPE_COMPREHENSIVE,
time_limit IN NUMBER := TIME_LIMIT_DEFAULT,
task_name IN VARCHAR2 := NULL,
description IN VARCHAR2 := NULL,
con_name IN VARCHAR2 := NULL,
database_link_to IN VARCHAR2 := NULL)
RETURN VARCHAR2;

SQL ID format:

DBMS_SQLTUNE.CREATE_TUNING_TASK (
sql_id IN VARCHAR2,
plan_hash_value IN NUMBER := NULL,
scope IN VARCHAR2 := SCOPE_COMPREHENSIVE,
time_limit IN NUMBER := TIME_LIMIT_DEFAULT,
task_name IN VARCHAR2 := NULL,
description IN VARCHAR2 := NULL,
con_name IN VARCHAR2 := NULL,
database_link_to IN VARCHAR2 := NULL)
RETURN VARCHAR2;

AWR format:

DBMS_SQLTUNE.CREATE_TUNING_TASK (
begin_snap IN NUMBER,
end_snap IN NUMBER,
sql_id IN VARCHAR2,
plan_hash_value IN NUMBER := NULL,
scope IN VARCHAR2 := SCOPE_COMPREHENSIVE,
time_limit IN NUMBER := TIME_LIMIT_DEFAULT,
task_name IN VARCHAR2 := NULL,
description IN VARCHAR2 := NULL,
con_name IN VARCHAR2 := NULL,
dbid IN NUMBER := NULL,
database_link_to IN VARCHAR2 := NULL)
RETURN VARCHAR2;

SQL tuning set format:

DBMS_SQLTUNE.CREATE_TUNING_TASK (
sqset_name IN VARCHAR2,
basic_filter IN VARCHAR2 := NULL,
object_filter IN VARCHAR2 := NULL,
r1 IN VARCHAR2 := NULL,
r2 IN VARCHAR2 := NULL,
r3 IN VARCHAR2 := NULL,
result_percentage IN NUMBER := NULL,
result_limit IN NUMBER := NULL,
scope IN VARCHAR2 := SCOPE_COMPREHENSIVE,
time_limit IN NUMBER := TIME_LIMIT_DEFAULT,
task_name IN VARCHAR2 := NULL,
description IN VARCHAR2 := NULL,
plan_filter IN VARCHAR2 := 'MAX_ELAPSED_TIME',

SQL Performance Analyzer format:

```sql
DBMS_SQLTUNE.CREATE_TUNING_TASK (
    spa_task_name     IN VARCHAR2,
    spa_task_owner    IN VARCHAR2 :=  NULL,
    spa_compare_exec  IN VARCHAR2 :=  NULL,
    basic_filter      IN VARCHAR2 :=  NULL,
    time_limit        IN NUMBER   :=  TIME_LIMIT_DEFAULT,
    task_name         IN VARCHAR2 :=  NULL,
    description       IN VARCHAR2 :=  NULL)
RETURN VARCHAR2;
```

Parameters

Table 160-18 CREATE_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>Specifies the text of a SQL statement.</td>
</tr>
<tr>
<td>begin_snap</td>
<td>Specifies the begin snapshot identifier.</td>
</tr>
<tr>
<td>end_snap</td>
<td>Specifies the end snapshot identifier.</td>
</tr>
<tr>
<td>sql_id</td>
<td>Specifies the identifier of a SQL statement.</td>
</tr>
<tr>
<td>bind_list</td>
<td>Defines an ordered list of bind values in ANYDATA type.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Specifies the hash value of the SQL execution plan.</td>
</tr>
<tr>
<td>sqlset_name</td>
<td>Specifies the SQL tuning set name.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Specifies the predicate used to filter the SQL from the SQL tuning set.</td>
</tr>
<tr>
<td>object_filter</td>
<td>Specifies the object filter.</td>
</tr>
<tr>
<td>rank(i)</td>
<td>Specifies an ORDER BY clause on the selected SQL statement.</td>
</tr>
<tr>
<td>result_percentage</td>
<td>Specifies the percentage on the sum of a ranking measure.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Specifies the top L(imit) SQL from the filtered or ranked SQL.</td>
</tr>
<tr>
<td>user_name</td>
<td>Specifies the user name for whom the statement is to be tuned.</td>
</tr>
<tr>
<td>scope</td>
<td>Specifies the tuning scope:</td>
</tr>
<tr>
<td></td>
<td>• LIMITED: SQL Tuning Advisor produces recommendations based on statistical checks, access path analysis, and SQL structure analysis. SQL profile recommendations are not generated.</td>
</tr>
<tr>
<td></td>
<td>• COMPREHENSIVE: SQL Tuning Advisor carries out all the analysis it performs under limited scope plus SQL profiling.</td>
</tr>
<tr>
<td>time_limit</td>
<td>Specifies the maximum duration in seconds for the tuning session.</td>
</tr>
<tr>
<td>task_name</td>
<td>Specifies an optional tuning task name.</td>
</tr>
<tr>
<td>description</td>
<td>Provides a description of the SQL tuning session, up to a maximum of 256 characters.</td>
</tr>
</tbody>
</table>

NOTE: This parameter is not supported on a standby database.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_filter</td>
<td>Specifies the plan filter. It is applicable when multiple plans (plan_hash_value) are associated with the same statement. This filter allows for selecting one plan (plan_hash_value) only. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• LAST_GENERATED: most recent timestamp</td>
</tr>
<tr>
<td></td>
<td>• FIRST_GENERATED: earliest timestamp, the opposite to LAST_GENERATED</td>
</tr>
<tr>
<td></td>
<td>• LAST_LOADED: most recent first_load_time statistics information</td>
</tr>
<tr>
<td></td>
<td>• FIRST_LOADED: earliest first_load_time statistics information, the opposite to LAST_LOADED</td>
</tr>
<tr>
<td></td>
<td>• MAX_ELAPSED_TIME: maximum elapsed time</td>
</tr>
<tr>
<td></td>
<td>• MAX_BUFFER_GETS: maximum buffer gets</td>
</tr>
<tr>
<td></td>
<td>• MAX_DISK_READS: maximum disk reads</td>
</tr>
<tr>
<td></td>
<td>• MAX_DIRECT_WRITES: maximum direct writes</td>
</tr>
<tr>
<td></td>
<td>• MAX_OPTIMIZER_COST: maximum optimizer cost</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for the current schema owner.</td>
</tr>
<tr>
<td>spa_task_name</td>
<td>Specifies the name of the SQL Performance Analyzer task whose regressions are to be tuned.</td>
</tr>
<tr>
<td>spa_task_owner</td>
<td>Specifies the owner of specified SQL Performance Analyzer task or NULL for current user.</td>
</tr>
<tr>
<td>spa_compare_exec</td>
<td>Specifies the execution name of the Compare Performance trial of SQL Performance Analyzer task. If NULL, then the advisor uses the most recent execution of the given SQL Performance Analyzer task, of type COMPARE PERFORMANCE.</td>
</tr>
<tr>
<td>dbid</td>
<td>Specifies the DBID for imported or PDB-level AWR data. If NULL, then the current database DBID is used.</td>
</tr>
</tbody>
</table>
Table 160-18  (Cont.) CREATE_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>con_name</td>
<td>Specifies the container for the tuning task. The semantics depend on the function format:</td>
</tr>
<tr>
<td></td>
<td>• For the SQL text format, this parameter specifies the container in which SQL Tuning Advisor tunes the SQL statement. If null (default), then SQL Tuning Advisor uses the current container.</td>
</tr>
<tr>
<td></td>
<td>• For the SQL ID format, this parameter specifies the container from which the database fetches the SQL statement for tuning. SQL Tuning Advisor tunes the statement in this container. If null, then the database uses the current PDB for tuning, fetches the statement from the cursor cache of all valid containers executing the SQL statement, and tunes the most expensive statement in its container.</td>
</tr>
<tr>
<td></td>
<td>• For the AWR format, this parameter specifies the container from whose AWR data the database fetches the SQL statement for tuning. SQL Tuning Advisor tunes the statement in this container. If null, then the database uses the current PDB for tuning, fetches the statement from the AWR of all valid containers that have this SQL statement, and tunes the most expensive statement in its container.</td>
</tr>
<tr>
<td></td>
<td>The following statements are true of all function formats:</td>
</tr>
<tr>
<td></td>
<td>• In a non-CDB, this parameter is ignored.</td>
</tr>
<tr>
<td></td>
<td>• In a PDB, this parameter must be null or match the container name of the PDB. Otherwise, an error occurs.</td>
</tr>
<tr>
<td></td>
<td>• In a CDB root, this parameter must be null or match the container name of a container in this CDB. Otherwise, an error occurs.</td>
</tr>
<tr>
<td>database_link_to</td>
<td>The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database.</td>
</tr>
<tr>
<td></td>
<td>The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYSSUMF. The following sample statement creates a link named lnk_to_pri:</td>
</tr>
<tr>
<td></td>
<td>CREATE DATABASE LINK lnk_to_pri CONNECT TO SYSSUMF IDENTIFIED BY password USING 'inst1';</td>
</tr>
</tbody>
</table>

Return Values

A SQL tuning task name that is unique by user (two different users can give the same name to their advisor tasks).

Usage Notes

With regard to the form of this subprogram that takes a SQL tuning set, filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed
with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

**Example 160-1  Examples**

The following examples assume the following variable definitions:

```sql
VARIABLE stmt_task     VARCHAR2(64);
VARIABLE sts_task      VARCHAR2(64);
VARIABLE spa_tune_task VARCHAR2(64);
```

**Example 160-2  Create Tuning Task with SQL Text Format**

```sql
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
    sql_text => 'SELECT quantity_sold FROM sales s, times t WHERE 
s.time_id = t.time_id AND s.time_id = TO_DATE('''24-NOV-00''')');
```

**Example 160-3  Create Tuning Task with SQL ID Format**

```sql
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'aylm3ssvtrh24');
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'aylm3ssvtrh24', -
    scope => 'LIMITED');
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'aylm3ssvtrh24', -
    time_limit => 600);
```

**Example 160-4  Create Tuning Task with AWR Snapshot Format**

```sql
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(begin_snap => 1, -
    end_snap => 2, sql_id => 'aylm3ssvtrh24');
```

**Example 160-5  Create Tuning Task with SQL Tuning Set Format**

This example creates a task that tunes SQL statements in order by buffer gets, and also sets a time limit of one hour. The default ranking measure is elapsed time.

```sql
EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
    sqlset_name  => 'my_workload', -
    rank1        => 'BUFFER_GETS', -
    time_limit   => 3600, -
    description  => 'tune my workload ordered by buffer gets');
```
Example 160-6  Create Tuning Task with SPA Task Format

This example tunes the SQL statement that were reported as having regressed from the compare performance execution of the SQL Performance Analyzer task named task_123.

EXEC :spa_tune_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(
    spa_task_name    => 'task_123',
    spa_task_owner   => 'SCOTT',
    spa_compare_exec => 'exec1');

Example 160-7  Creating SQL Tuning Task on Standby Database

This example creates a tuning task on the standby database. The tune_stby_wkld task uses the lnk_to_primary database link to write data to the primary database, which is open read/write.

VAR tname VARCHAR2(30);
VAR query VARCHAR2(500);
EXEC :tname := 'tune_stby_wkld';
EXEC :query := 'SELECT /*+ FULL(t)*/ col1 FROM table1 t WHERE col1=9000';
EXEC :tname := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_text => :query, -
    task_name => :tname, database_link_to => 'lnk_to_primary');

160.5.12 DELETE_SQLSET Procedure

This procedure deletes a set of SQL statements from a SQL tuning set.

Syntax

DBMS_SQLTUNE.DELETE_SQLSET (  
    sqlset_name IN  VARCHAR2,  
    basic_filter IN  VARCHAR2 := NULL,  
    sqlset_owner IN  VARCHAR2 := NULL);

Parameters

Table 160-19  DELETE_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate to filter the SQL from the SQL tuning set. This basic filter is used as a where clause on the SQL tuning set content to select a desired subset of SQL from the SQL tuning set.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for current schema owner.</td>
</tr>
</tbody>
</table>

Examples

-- Delete all statements in a sql tuning set.
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name => 'my_workload');
-- Delete all statements in a sql tuning set which ran for less than a second
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name => 'my_workload', -
    basic_filter => 'elapsed_time <
    1000000');

160.5.13 DROP_SQL_PROFILE Procedure

This procedure drops the named SQL profile from the database.

See Also:
DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.DROP_SQL_PROFILE (  
  name          IN  VARCHAR2,
  ignore        IN  BOOLEAN  := FALSE);

Parameters

Table 160-20  DROP_SQL_PROFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The (mandatory) name of SQL profile to be dropped. The name is case sensitive.</td>
</tr>
<tr>
<td>ignore</td>
<td>Ignores errors due to object not existing</td>
</tr>
</tbody>
</table>

Usage Notes

Requires the DROP ANY SQL PROFILE privilege.

Examples

-- Drop the profile:
EXEC DBMS_SQLTUNE.DROP_SQL_PROFILE(:pname);
160.5.14 DROP_SQLSET Procedure

This procedure drops a SQL tuning set if it is not active.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DROP_SQLSET (  
    sqlset_name  IN  VARCHAR2,  
    sqlset_owner IN  VARCHAR2 := NULL);
```

Parameters

Table 160-21  DROP_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for current schema owner.</td>
</tr>
</tbody>
</table>

Usage Notes

You cannot drop a SQL tuning set when it is referenced by one or more clients.

Examples

```
-- Drop the sqlset.
EXEC DBMS_SQLTUNE.DROP_SQLSET ('my_workload');
```

160.5.15 DROP_TUNING_TASK Procedure

This procedure drops a SQL tuning task. The task and all its result data are deleted.

See Also:

DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group
Syntax

DBMS_SQLTUNE.DROP_TUNING_TASK (
    task_name      IN VARCHAR2);

Parameters

Table 160-22  DROP_TUNING_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Specifies name of the tuning task to drop.</td>
</tr>
</tbody>
</table>

160.5.16 EXECUTE_TUNING_TASK Function and Procedure

This function and procedure executes a previously created tuning task. Both the function and the procedure run in the context of a new task execution. The difference is that the function version returns that new execution name.

See Also:

DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
    task_name      IN VARCHAR2,
    execution_name IN VARCHAR2              := NULL,
    execution_params IN dbms_advisor.argList := NULL,
    execution_desc  IN VARCHAR2              := NULL,
    database_link_to IN VARCHAR2              := NULL)
RETURN VARCHAR2;

DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
    task_name      IN VARCHAR2,
    execution_name IN VARCHAR2              := NULL,
    execution_params IN dbms_advisor.argList := NULL,
    execution_desc  IN VARCHAR2              := NULL,
    database_link_to IN VARCHAR2              := NULL);

Parameters

Table 160-23  EXECUTE_TUNING_TASK Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the tuning task to execute.</td>
</tr>
<tr>
<td>execution_name</td>
<td>A name to qualify and identify an execution. If not specified, it is generated by the advisor and returned by function.</td>
</tr>
</tbody>
</table>
Table 160-23  (Cont.) EXECUTE_TUNING_TASK Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execution_params</td>
<td>List of parameters (name, value) for the specified execution. The execution parameters have effect only on the execution for which they are specified. They override the values for the parameters stored in the task (set through the SET_TUNING_TASK_PARAMETER Procedures).</td>
</tr>
<tr>
<td>execution_desc</td>
<td>A 256-length string describing the execution.</td>
</tr>
</tbody>
</table>
| database_link_to | Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMP. The following sample statement creates a link named lnk_to_pri:

```sql
CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMP IDENTIFIED BY password USING 'inst1';
```

Usage Notes
A tuning task can be executed multiples times without having to reset it.

Examples
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);

160.5.17 IMPLEMENT_TUNING_TASK Procedure

This procedure implements a set of SQL profile recommendations made by SQL Tuning Advisor.

Executing IMPLEMENT_TUNING_TASK is equivalent to executing the SCRIPT_TUNING_TASK Function and then running the script.

See Also:
DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group
Syntax

```
DBMS_SQLTUNE.IMPLEMENT_TUNING_TASK(
    task_name         IN  VARCHAR2,
    rec_type          IN  VARCHAR2 := REC_TYPE_SQL_PROFILES,
    owner_name        IN  VARCHAR2 := NULL,
    execution_name    IN  VARCHAR2 := NULL,
    database_link_to  IN  VARCHAR2 := NULL);
```

Parameters

**Table 160-24 IMPLEMENT_TUNING_TASK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the tuning task for which to implement recommendations.</td>
</tr>
<tr>
<td>rec_type</td>
<td>Filter the types of recommendations to implement. Only 'PROFILES' is supported.</td>
</tr>
<tr>
<td>owner_name</td>
<td>Owner of the relevant tuning task or NULL for the current user.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name of the task execution to use. If NULL, then the procedure implements recommendations from the last task execution.</td>
</tr>
<tr>
<td>database_link_to</td>
<td>Name of a database link that exists on a standby database.</td>
</tr>
</tbody>
</table>

The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local.

**Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database.**

The `database_link_to` parameter must specify a private database link. This link must be owned by `SYS` and accessed by the default privileged user `SYSSUMP`. The following sample statement creates a link named `lnk_to_pri`:

```
CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';
```
160.5.18 INTERRUPT_TUNING_TASK Procedure

This procedure interrupts the currently executing tuning task. The task ends its operations as it would at normal exit so that the user can access the intermediate results.

See Also:

DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.INTERRUPT_TUNING_TASK (  
task_name         IN VARCHAR2);  

Parameters

Table 160-25  INTERRUPT_TUNING_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the tuning task to interrupt</td>
</tr>
</tbody>
</table>

Examples

EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:my_task);

160.5.19 LOAD_SQLSET Procedure

This procedure populates the SQL tuning set with a set of selected SQL statements. You can call the procedure multiple times to add new SQL statements or replace attributes of existing statements.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.LOAD_SQLSET (  
sqlset_name       IN VARCHAR2,  
populate_cursor   IN sqlset_cursor,  
load_option       IN VARCHAR2 := 'INSERT',  
update_option     IN VARCHAR2 := 'REPLACE',  
update_condition  IN VARCHAR2 := NULL,  
)
Parameters

Table 160-26  LOAD_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of SQL tuning set to be loaded.</td>
</tr>
<tr>
<td>populate_cursor</td>
<td>Specifies the cursor reference to the SQL tuning set to be loaded.</td>
</tr>
</tbody>
</table>
| load_option    | Specifies which statements are loaded into the SQL tuning set. The possible values are:  
  • INSERT (default) — Adds only new statements.  
  • UPDATE — Updates existing the SQL statements and ignores any new statements.  
  • MERGE — Inserts new statements and updates the information of the existing ones.  
| update_option  | Specifies how existing SQL statements are updated.                         |
| update_condition| Specifies when to perform the update.                                      |

This parameter is considered only if load_option is specified with UPDATE or MERGE as an option. The possible values are:  
  • REPLACE (default) — Updates the statement using the new statistics, bind list, object list, and so on.  
  • ACCUMULATE — Combines attributes when possible (for example, statistics such as elapsed_time), and otherwise replaces the existing values (for example, module and action) with the provided values. The SQL statement attributes that can be accumulated are: elapsed_time, buffer_gets, direct_writes, disk_reads, row_processed, fetches, executions, end_of_fetch_count, stat_period and active_stat_period.  

The procedure only performs the update when the specified condition is satisfied. The condition can refer to either the data source or destination. The condition must use the following prefixes to refer to attributes from the source or the destination:  
  • OLD — Refers to statement attributes from the SQL tuning set (destination).  
  • NEW — Refers to statement attributes from the input statements (source).
Table 160-26  (Cont.) LOAD_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| update_attributes | Specifies the list of SQL statement attributes to update during a merge or update.  
  The possible values are:  
  • NULL (default) — Specifies the content of the input cursor except the execution context. On other terms, it is equivalent to ALL without execution contexts such as module and action.  
  • BASIC — Specifies statistics and binds only.  
  • TYPICAL — Specifies BASIC with SQL plans (without row source statistics) and without an object reference list.  
  • ALL — Specifies all attributes, including the execution context attributes such as module and action.  
  • List of comma separated attribute names to update:  
    – EXECUTION_CONTEXT  
    – EXECUTION_STATISTICS  
    – SQL_BINDS  
    – SQL_PLAN  
    – SQL_PLAN_STATISTICS (similar to SQL_PLAN with added row source statistics) |
| ignore_null     | Specifies whether to update attributes when the new value is NULL.  
  If TRUE, then the procedure does not update an attribute when the new value is NULL. That is, do not override with NULL values unless intentional. |
| commit_rows     | Specifies whether to commit statements after DML.  
  If a value is provided, then the load commits after each specified number of statements is inserted. If NULL is provided, then the load commits only once, at the end of the operation.  
  Providing a value for this argument enables you to monitor the progress of a SQL tuning set load operation in the DBA_SQLSET views. The STATEMENT_COUNT value increases as new SQL statements are loaded. |
| sqlset_owner    | Defines the owner of the SQL tuning set, or the current schema owner (or NULL for the current owner). |

Exceptions

- This procedure returns an error when sqlset_name is invalid, or a corresponding SQL tuning set does not exist, or the populate_cursor is incorrect and cannot be executed.
- Exceptions are also raised when invalid filters are provided. Filters can be invalid either because they don't parse (for example, they refer to attributes not in sqlset_row), or because they violate the user's privileges.

Usage Notes

Rows in the input populate_cursor must be of type SQLSET_ROW.
Examples

In this example, you create and populate a SQL tuning set with all shared SQL area
statements with an elapsed time of 5 seconds or more excluding statements that
belong to SYS schema (to simulate an application user workload). You select all
attributes of the SQL statements and load them in the tuning set using the default
mode, which loads only new statements, since the SQL tuning set is empty.

```sql
-- create the tuning set
EXEC DBMS_SQLTUNE.CREATE_SQLSET('my_workload');
-- populate the tuning set from the shared SQL area
DECLARE
  cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
        'parsing_schema_name <> ''SYS'' AND elapsed_time > 5000000',
        NULL, NULL, NULL, 1, NULL,
        'ALL')) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
  populate_cursor => cur);
END;
/
```

Suppose now you wish to augment this information with what is stored in the workload
repository (AWR). You populate the tuning set with 'ACCUMULATE' as your
update_option because it is assumed the cursors currently in the cache had aged out
since the snapshot was taken.

You omit the elapsed_time filter because it is assumed that any statement captured in
AWR is important, but still you throw away the SYS-parsed cursors to avoid recursive
SQL.

```sql
DECLARE
  cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2,
        'parsing_schema_name <> ''SYS''',
        NULL, NULL, NULL, NULL, 1, NULL,
        'ALL')) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
  populate_cursor => cur,
  Using DBMS_SQLTUNE
  load_option => 'MERGE',
/)```
The following example is a simple load that only inserts new statements from the workload repository, skipping existing ones (in the SQL tuning set). Note that 'INSERT' is the default value for the load_option argument of the LOAD_SQLSET procedure.

DECLARE
   cur sys_refcursor;
BEGIN
   OPEN cur FOR
   SELECT VALUE(P)
   FROM table(DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2)) P;
   DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload', populate_cursor => cur);
END;
/

The next example demonstrates a load with UPDATE option. This updates statements that already exist in the SQL tuning set but does not add new ones. By default, old statistics are replaced by their new values.

DECLARE
   cur sys_refcursor;
BEGIN
   OPEN cur FOR
   SELECT VALUE(P)
   FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;
   DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
                             populate_cursor => cur,
                             load_option => 'UPDATE');
END;
/

160.5.20 PACK_STGTAB_SQLPROF Procedure

This procedure copies profile data from the SYS. schema into the staging table.

See Also:

DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (  
   profile_name         IN VARCHAR2 := '%',
   profile_category     IN VARCHAR2 := 'DEFAULT',
)


```sql
staging_table_name    IN VARCHAR2,
staging_schema_owner  IN VARCHAR2 := NULL);
```

**Parameters**

**Table 160-27  PACK_STGTAB_SQLPROF Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>The name of the profile to pack (% wildcards acceptable, case-sensitive)</td>
</tr>
<tr>
<td>profile_category</td>
<td>The category to pack profiles from (% wildcards acceptable, case-sensitive)</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>The name of the table to use (case-insensitive unless double quoted).</td>
</tr>
<tr>
<td>staging_schema_owner</td>
<td>The schema where the table resides, or NULL for current schema (case-</td>
</tr>
<tr>
<td></td>
<td>insensitive unless double quoted)</td>
</tr>
</tbody>
</table>

**Security Model**

This procedures requires `ADMINISTER SQL MANAGEMENT OBJECT` privilege and `INSERT` privilege on the staging table.

**Usage Notes**

This function issues a `COMMIT` after packing each SQL profile. If an error is raised mid-execution, then clear the staging table by deleting its rows.

**Examples**

Put only those profiles in the `DEFAULT` category into the staging table. This corresponds to all profiles used by default on this system.

```sql
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (staging_table_name =>
    'PROFILE_STGTAB');
```

This is another example where you put all profiles into the staging table. Note this moves profiles that are not currently being used by default but are in other categories, such as for testing purposes.

```sql
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (profile_category => '%',
    staging_table_name =>
    'PROFILE_STGTAB');
```
160.5.21 PACK_STGTAB_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the SYS schema to a staging table created by the CREATE_STGTAB_SQLSET procedure.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.PACK_STGTAB_SQLSET (
    sqlset_name          IN VARCHAR2,
    sqlset_owner         IN VARCHAR2 := NULL,
    staging_table_name   IN VARCHAR2,
    staging_schema_owner IN VARCHAR2 := NULL,
    db_version           IN NUMBER   := NULL);
```

Examples

Put all SQL tuning sets on the database in the staging table:

```
BEGIN
    DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
        sqlset_name => '%',
        sqlset_owner => '%',
        staging_table_name => 'STGTAB_SQLSET');
END;
```

Put only those SQL tuning sets owned by the current user in the staging table:

```
BEGIN
    DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
        sqlset_name => '%',
        staging_table_name => 'STGTAB_SQLSET');
END;
```

Pack a specific SQL tuning set:

```
BEGIN
    DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
        sqlset_name => 'my_workload',
        staging_table_name => 'STGTAB_SQLSET');
END;
```
Pack a second SQL tuning set:

```
BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
    sqlset_name => 'workload_subset',
    staging_table_name => 'STGTAB_SQLSET');
END;
```

Pack the STS `my_workload_subset` into a staging table `stgtab_sqlset` created for Oracle Database 11g Release 1 (11.2):

```
BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
    sqlset_name          => 'workload_subset',
    staging_table_name   => 'STGTAB_SQLSET',
    db_version           => DBMS_SQLTUNE.STS_STGTAB_11_2_VERSION);
END;
```

### 160.5.22 REMAP_STGTAB_SQLPROF Procedure

This procedure changes the profile data values kept in the staging table prior to performing an unpack operation.

You can use this procedure to change the category of a profile. You can also use it to change the name of a profile if one already exists on the system with the same name.

See Also:

- [DBMS_SQLTUNE SQL Profile Subprograms](#) for other subprograms in this group

**Syntax**

```
DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF ( 
  old_profile_name      IN VARCHAR2, 
  new_profile_name      IN VARCHAR2 := NULL, 
  new_profile_category  IN VARCHAR2 := NULL, 
  staging_table_name    IN VARCHAR2, 
  staging_schema_owner  IN VARCHAR2 := NULL);
```

**Parameters**

**Table 160-28   REMAP_STGTAB_SQLPROF Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_profile_name</td>
<td>The name of the profile to target for a remap operation (case-sensitive)</td>
</tr>
</tbody>
</table>
Table 160-28 (Cont.) REMAP_STGTAB_SQLPROF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_profile_name</td>
<td>The new name of the profile, or NULL to remain the same (case-sensitive)</td>
</tr>
<tr>
<td>new_profile_category</td>
<td>The new category for the profile, or NULL to remain the same (case-sensitive)</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>The name of the table on which to perform the remap operation (case-sensitive)</td>
</tr>
<tr>
<td>staging_schema_owner</td>
<td>The schema where the table resides, or NULL for current schema (case-sensitive)</td>
</tr>
</tbody>
</table>

Security Model

This procedure requires the UPDATE privilege on the staging table.

Examples

Change the name of a profile before we unpack, to avoid conflicts

```
BEGIN
    DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(
        old_profile_name    => :pname,
        new_profile_name    => 'IMP' || :pname,
        staging_table_name  => 'PROFILE_STGTAB');
END;
```

Change the SQL profile in the staging table to be 'TEST' category before we import it. This way users can test the profile on the new system before it is active.

```
BEGIN
    DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(
        old_profile_name     => :pname,
        new_profile_category => 'TEST',
        staging_table_name   => 'PROFILE_STGTAB');
END;
```

160.5.23 REMAP_STGTAB_SQLSET Procedure

This procedure changes the tuning set names and owners in the staging table so that they can be unpacked with different values.

See Also:

DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group
Syntax

```sql
DBMS_SQLTUNE.REMAP_STGTAB_SQLSET (    old_sqlset_name        IN VARCHAR2,
  old_sqlset_owner       IN VARCHAR2 := NULL,
  new_sqlset_name        IN VARCHAR2 := NULL,
  new_sqlset_owner       IN VARCHAR2 := NULL,
  staging_table_name     IN VARCHAR2,
  staging_schema_owner   IN VARCHAR2 := NULL,
  old_con_dbid           IN NUMBER   := NULL,
  new_con_dbid           IN NUMBER   := NULL);
```

Parameters

The parameters are identical for the DBMS_SQLTUNE.REMAP_STGTAB_SQLSET and DBMS_SQLSET.REMAP_SQLSET procedures.

Table 160-29 REMAP_STGTAB_SQLSET and REMAP_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_sqlset_name</td>
<td>Specifies the name of the tuning set to target for a remap operation. Wildcard characters (%) are not supported.</td>
</tr>
<tr>
<td>old_sqlset_owner</td>
<td>Specifies the new name of the tuning set owner to target for a remap operation. NULL for current schema owner</td>
</tr>
<tr>
<td>new_sqlset_name</td>
<td>Specifies the new name for the tuning set, or NULL to keep the same tuning set name.</td>
</tr>
<tr>
<td>new_sqlset_owner</td>
<td>Specifies the new owner for the tuning set, or NULL to keep the same owner name.</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>Specifies the name of the table on which to perform the remap operation. The value is case sensitive.</td>
</tr>
<tr>
<td>staging_schema_owner</td>
<td>Specifies the name of staging table owner, or NULL for the current schema owner. The value is case sensitive.</td>
</tr>
<tr>
<td>old_con_dbid</td>
<td>Specifies the old container DBID to be remapped to a new container DBID. Specify NULL to use the same container DBID. You must provide both old_con_dbid and new_con_dbid for the remap to succeed.</td>
</tr>
<tr>
<td>new_con_dbid</td>
<td>Specifies the new container DBID to replace with the old container DBID. Specify NULL to use the same container DBID. You must provide both old_con_dbid and new_con_dbid for the remap to succeed.</td>
</tr>
</tbody>
</table>

Usage Notes

Call this procedure multiple times to remap more than one tuning set name or owner. This procedure only handles one tuning set per call.

Examples

```sql
-- Change the name of an STS in the staging table before unpacking it.
BEGIN
```


DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(
    old_sqlset_name => 'my_workload',
    old_sqlset_owner => 'SH',
    new_sqlset_name => 'imp_workload',
    staging_table_name => 'STGTAB_SQLSET');

-- Change the owner of an STS in the staging table before unpacking it.
DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(
    old_sqlset_name => 'imp_workload',
    old_sqlset_owner => 'SH',
    new_sqlset_owner => 'SYS',
    staging_table_name => 'STGTAB_SQLSET');

END;

160.5.24 REMOVE_SQLSET_REFERENCE Procedure

This procedure deactivates a SQL tuning set to indicate that it is no longer used by the client.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE (  
sqlset_name IN VARCHAR2,  
reference_id IN NUMBER,  
sqlset_owner IN VARCHAR2 := NULL,  
force_remove IN NUMBER := 0);

Parameters

The parameters are identical for the DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE and DBMS_SQLSET.REMOVE_REFERENCE procedures.

Table 160-30  REMOVE_SQLSET_REFERENCE and REMOVE_REFERENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>reference_id</td>
<td>Specifies the identifier of the reference to remove.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the owner of the SQL tuning set (or NULL for the current schema owner).</td>
</tr>
</tbody>
</table>
Table 160-30  (Cont.) REMOVE_SQLSET_REFERENCE and REMOVE_REFERENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force_remove</td>
<td>Specifies whether references can be removed for other users (1) or whether they cannot be removed (0). Setting this parameter to 1 only takes effect when the user has the ADMINISTER ANY SQL TUNING SET privilege. Otherwise, the database only removes references owned by the user.</td>
</tr>
</tbody>
</table>

Examples

You can remove references on a given SQL tuning set when you finish using it and want to make it writable again. The following example removes the reference to my_workload:

```sql
EXEC DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE( -
    sqlset_name   => 'my_workload', -
    reference_id  => :rid,
    sqlset_owner  => NULL,
    force_remove  => 0);
```

To find all references to a given SQL tuning set, query the `DBA_SQLSET_REFERENCES` view.

160.5.25 REPORT_AUTO_TUNING_TASK Function

This function displays a report from the automatic tuning task.

This function reports on a range of task executions, whereas the `REPORT_TUNING_TASK Function` reports on a single execution. Note that this function is deprecated with Oracle Database 11g Release 2 (11.2) in favor of `DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK`.

See Also:

- DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group
- REPORT_AUTO_TUNING_TASK Function

Syntax

```sql
DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK( -
    begin_exec   IN VARCHAR2 := NULL,
    end_exec     IN VARCHAR2 := NULL,
    type         IN VARCHAR2 := TYPE_TEXT,
    level        IN VARCHAR2 := LEVEL_TYPICAL,
    section      IN VARCHAR2 := SECTION_ALL,
```
object_id     IN NUMBER    := NULL,
result_limit IN NUMBER    := NULL)
RETURN CLOB;

Parameters

Table 160-31   REPORT_AUTO_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin_exec</td>
<td>Specifies the name of the execution from which to begin the report. NULL retrieves a report on the most recent run.</td>
</tr>
<tr>
<td>end_exec</td>
<td>Specifies the name of the execution at which to end the report. NULL retrieves a report on the most recent run.</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the type of the report to produce. Possible values are TYPE_TEXT which produces a text report</td>
</tr>
<tr>
<td>level</td>
<td>Specifies the level of detail in the report:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>section</td>
<td>Limits the report to a single section (ALL for all sections):</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>object_id</td>
<td>Specifies the advisor framework object ID that represents a single statement to restrict reporting to. Specify NULL for all statements. Only valid for reports that target a single execution.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Specifies the maximum number of SQL statements to show in the report.</td>
</tr>
</tbody>
</table>

Return Values

A CLOB containing the desired report.

160.5.26 REPORT_SQL_DETAIL Function

This function builds a report for a specific SQLID. For each SQLID it gives various statistics and details as obtained from the V$ views and AWR.

See Also:

DBMS_SQLTUNE SQL Performance Reporting Subprograms for other subprograms in this group
Syntax

```sql
DBMS_SQLTUNE.REPORT_SQL_DETAIL (
    sql_id                   IN  VARCHAR2   DEFAULT NULL,
    sql_plan_hash_value      IN  NUMBER     DEFAULT NULL,
    start_time               IN  DATE       DEFAULT NULL,
    duration                 IN  NUMBER     DEFAULT NULL,
    inst_id                  IN  NUMBER     DEFAULT NULL,
    dbid                     IN  NUMBER     DEFAULT NULL,
    event_detail             IN  VARCHAR2   DEFAULT 'YES',
    bucket_max_count         IN  NUMBER     DEFAULT 128,
    bucket_interval          IN  NUMBER     DEFAULT NULL,
    top_n                    IN  NUMBER     DEFAULT 10,
    report_level             IN  VARCHAR2   DEFAULT 'TYPICAL',
    type                     IN  VARCHAR2   DEFAULT 'ACTIVE',
    data_source              IN  VARCHAR2   DEFAULT 'AUTO',
    end_time                 IN  DATE       DEFAULT NULL,
    duration_stats           IN  NUMBER     DEFAULT NULL,
    con_name                 IN  VARCHAR2   DEFAULT NULL
) RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQLID for which monitoring information should be displayed. If NULL (the default), display statistics for the SQLID of the last SQL statement executed in the current session.</td>
</tr>
<tr>
<td>sql_plan_hash_value</td>
<td>Displays SQL statistics and details for a specific plan_hash_value. If NULL (default), displays statistics and details for all plans of the SQL_ID.</td>
</tr>
<tr>
<td>start_time</td>
<td>If specified, shows SQL activity (from GV$ACTIVE_SESSION_HISTORY) starting at this time. On Oracle RAC, the minimum start_time is the earliest sample_time of the in-memory ASH buffers across all instances. If NULL (default), one hour before the current time.</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of activity in seconds for the report. If NULL (default) uses a value of 1 hour.</td>
</tr>
<tr>
<td>inst_id</td>
<td>Target instance to get SQL details from. If NULL, uses data from all instances. If 0 or -1, uses current instance.</td>
</tr>
<tr>
<td>dbid</td>
<td>DBID from which to get SQL details. If NULL, uses current DBID.</td>
</tr>
<tr>
<td>event_detail</td>
<td>When set to 'NO', the activity is aggregated by wait_class only. Use 'YES' (the default) to aggregate by (wait_class, event_name).</td>
</tr>
<tr>
<td>bucket_max_count</td>
<td>If specified, this should be the maximum number of histogram buckets created in the report. If not specified, a value of 128 is used.</td>
</tr>
<tr>
<td>bucket_interval</td>
<td>If specified, this represents the exact time interval in seconds, of all histogram buckets. If specified, bucket_max_count is ignored.</td>
</tr>
<tr>
<td>top_n</td>
<td>Controls the number of entries to display per dimension in the top dimensions section. If not specified, a default value of 10 is used.</td>
</tr>
</tbody>
</table>
Table 160-32  (Cont.) REPORT_SQL_DETAIL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report_level</td>
<td>Level of detail for the report, either 'BASIC', 'TYPICAL' or 'ALL'. Default assumes 'TYPICAL'. Their meanings are explained below.</td>
</tr>
<tr>
<td></td>
<td>In addition, individual report sections can also be enabled or disabled by using a +/- section_name. Several sections are defined:</td>
</tr>
<tr>
<td></td>
<td>• 'TOP' - Show top values for the ASH dimensions for a SQL statement; ON by default</td>
</tr>
<tr>
<td></td>
<td>• 'SPM' - Show existing plan baselines for a SQL statement; OFF by default</td>
</tr>
<tr>
<td></td>
<td>• 'MISMATCH' - Show reasons for creating new child cursors (sharing criteria violations); OFF by default</td>
</tr>
<tr>
<td></td>
<td>• 'STATS' - Show SQL execution statistics per plan from GV$SQLAREA_PLAN_HASH; ON by default</td>
</tr>
<tr>
<td></td>
<td>• 'ACTIVITY' - Show top activity from ASH for each plan of a SQL statement; ON by default</td>
</tr>
<tr>
<td></td>
<td>• 'ACTIVITY_ALL' - Show top activity from ASH for each line of the plan for a SQL statement; OFF by default</td>
</tr>
<tr>
<td></td>
<td>• 'HISTOGRAM' - Show activity histogram for each plan of a SQL statement (plan time line histogram); ON by default</td>
</tr>
<tr>
<td></td>
<td>• 'SESSIONS' - Show activity for top sessions for each plan of a SQL statement; OFF by default</td>
</tr>
<tr>
<td></td>
<td>• 'MONITOR' - Show show one monitored SQL execution per execution plan; ON by default</td>
</tr>
<tr>
<td></td>
<td>• 'XPLAN' - Show execution plans; ON by default</td>
</tr>
<tr>
<td></td>
<td>• 'BINDS' - Show captured bind data; ON by default</td>
</tr>
<tr>
<td></td>
<td>In addition, SQL text can be specified at different levels:</td>
</tr>
<tr>
<td></td>
<td>• -SQL_TEXT - No SQL text in report</td>
</tr>
<tr>
<td></td>
<td>• +SQL_TEXT - OK with partial SQL text up to the first 2000 chars as stored in GV$SQL_MONITOR</td>
</tr>
<tr>
<td></td>
<td>• -SQL_FULLTEXT - No full SQL text (+SQL_TEXT)</td>
</tr>
<tr>
<td></td>
<td>• +SQL_FULLTEXT - Show full SQL text (default value)</td>
</tr>
<tr>
<td></td>
<td>The meanings of the three top-level report levels are:</td>
</tr>
<tr>
<td></td>
<td>• NONE - minimum possible</td>
</tr>
<tr>
<td></td>
<td>• BASIC - SQL_TEXT+STATS+ACTIVITY+HISTOGRAM</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL - SQL_FULLTEXT+TOP+STATS+ACTIVITY+HISTOGRAM+XPLAN+MONITOR</td>
</tr>
<tr>
<td></td>
<td>• ALL - everything</td>
</tr>
<tr>
<td></td>
<td>Only one of these 4 levels can be specified and, if it is, it has to be at the start of the REPORT_LEVEL string</td>
</tr>
<tr>
<td>type</td>
<td>Report format: 'ACTIVE' by default. Can also be 'XML' (see Usage Notes).</td>
</tr>
<tr>
<td>data_source</td>
<td>Determines the data source of SQL data based on one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• MEMORY: The data source is GV$ view</td>
</tr>
<tr>
<td></td>
<td>• DISK: The data source is DBA_HIST_* view</td>
</tr>
<tr>
<td></td>
<td>• AUTO: Automatically determines the data source based on the time frame (default)</td>
</tr>
</tbody>
</table>
Table 160-32   (Cont.) REPORT_SQL_DETAIL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end_time</td>
<td>If specified, shows SQL activity from start_time to end_time. If NULL(default), shows SQL activity for systimestamp.</td>
</tr>
<tr>
<td>duration_stats</td>
<td>Duration of additional SQL execution statistics from AWR (in hours), for the report. If NULL (default), then the duration of 24 hours is considered.</td>
</tr>
<tr>
<td>con_name</td>
<td>Name of the multitenant container database (CDB).</td>
</tr>
</tbody>
</table>

Security Model

The invoker needs the EXECUTE privilege on the DBMS_XPLAN package.

Return Values

A CLOB containing the desired report.

Usage Notes

- ACTIVE reports have a rich, interactive user interface similar to Enterprise Manager while not requiring any EM installation. The report file built is in HTML format, so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.
- The invoker needs the SELECT or READ privilege on the following views:
  - V$SESSION
  - DBA_ADVISOR_FINDINGS
  - V$DATABASE
  - GV$ASH_INFO
  - GV$ACTIVE_SESSION_HISTORY
  - GV$SQLAREA_PLAN_HASH
  - GV$SQL
  - DBA_HIST_SNAPSHOT
  - DBA_HIST_WR_CONTROL
  - DBA_HIST_ACTIVE_SESS_HISTORY
  - DBA_HIST_SQLSTAT
  - DBA_HIST_SQL_BIND_METADATA
  - DBA_HIST_SQLTEXT
  - DBA_SQL_PLAN_BASELINES
  - DBA_SQL_PROFILES
  - DBA_ADVISOR_TASKS
  - DBA_SERVICES
160.5.27 REPORT_SQL_MONITOR Function

This function builds a report (text, simple HTML, active HTML, XML) for the monitoring information collected on behalf of the targeted statement execution.

See Also:

Real-Time SQL Monitoring for other subprograms in this group

Syntax

```sql
DBMS_SQLTUNE.REPORT_SQL_MONITOR(
  sql_id                    IN VARCHAR2  DEFAULT  NULL,
  dbop_name                 IN VARCHAR2  DEFAULT  NULL,
  dbop_exec_id              IN NUMBER    DEFAULT  NULL,
  session_id                IN NUMBER    DEFAULT  NULL,
  session_serial            IN NUMBER    DEFAULT  NULL,
  sql_exec_start            IN DATE      DEFAULT  NULL,
  sql_exec_id               IN NUMBER    DEFAULT  NULL,
  inst_id                   IN NUMBER    DEFAULT  NULL,
  start_time_filter         IN DATE      DEFAULT  NULL,
  end_time_filter           IN DATE      DEFAULT  NULL,
  instance_id_filter        IN NUMBER    DEFAULT  NULL,
  parallel_filter           IN VARCHAR2  DEFAULT  'YES',
  plan_line_filter          IN NUMBER    DEFAULT  NULL,
  event_detail              IN VARCHAR2  DEFAULT  NULL,
  bucket_max_count          IN NUMBER    DEFAULT  128,
  bucket_interval           IN NUMBER    DEFAULT  NULL,
  base_path                 IN VARCHAR2  DEFAULT  NULL,
  last_refresh_time         IN DATE      DEFAULT  NULL,
  report_level              IN VARCHAR2  DEFAULT  'TYPICAL',
  type                      IN VARCHAR2  DEFAULT  'TEXT',
  sql_plan_hash_value       IN NUMBER    DEFAULT  NULL,
  con_name                  IN VARCHAR2  DEFAULT  NULL,
  report_id                 IN NUMBER    DEFAULT  NULL)
RETURN CLOB;
```

Parameters

Table 160-33 REPORT_SQL_MONITOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQL_ID for which monitoring information should be displayed. Use NULL (the default) to report on the last statement monitored by Oracle.</td>
</tr>
</tbody>
</table>
### Table 160-33  (Cont.) REPORT_SQL_MONITOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbop_name</td>
<td>DBOP_NAME for which monitoring information of the composite database operation is displayed.</td>
</tr>
<tr>
<td>dbop_exec_id</td>
<td>Execution ID for the composite database operation for which monitoring information is displayed.</td>
</tr>
<tr>
<td>session_id</td>
<td>If not NULL, this parameter targets only the sub-set of statements executed by the specified session. Default is NULL. Use USERENV('SID') for current session.</td>
</tr>
<tr>
<td>session_serial</td>
<td>In addition to the session_id parameter, one can also specify its session serial to ensure that the desired session incarnation is targeted. This parameter is ignored when session_id is NULL.</td>
</tr>
<tr>
<td>sql_exec_start</td>
<td>This parameter, along with sql_exec_id, is only applicable when sql_id is also specified. Jointly, they can be used to display monitoring information associated to any execution of the statement identified by sql_id, assuming that this statement was monitored. When NULL (the default), the last monitored execution of SQL sql_id is shown.</td>
</tr>
<tr>
<td>sql_exec_id</td>
<td>This parameter, along with sql_exec_start, is only applicable when sql_id is also specified. Jointly, they can be used to display monitoring information associated to any execution of the statement identified by sql_id, assuming that this statement was monitored. When NULL (the default), the last monitored execution of SQL sql_id is shown.</td>
</tr>
<tr>
<td>inst_id</td>
<td>Only considers statements started on the specified instance. Use -1 to target the login instance. NULL (default) targets all instances.</td>
</tr>
<tr>
<td>start_time_filter</td>
<td>If not NULL, the report considers only the activity (from GV$ACTIVE_SESSION_HISTORY) recorded after the specified date. If NULL, the reported activity starts when the execution of the targeted SQL statement has started.</td>
</tr>
<tr>
<td>end_time_filter</td>
<td>If not NULL, the report shows only the activity (from GV$ACTIVE_SESSION_HISTORY) collected before the date end_time_filter. If NULL, the reported activity ends when the targeted SQL statement execution has ended or is the current time if the statement is still executing.</td>
</tr>
<tr>
<td>instance_id_filter</td>
<td>Only applies when the execution runs parallel across multiple Oracle Real Application Cluster (Oracle RAC) instances. This parameter allows to only report the activity of the specified instance. Use a NULL value (the default) to include the activity on all instances where the parallel query was executed.</td>
</tr>
</tbody>
</table>
### Table 160-33  (Cont.) REPORT_SQL_MONITOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| parallel_filter   | Applies only to parallel execution and allows reporting the activity of only a subset of the processes involved in the parallel execution (Query Coordinator and/or Parallel eXecution servers). The value of this parameter can be:  
  - NULL to target all processes  
  - qc|servers(<svr_grp>, <svr_set>, <srv_num>)):  
  - 'qc' stands for query coordinator and 'servers()' stipulate which PX servers to consider.  
  The following examples show how to target a subset of the parallel processes:  
  - qc: targets only the query coordinator  
  - servers(1): targets all parallel execution servers in group number 1. Note that statement running parallel have one main server group (group number 1) plus one additional group for each nested sub-query running parallel.  
  - servers(,2): targets all parallel execution servers from any group but only running in set 1 of each group (each group has at most two set of parallel execution servers)  
  - servers(1,1): consider only group 1, set 1  
  - servers(1,2,4): consider only group 1, set 2, server number 4. This reports for a single parallel server process  
  - qc servers(1,2,4): same as above by also including the query coordinator  |
| event_detail      | When value is 'YES' (the default), reported activity from GV$ACTIVE_SESSION_HISTORY is aggregated by (wait_class, event_name). Use 'NO' to only aggregate by wait_class.                                              |
| bucket_max_count  | If specified, this should be the maximum number of histogram buckets created in the report                                                                                                                  |
| bucket_interval   | If specified, this represents the exact time interval in seconds, of all histogram buckets. If specified, bucket_max_count is ignored.                                                                          |
| base_path         | URL path for flex HTML resources since flex HTML format is required to access external files (java scripts and the flash SWF file itself)                                                                 |
| last_refresh_time | If not NULL (default is NULL), the time when the report was last retrieved (see SYSDATE attribute of the report tag). Use this option to display the report of a running query, and when the report is refreshed on a regular basis. This optimizes the size of the report since only the new or changed information is returned. In particular, the following are optimized:  
  - SQL text is not returned when this option is specified  
  - activity histogram starts at the bucket that intersect at that time. The entire content of the bucket is returned, even if last_refresh_time is after the start of that bucket |
### Table 160-33 (Cont.) REPORT_SQL_MONITOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report_level</td>
<td>Level of detail for the report: 'NONE', 'BASIC', 'TYPICAL' or 'ALL'.</td>
</tr>
<tr>
<td></td>
<td>Default assumes 'TYPICAL'.</td>
</tr>
<tr>
<td></td>
<td>In addition, individual report sections can also be enabled or disabled by</td>
</tr>
<tr>
<td></td>
<td>using a +/- section_name. Several sections are defined:</td>
</tr>
<tr>
<td></td>
<td>+ 'XPLAN' - Show explain plan; ON by default</td>
</tr>
<tr>
<td></td>
<td>+ 'PLAN' - Show plan monitoring statistics; ON by default</td>
</tr>
<tr>
<td></td>
<td>+ 'SESSIONS' - Show session details. Applies only to parallel queries; ON</td>
</tr>
<tr>
<td></td>
<td>by default</td>
</tr>
<tr>
<td></td>
<td>+ 'INSTANCE' - Show instance details. Applies only to parallel and cross</td>
</tr>
<tr>
<td></td>
<td>instance.</td>
</tr>
<tr>
<td></td>
<td>+ 'PARALLEL' - An umbrella parameter for specifying sessions+instance</td>
</tr>
<tr>
<td></td>
<td>details</td>
</tr>
<tr>
<td></td>
<td>+ 'ACTIVITY' - Show activity summary at global level, plan line level and</td>
</tr>
<tr>
<td></td>
<td>session or instance level (if applicable); ON by default</td>
</tr>
<tr>
<td></td>
<td>+ 'BINDS' - Show bind information when available; ON by default</td>
</tr>
<tr>
<td></td>
<td>+ 'METRICS' - Show metric data (CPU, I/Os, ...) over time; ON by default</td>
</tr>
<tr>
<td></td>
<td>+ 'ACTIVITY_HISTOGRAM' - Show an histogram of the overall query activity;</td>
</tr>
<tr>
<td></td>
<td>ON by default</td>
</tr>
<tr>
<td></td>
<td>+ 'PLAN_HISTOGRAM' - Show activity histogram at plan line level; OFF by</td>
</tr>
<tr>
<td></td>
<td>default</td>
</tr>
<tr>
<td></td>
<td>+ 'OTHER' - Other info; ON by default</td>
</tr>
<tr>
<td></td>
<td>In addition, SQL text can be specified at different levels:</td>
</tr>
<tr>
<td></td>
<td>+ SQL_TEXT - No SQL text in report</td>
</tr>
<tr>
<td></td>
<td>+ SQL_TEXT - OK with partial SQL text up to the first 2000 chars as stored</td>
</tr>
<tr>
<td></td>
<td>in GV$SQL_MONITOR</td>
</tr>
<tr>
<td></td>
<td>+ SQL_FULLTEXT - Show full SQL text (default value)</td>
</tr>
<tr>
<td></td>
<td>+ SQL_FULLTEXT - Show full SQL text (default value)</td>
</tr>
<tr>
<td></td>
<td>The meanings of the three top-level report levels are:</td>
</tr>
<tr>
<td></td>
<td>+ NONE - minimum possible</td>
</tr>
<tr>
<td></td>
<td>+ BASIC - SQL_TEXT-PLAN-XPLAN-SESSIONS-INSTANCE-</td>
</tr>
<tr>
<td></td>
<td>ACTIVITY_HISTOGRAM-PLAN_HISTOGRAM-METRICS</td>
</tr>
<tr>
<td></td>
<td>+ TYPICAL - everything but PLAN_HISTOGRAM</td>
</tr>
<tr>
<td></td>
<td>+ ALL - everything</td>
</tr>
<tr>
<td></td>
<td>Only one of these 4 levels can be specified and, if it is, it has to be at</td>
</tr>
<tr>
<td></td>
<td>the start of the REPORT_LEVEL string</td>
</tr>
<tr>
<td>type</td>
<td>Report format, 'TEXT' by default. Can be 'TEXT', 'HTML', 'XML' or</td>
</tr>
<tr>
<td></td>
<td>'ACTIVE' (see Usage Notes).</td>
</tr>
<tr>
<td>sql_plan_hash_value</td>
<td>Target only those SQL executions with the specified</td>
</tr>
<tr>
<td></td>
<td>plan_hash_value. Default is NULL.</td>
</tr>
<tr>
<td>con_name</td>
<td>Name of the multitenant container database (CDB).</td>
</tr>
<tr>
<td>report_id</td>
<td>ID of the report in auto-report repository. Report IDs can be found in</td>
</tr>
<tr>
<td></td>
<td>DBA_HIST_REPORTS.</td>
</tr>
</tbody>
</table>
Return Values

A CLOB containing the desired report.

Usage Notes

- The target SQL statement for this report can be:
  - The most recent SQL statement monitored by Oracle Database. This is the default behavior, so there is no need to specify any parameter.
  - The most recent SQL statement executed by a specific session and monitored by Oracle. The session is identified by its session id and optionally its serial number. For example, use `session_id =>` for the current session or `session_id => 20, session_serial => 103` for session ID 20, serial number 103.
  - The most recent execution of a specific statement identified by its `sql_id`.
  - A specific execution of a SQL statement identified by its execution key (`sql_id, sql_exec_start` and `sql_exec_id`).

- This report produces performance data exposed by several fixed views, listed below. For this reason, the invoker of the report function must have privilege to select data from these fixed views (such as the `SELECT_CATALOG` role).
  - `GV$SQL_MONITOR`
  - `GV$SQL_PLAN_MONITOR`
  - `GV$SQL_PLAN`
  - `GV$ACTIVE_SESSION_HISTORY`
  - `GV$SESSION_LONGOPS`
  - `GV$SQL`

- The `bucket_max_count` and `bucket_interval` parameters control the activity histogram.

By default, the maximum number of buckets is set to 128. The database derives the `bucket_interval` value based on this count. The `bucket_interval` (value is in seconds) is computed such that it is the smallest possible power of 2 value (starting at 1 second) without exceeding the maximum number of buckets. For example, if the query has executed for 600 seconds, then the database selects a `bucket_interval` of 8 seconds (a power of two). The database chooses the value of 8 because $600/8 = 74$, which is less than 128 buckets maximum. Smaller than 8 seconds would be 4 seconds, which would lead to more buckets than the 128 maximum. If `bucket_interval` is specified, then the database uses the specified value instead of deriving it from `bucket_max_count`.

- ACTIVE reports have a rich, interactive user interface similar to Enterprise Manager, while not requiring any EM installation.

The report file is in HTML format. The code powering the active report is downloaded transparently by the web browser when the report is first viewed. Therefore, viewing the report requires outside connectivity.
160.5.28 REPORT_SQL_MONITOR_LIST Function

This function builds a report for all or a subset of statements monitored by Oracle Database. For each statement, the subprogram gives key information and associated global statistics.

Use the REPORT_SQL_MONITOR Function to get detailed monitoring information for a single SQL statement.

See Also:

Real-Time SQL Monitoring for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REPORT_SQL_MONITOR_LIST(
  sql_id                    IN VARCHAR2  DEFAULT  NULL,
  session_id                IN NUMBER    DEFAULT  NULL,
  session_serial            IN NUMBER    DEFAULT  NULL,
  inst_id                   IN NUMBER    DEFAULT  NULL,
  active_since_date         IN DATE      DEFAULT  NULL,
  active_since_sec          IN NUMBER    DEFAULT  NULL,
  active_before_date        IN DATE      DEFAULT  NULL,
  last_refresh_time         IN DATE      DEFAULT  NULL,
  dbop_name                 IN VARCHAR2  DEFAULT  NULL,
  monitor_type              IN NUMBER    DEFAULT  MONITOR_TYPE_ALL,
  max_sqltext_length        IN NUMBER    DEFAULT  NULL,
  top_n_count               IN NUMBER    DEFAULT  NULL,
  top_n_rankby              IN VARCHAR2  DEFAULT  'LAST_ACTIVE_TIME',
  report_level              IN VARCHAR2  DEFAULT  'TYPICAL',
  auto_refresh              IN NUMBER    DEFAULT  NULL,
  base_path                 IN VARCHAR2  DEFAULT  NULL,
  type                      IN VARCHAR2  DEFAULT  'TEXT',
  con_name                  IN VARCHAR2  DEFAULT  NULL,
  top_n_detail_count        IN NUMBER    DEFAULT  NULL)
RETURN CLOB;
```
### Parameters

Table 160-34 REPORT_SQL_MONITOR_LIST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQL ID for which monitoring information should be displayed. Use NULL (the default) to report on the last statement monitored by Oracle.</td>
</tr>
<tr>
<td>session_id</td>
<td>If not NULL, then this parameter targets only the subset of statements executed by the specified session. Default is NULL. Use -1 or USERENV('SID') for current session.</td>
</tr>
<tr>
<td>session_serial</td>
<td>In addition to the session_id parameter, you can also specify its session serial to ensure that the desired session incarnation is targeted. This parameter is ignored when session_id is NULL.</td>
</tr>
<tr>
<td>inst_id</td>
<td>Only considers statements started on the specified instance. Use -1 to target the login instance. NULL (default) targets all instances.</td>
</tr>
<tr>
<td>active_since_date</td>
<td>If not NULL (default), returns only monitored statements active since the specified time. This includes all statements that are still executing along with all statements that have completed their execution after the specified date and time.</td>
</tr>
<tr>
<td>active_since_sec</td>
<td>Same as active_since_date but with the date specified relative to the current SYSDATE minus a specified number of seconds. For example, use 3600 to apply a limit of 1 hour.</td>
</tr>
<tr>
<td>active_before_date</td>
<td>If not NULL (default), returns only monitored statements that have been active before the specified date and time.</td>
</tr>
<tr>
<td>last_refresh_time</td>
<td>If not NULL (default), the date and time when the list report was last retrieved. This optimizes the case where an application shows the list and refreshes the report on a regular basis (such as once every 5 seconds). In this case, the report shows detail about the execution of monitored queries that active since the specified last_refresh_time. For other queries, the report returns the execution key (sql_id, sql_exec_start, sql_exec_id). For queries with a first refresh time after the specified date, the function returns only the SQL execution key and statistics.</td>
</tr>
<tr>
<td>dbop_name</td>
<td>DB operation name. Specify NULL to display all the monitored DB operations.</td>
</tr>
<tr>
<td>monitor_type</td>
<td>Type of the monitor operation. It can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• MONITOR_TYPE_SQL - Returns only SQLs</td>
</tr>
<tr>
<td></td>
<td>• MONITOR_TYPE_DBOP - Returns only DB operations</td>
</tr>
<tr>
<td></td>
<td>• MONITOR_TYPE_ALL - Returns SQLs as well as DB operations</td>
</tr>
<tr>
<td>max_sqltext_length</td>
<td>Maximum length of the SQL text. Default is NULL (no limit).</td>
</tr>
<tr>
<td>top_n_count</td>
<td>Limits the number of top-N SQL statements that need to be included in the report.</td>
</tr>
</tbody>
</table>
Table 160-34  (Cont.) REPORT_SQL_MONITOR_LIST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top_n_rankby</td>
<td>Specifies the attribute to rank the SQL statements. Specify this value when top_n_count value is not NULL. The ranking of an SQL statement is done based on one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• LAST_ACTIVE_TIME - Last active date and time (top N most recent)</td>
</tr>
<tr>
<td></td>
<td>• DURATION - Total duration of execution</td>
</tr>
<tr>
<td></td>
<td>• DB_TIME - DB time used</td>
</tr>
<tr>
<td></td>
<td>• CPU_TIME - CPU time used</td>
</tr>
<tr>
<td></td>
<td>• IO_REQUESTS - Number of I/O requests</td>
</tr>
<tr>
<td></td>
<td>• IO_BYTES - Number of I/O bytes</td>
</tr>
<tr>
<td>report_level</td>
<td>Level of detail for the report. The level is one of the following:</td>
</tr>
<tr>
<td></td>
<td>• BASIC - SQL text up to 200 characters</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL - include full SQL text assuming that cursor has not aged out, in which case the SQL text is included up to 2000 characters</td>
</tr>
<tr>
<td></td>
<td>• ALL - currently the same as TYPICAL</td>
</tr>
<tr>
<td>auto_refresh</td>
<td>Currently non-operational, reserved for future use.</td>
</tr>
<tr>
<td>base_path</td>
<td>URL path for flex HTML resources because flex HTML format is required to access external files (java scripts and the flash SWF file itself).</td>
</tr>
<tr>
<td>type</td>
<td>Report format: TEXT (default), HTML, or XML.</td>
</tr>
<tr>
<td>con_name</td>
<td>Name of the multitenant container database (CDB)</td>
</tr>
<tr>
<td>top_n_detail_count</td>
<td>Limits the number of top-N SQL statements for which the SQL monitor details need to be included in the report.</td>
</tr>
</tbody>
</table>

Return Values

A report (XML, text, HTML) for the list of SQL statements that have been monitored.

Usage Notes

You must have the privilege to access the following fixed views: GV$SQL_MONITOR and GV$SQL.

See Also:

Oracle Database SQL Tuning Guide for more information about SQL real-time monitoring.
160.5.29 REPORT_TUNING_TASK Function

This function displays the results of a tuning task. By default the report is in text format.

**See Also:**

DBMS_SQLTUNE SQL Performance Reporting Subprograms for other subprograms in this group

### Syntax

```sql
DBMS_SQLTUNE.REPORT_TUNING_TASK(
    task_name        IN   VARCHAR2,
    type             IN   VARCHAR2   := 'TEXT',
    level            IN   VARCHAR2   := 'TYPICAL',
    section          IN   VARCHAR2   := ALL,
    object_id        IN   NUMBER     := NULL,
    result_limit     IN   NUMBER     := NULL,
    owner_name       IN   VARCHAR2   := NULL,
    execution_name   IN   VARCHAR2   := NULL,
    database_link_to IN   VARCHAR2   := NULL)
RETURN CLOB;
```

### Parameters

**Table 160-35 REPORT_TUNING_TASK Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the tuning task.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the report to produce. Possible values are <code>TEXT</code> which produces a text report.</td>
</tr>
<tr>
<td>level</td>
<td>Level of detail in the report:</td>
</tr>
<tr>
<td></td>
<td>- <code>BASIC</code>: simple version of the report. Just show info about the actions taken by the advisor.</td>
</tr>
<tr>
<td></td>
<td>- <code>TYPICAL</code>: show information about every statement analyzed, including requests not implemented.</td>
</tr>
<tr>
<td></td>
<td>- <code>ALL</code>: highly detailed report level, also provides annotations about statements skipped over.</td>
</tr>
<tr>
<td>section</td>
<td>Section of the report to include.</td>
</tr>
<tr>
<td></td>
<td>You can limit the report to any of the following single sections (<code>ALL</code> for all sections):</td>
</tr>
<tr>
<td></td>
<td>- <code>SUMMARY</code>: Summary information</td>
</tr>
<tr>
<td></td>
<td>- <code>FINDINGS</code>: Tuning findings</td>
</tr>
<tr>
<td></td>
<td>- <code>PLAN</code>: Explain plans</td>
</tr>
<tr>
<td></td>
<td>- <code>INFORMATION</code>: General information</td>
</tr>
<tr>
<td></td>
<td>- <code>ERROR</code>: Statements with errors</td>
</tr>
<tr>
<td></td>
<td>- <code>ALL</code>: All statements</td>
</tr>
</tbody>
</table>
Table 160-35  (Cont.) REPORT_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>Advisor framework object ID that represents a single statement to restrict reporting to. NULL for all statements. Only valid for reports that target a single execution.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Maximum number of SQL statements to show in the report.</td>
</tr>
<tr>
<td>owner_name</td>
<td>Owner of the relevant tuning task. The default is the current schema owner.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name of the task execution to use. If NULL, then the function generates the report for the last task execution.</td>
</tr>
<tr>
<td>database_link_to</td>
<td>Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';</td>
</tr>
</tbody>
</table>

Return Values

A CLOB containing the desired report.

Examples

-- Display the report for a single statement.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:stmt_task) FROM   DUAL;

-- Display the summary for a SQL tuning set.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT', 'TYPICAL', 'SUMMARY') FROM   DUAL;

-- Display the findings for a specific statement.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT', 'TYPICAL','FINDINGS', 5) FROM   DUAL;
160.5.30 REPORT_TUNING_TASK_XML Function

This function displays an XML report of a tuning task.

See Also:

DBMS_SQLTUNE SQL Performance Reporting Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.REPORT_TUNING_TASK_LIST_XML(
    task_name        IN   VARCHAR2   := NULL,
    level            IN   VARCHAR2   := LEVEL_TYPICAL,
    section          IN   VARCHAR2   := SECTION_ALL,
    object_id        IN   NUMBER     := NULL,
    result_limit     IN   NUMBER     := 160,
    owner_name       IN   VARCHAR2   := NULL,
    execution_name   IN   VARCHAR2   := NULL,
    autotune_period  IN   NUMBER     := NULL,
    report_tag       IN   VARCHAR2   := NULL)
RETURN XMLTYPE;

Parameters

Table 160-36  REPORT_TUNING_TASK_XML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the tuning task.</td>
</tr>
<tr>
<td>level</td>
<td>Level of detail in the report:</td>
</tr>
<tr>
<td></td>
<td>- BASIC: simple version of the report. Just show info about the</td>
</tr>
<tr>
<td></td>
<td>actions taken by the advisor.</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL: show information about every statement analyzed,</td>
</tr>
<tr>
<td></td>
<td>including requests not implemented.</td>
</tr>
<tr>
<td></td>
<td>- ALL: highly detailed report level, also provides annotations about</td>
</tr>
<tr>
<td></td>
<td>statements skipped over.</td>
</tr>
<tr>
<td>section</td>
<td>Section of the report to include.</td>
</tr>
<tr>
<td></td>
<td>You can limit the report to any of the following single sections (ALL for</td>
</tr>
<tr>
<td></td>
<td>all sections):</td>
</tr>
<tr>
<td></td>
<td>- SUMMARY - Summary information</td>
</tr>
<tr>
<td></td>
<td>- ALL - All statements</td>
</tr>
<tr>
<td>object_id</td>
<td>Advisor framework object ID that represents a single statement to</td>
</tr>
<tr>
<td></td>
<td>restrict reporting to. NULL for all statements. Only valid for reports</td>
</tr>
<tr>
<td></td>
<td>that target a single execution.</td>
</tr>
</tbody>
</table>
Table 160-36  (Cont.) REPORT_TUNING_TASK_XML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result_limit</td>
<td>The number of statements in a SQL tuning set or snapshot range for which the report is generated. The default is 160 (20 statements * 8 categories). The categories are as follows:</td>
</tr>
<tr>
<td></td>
<td>· Profile</td>
</tr>
<tr>
<td></td>
<td>· Index</td>
</tr>
<tr>
<td></td>
<td>· Restructure SQL</td>
</tr>
<tr>
<td></td>
<td>· Alternate plan</td>
</tr>
<tr>
<td></td>
<td>· Statistics</td>
</tr>
<tr>
<td></td>
<td>· Errors</td>
</tr>
<tr>
<td></td>
<td>· Information</td>
</tr>
<tr>
<td></td>
<td>· No findings</td>
</tr>
<tr>
<td>owner_name</td>
<td>Owner of the relevant tuning task. The default is the current schema owner.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name of the task execution to use. If NULL, then the function generates the report for the most recent task execution.</td>
</tr>
<tr>
<td>autotune_period</td>
<td>The time period for the automatic SQL tuning. This setting applies only to the automatic SQL Tuning Advisor task. Possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td>· Null or negative value (default) — All or full. The result includes all task executions.</td>
</tr>
<tr>
<td></td>
<td>· 0 — Result of the current or most recent task execution.</td>
</tr>
<tr>
<td></td>
<td>· 1 — Result for the most recent 24-hour period.</td>
</tr>
<tr>
<td></td>
<td>· 7 — Result for the most recent 7-day period.</td>
</tr>
<tr>
<td></td>
<td>The procedure interprets any other value as the time of the most recent task execution minus the value of this argument.</td>
</tr>
<tr>
<td>report_tag</td>
<td>The name of the root XML tag. By default, the tag is the report reference generated by the reporting framework.</td>
</tr>
</tbody>
</table>

Return Values

A CLOB containing the desired report.

160.5.31 RESET_TUNING_TASK Procedure

This procedure is called on a tuning task that is not currently executing to prepare it for re-execution.

See Also:

DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group
Syntax

```
DBMS_SQLTUNE.RESET_TUNING_TASK(
    task_name         IN VARCHAR2);
```

Parameters

**Table 160-37  RESET_TUNING_TASK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the tuning task to reset</td>
</tr>
</tbody>
</table>

Examples

```
-- reset and re-execute a task
EXEC DBMS_SQLTUNE.RESET_TUNING_TASK(:sts_task);

-- re-execute the task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);
```

160.5.32 RESUME_TUNING_TASK Procedure

This procedure resumes a previously interrupted task that was created to process a SQL tuning set.

**See Also:**

DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.RESUME_TUNING_TASK(
    task_name         IN VARCHAR2,
    basic_filter      IN VARCHAR2 := NULL);
```

Parameters

**Table 160-38  RESUME_TUNING_TASK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the tuning task to resume.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>A SQL predicate to filter the SQL from the SQL tuning set. Note that this filter is applied in conjunction with the basic filter (i.e., parameter basic_filter) when calling CREATE_TUNING_TASK Functions.</td>
</tr>
</tbody>
</table>
Usage Notes

Resuming a single SQL tuning task (a task that was created to tune a single SQL statement as compared to a SQL tuning set) is not supported.

Examples

-- Interrupt the task
EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:conc_task);

-- Once a task is interrupted, we can elect to reset it, resume it, or check
-- out its results and then decide. For this example we will just resume.
EXEC DBMS_SQLTUNE.RESUME_TUNING_TASK(:conc_task);

160.5.33 SCHEDULE_TUNING_TASK Function

This function creates a tuning task for a single SQL statement and schedules a DBMS_SCHEDULER job to execute the tuning task. One form of the function finds the information about the statement to be tuned in the shared SQL area, whereas the other finds the information in AWR.

See Also:

DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

Shared SQL Area Format:

```
DBMS_SQLTUNE.SCHEDULE_TUNING_TASK(
    sql_id          IN VARCHAR2,
    plan_hash_value IN NUMBER                   := NULL,
    start_date      IN TIMESTAMP WITH TIME ZONE := NULL,
    scope           IN VARCHAR2                 := SCOPE_COMPREHENSIVE,
    time_limit      IN NUMBER                   := TIME_LIMIT_DEFAULT,
    task_name       IN VARCHAR2                 := NULL,
    description     IN VARCHAR2                 := NULL,
    con_name        IN VARCHAR2                 := NULL)
RETURN VARCHAR2;
```

AWR Format:

```
DBMS_SQLTUNE.SCHEDULE_TUNING_TASK(
    begin_snap     IN NUMBER,
    end_snap       IN NUMBER,
```
Parameters

Table 160-39  SCHEDULE_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin_snap</td>
<td>The beginning snapshot identifier. The range is exclusive, which means that SQL statements in this snapshot ID are not included.</td>
</tr>
<tr>
<td>end_snap</td>
<td>The end snapshot identifier. The range is inclusive, which means that SQL statements in this snapshot ID are included.</td>
</tr>
<tr>
<td>sql_id</td>
<td>The SQL ID of the statement to be tuned.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>The plan hash value of the statement to be tuned. For example, the tuning job fetches captured binds for this SQL plan.</td>
</tr>
<tr>
<td>start_date</td>
<td>The date on which the schedule becomes valid. If null, then SQL Tuning Advisor immediately executes the task.</td>
</tr>
<tr>
<td>scope</td>
<td>The scope of the tuning job: limited, or comprehensive.</td>
</tr>
<tr>
<td>time_limit</td>
<td>The maximum duration in seconds for the SQL tuning session.</td>
</tr>
<tr>
<td>task_name</td>
<td>Optional SQL tuning task name.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the SQL tuning session. The description can contain a maximum of 256 characters.</td>
</tr>
<tr>
<td>con_name</td>
<td>The container from which SQL Tuning Advisor accesses the SQL statement information.</td>
</tr>
<tr>
<td>dbid</td>
<td>DBID for imported or PDB-level AWR data. If NULL, then the current database DBID is used.</td>
</tr>
</tbody>
</table>

Security Model

The caller must possess the CREATE_JOB privilege for the job.

Return Values

A SQL tuning task name that is unique for each user. Multiple users can assign the same name to their advisor tasks.

Usage Notes

- The task is scheduled only once.
- The name of the scheduler job is created as follows:
  sqltune_job_taskid_orahash(systimestamp).
160.5.34 SCRIPT_TUNING_TASK Function

This function creates a SQL*Plus script which can then be executed to implement a set of SQL Tuning Advisor recommendations.

See Also:

DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SCRIPT_TUNING_TASK(
    task_name         IN VARCHAR2,
    rec_type          IN VARCHAR2  := REC_TYPE_ALL,
    object_id         IN NUMBER    := NULL,
    result_limit      IN NUMBER    := NULL,
    owner_name        IN VARCHAR2  := NULL,
    execution_name    IN VARCHAR2  := NULL,
    database_link_to  IN VARCHAR2  := NULL)
RETURN CLOB;
```

Parameters

Table 160-40  SCRIPT_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the tuning task for which to apply a script.</td>
</tr>
<tr>
<td>rec_type</td>
<td>Filter the script by types of recommendations to include. You can use any subset of the following values, separated by commas: 'ALL','PROFILES','STATISTICS','INDEXES'. For example, a script with profiles and statistics would use the filter 'PROFILES,STATISTICS'.</td>
</tr>
<tr>
<td>object_id</td>
<td>Optionally filters by a single object ID.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Optionally shows commands for only top n SQL (ordered by object_id and ignored if an object_id is also specified).</td>
</tr>
<tr>
<td>owner_name</td>
<td>Owner of the relevant tuning task. Defaults to the current schema owner.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Name of the task execution to use. If NULL, the script is generated for the last task execution.</td>
</tr>
</tbody>
</table>
Table 160-40  (Cont.) SCRIPT_TUNING_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database_link_to</td>
<td>Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri:</td>
</tr>
<tr>
<td></td>
<td>CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';</td>
</tr>
</tbody>
</table>

Return Values

Returns a script in the form of a CLOB.

Usage Notes

- After the script is returned, check it before executing it.
- Wrap with a call to DBMS_ADVISOR.CREATE_FILE to put it into a file.

Examples

SET LINESIZE 140

-- Get a script for all actions recommended by the task.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task) FROM DUAL;

-- Get a script of only the sql profiles we should create.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'PROFILES') FROM DUAL;

-- Get a script of only stale / missing stats
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'STATISTICS') FROM DUAL;

-- Get a script with recommendations about only one SQL statement when we have
-- tuned an entire STS.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:sts_task, 'ALL', 5) FROM DUAL;
160.5.35 SELECT_CURSOR_CACHE Function

This function collects SQL statements from the shared SQL area.

See Also:
DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.SELECT_CURSOR_CACHE (  
  basic_filter        IN   VARCHAR2 := NULL,  
  object_filter       IN   VARCHAR2 := NULL,  
  ranking_measure1    IN   VARCHAR2 := NULL,  
  ranking_measure2    IN   VARCHAR2 := NULL,  
  ranking_measure3    IN   VARCHAR2 := NULL,  
  result_percentage   IN   NUMBER   := 1,  
  result_limit        IN   NUMBER   := NULL,  
  attribute_list      IN   VARCHAR2 := NULL,  
  recursive_sql       IN   VARCHAR2 := HAS_RECURSIVE_SQL)  
RETURN sys.sqlset PIPELINED;

Parameters

Table 160-41  SELECT_CURSOR_CACHE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| basic_filter      | Specifies the SQL predicate that filters the SQL from the shared SQL area defined on attributes of the SQLSET_ROW.  
                   | If basic_filter is not set by the caller, then the subprogram captures only statements of the type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE. |
| object_filter     | Currently not supported.                                                                                                                                 |
| ranking_measure(n)| Defines an ORDER BY clause on the selected SQL.  
                   |                                                                                                                                 |
| result_percentage | Specifies a filter that picks the top n% according to the supplied ranking measure.  
                   | The value applies only if one ranking measure is supplied.                                                                                                                                 |
| result_limit      | Defines the top limit SQL from the filtered source ranked by the ranking measure.                                                                 |

Chapter 160
Summary of DBMS_SQLTUNE Subprograms
Table 160-41  (Cont.) SELECT_CURSOR_CACHE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_list</td>
<td>Specifies the list of SQL statement attributes to return in the result. Possible values are:</td>
</tr>
<tr>
<td>recursive_sql</td>
<td>Specifies that the filter must include recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL, which is the default) or exclude it (NO_RECURSIVE_SQL).</td>
</tr>
</tbody>
</table>

**Return Values**

This function returns a one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

**Usage Notes**

- Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.
- Users need privileges on the shared SQL area views.

**Examples**

```sql
-- Get sql ids and sql text for statements with 500 buffer gets.
SELECT sql_id, sql_text
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('buffer_gets > 500'))
ORDER BY sql_id;

-- Get all the information we have about a particular statement.
SELECT *
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('sql_id = ''4rm4183czbs7j'''));

-- Notice that some statements can have multiple plans. The output of the SELECT XXX table functions is unique by (sql_id, plan_hash_value). This is because a data source can store multiple plans per sql statement.
SELECT sql_id, plan_hash_value
FROM table(dbms_sqltune.select_cursor_cache('sql_id = ''aylm3ssvtrh24'''))
```
ORDER BY sql_id, plan_hash_value;

-- PL/SQL examples: load_sqlset is called after opening a cursor, along the
-- lines given below

-- Select all statements in the shared SQL area.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT value(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;

  -- Process each statement (or pass cursor to load_sqlset).
  CLOSE cur;
END;/

-- Look for statements not parsed by SYS.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur for
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS''')) P;

  -- Process each statement (or pass cursor to load_sqlset).
  CLOSE cur;
end;/

-- All statements from a particular module/action.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT value(P)
    FROM table( DBMS_SQLTUNE.SELECT_CURSOR_CACHE('module = ''MY_APPLICATION'' and action = ''MY_ACTION'')) P;

  -- Process each statement (or pass cursor to load_sqlset)
  CLOSE cur;
END;/

-- all statements that ran for at least five seconds
DECLARE
  cur sys_refcursor;
BEGIN
OPEN cur FOR
  SELECT VALUE(P)
  FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('elapsed_time > 5000000')) P;

-- Process each statement (or pass cursor to load_sqlset)
CLOSE cur;
end;/

-- select all statements that pass a simple buffer_gets threshold and
-- are coming from an APPS user
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
        'buffer_gets > 100 and parsing_schema_name = ''APPS'''))P;

  -- Process each statement (or pass cursor to load_sqlset)
  CLOSE cur;
  end;/

-- select all statements exceeding 5 seconds in elapsed time, but also
-- select the plans (by default we only select execution stats and binds
-- for performance reasons - in this case the SQL_PLAN attribute of
sqlset_row
-- is NULL)
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(dbms_sqltune.select_cursor_cache(
      'elapsed_time > 5000000', NULL, NULL, NULL, NULL, 1, NULL,
      'EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN')) P;

  -- Process each statement (or pass cursor to load_sqlset)
  CLOSE cur;
  END;/

-- Select the top 100 statements in the shared SQL area ordering by
elapsed_time.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
-- Process each statement (or pass cursor to load_sqlset)

CLOSE cur;
end;
/

-- Select the set of statements which cumulatively account for 90% of the
-- buffer gets in the shared SQL area. This means that the buffer gets of all
-- of these statements added up is approximately 90% of the sum of all
-- statements currently in the cache.

DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE(NULL,
            NULL,
            'BUFFER_GETS', NULL,
            NULL,
            .9)) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
END;
/

160.5.36 SELECT_SQL_TRACE Function

This table function reads the content of one or more trace files and returns the SQL statements it finds in the format of sqlset_row.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.SELECT_SQL_TRACE (   directory              IN VARCHAR2,
  file_name              IN VARCHAR2 := NULL,
mapping_table_name IN VARCHAR2 := NULL,
mapping_table_owner IN VARCHAR2 := NULL,
select_mode IN POSITIVE := SINGLE_EXECUTION,
options IN BINARY_INTEGER := LIMITED_COMMAND_TYPE,
pattern_start IN VARCHAR2 := NULL,
pattern_end IN VARCHAR2 := NULL,
result_limit IN POSITIVE := NULL)
RETURN sys.sqlset PIPELINED;

Parameters

Table 160-42  SELECT_SQL_TRACE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>Defines the directory object containing the trace files. This field is mandatory.</td>
</tr>
<tr>
<td>file_name</td>
<td>Specifies all or part of the name of the trace files. If NULL, then the function uses the current or most recent file in the specified location or path. ' % ' wildcards are supported for matching trace file names.</td>
</tr>
<tr>
<td>mapping_table_name</td>
<td>Specifies the mapping table name. Note that the mapping table name is case insensitive. If the mapping table name is NULL, then the function uses the mappings in the current database.</td>
</tr>
<tr>
<td>mapping_table_owner</td>
<td>Specifies the mapping table owner. If it is NULL, then the function uses the current user.</td>
</tr>
<tr>
<td>select_mode</td>
<td>Specifies the mode for selecting SQL from the trace. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· SINGLE_EXECUTION — Returns one execution of a SQL. This is the default.</td>
</tr>
<tr>
<td></td>
<td>· ALL_EXECUTIONS — Returns all executions.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies which types of SQL statements are returned. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· LIMITED_COMMAND_TYPE — Returns the SQL statements with the command types CREATE, INSERT, SELECT, UPDATE, DELETE, and MERGE. This value is the default.</td>
</tr>
<tr>
<td></td>
<td>· ALL_COMMAND_TYPE — Returns the SQL statements with all command types.</td>
</tr>
<tr>
<td>pattern_start</td>
<td>Specifies the delimiting pattern of the trace file sections to consider. CURRENTLY INOPERABLE.</td>
</tr>
<tr>
<td>pattern_end</td>
<td>Specifies the closing delimiting pattern of the trace file sections to process. CURRENTLY INOPERABLE.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Specifies the top SQL from the filtered source. Default to MAXSB4 if NULL.</td>
</tr>
</tbody>
</table>

Return Values

This function returns a SQLSET_ROW object.

Usage Notes

The ability to create a directory object for the system directory creates a potential security issue. For example, in a CDB, all containers write trace files to the same directory. A local user with SELECT privileges on this directory can read the contents of trace files belonging to any container.
To prevent this type of unauthorized access, copy the files from the default SQL trace directory into a different directory, and then create a directory object. Use the **PATH_PREFIX** clause of the **CREATE PLUGGABLE DATABASE** statement to ensure that all directory object paths associated with the PDB are restricted to the specified directory or its subdirectories.

**Examples**

The following code shows how to enable SQL trace for a few SQL statements and load the results into a SQL tuning set:

```sql
-- turn on the SQL trace in the capture database
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 4'

-- run sql statements
SELECT 1 FROM DUAL;
SELECT COUNT(*) FROM dba_tables WHERE table_name = :mytab;

ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF';

-- create mapping table from the capture database
CREATE TABLE mapping AS
SELECT object_id id, owner, substr(object_name, 1, 30) name
FROM dba_objects
WHERE object_type NOT IN ('CONSUMER GROUP', 'EVALUATION CONTEXT',
'FUNCTION', 'INDEXTYPE', 'JAVA CLASS',
'JAVA DATA', 'JAVA RESOURCE', 'LIBRARY',
'LOB', 'OPERATOR', 'PACKAGE',
'PACKAGE BODY', 'PROCEDURE', 'QUEUE',
'RESOURCE PLAN', 'TRIGGER', 'TYPE',
'TYPE BODY')
UNION ALL
SELECT user_id id, username owner, NULL name
FROM dba_users;

-- create the directory object where the SQL traces are stored
CREATE DIRECTORY SQL_TRACE_DIR as '/home/foo/trace';

-- create the STS
EXEC DBMS_SQLTUNE.CREATE_SQLSET('my_sts', 'test purpose');

-- load the SQL statements into STS from SQL TRACE
DECLARE
cur sys_refcursor;
BEGIN
  OPEN cur FOR
  SELECT value(p)
  FROM TABLE(
    DBMS_SQLTUNE.SELECT_SQL_TRACE(
      directory=>'SQL_TRACE_DIR',
      file_name=>'%trc',
      mapping_table_name=>'mapping')) p;
  DBMS_SQLTUNE.LOAD_SQLSET('my_sts', cur);
  CLOSE cur;
```
160.5.37 SELECT_SQLPA_TASK Function

This function collects SQL statements from a SQL Performance Analyzer comparison task.

Syntax

```sql
DBMS_SQLTUNE.SELECT_SQLPA_TASK(
    task_name         IN VARCHAR2,
    task_owner        IN VARCHAR2 := NULL,
    execution_name    IN VARCHAR2 := NULL,
    level_filter      IN VARCHAR2 := 'REGRESSED',
    basic_filter      IN VARCHAR2 := NULL,
    object_filter     IN VARCHAR2 := NULL,
    attribute_list    IN VARCHAR2 := 'TYPICAL')
RETURN sys.sqlset PIPELINED;
```

Parameters

**Table 160-43  SELECT_SQLPA_TASK Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Specifies the name of the SQL Performance Analyzer task.</td>
</tr>
<tr>
<td>task_owner</td>
<td>Specifies the owner of the SQL Performance Analyzer task. If NULL, then assume the current user.</td>
</tr>
<tr>
<td>execution_name</td>
<td>Specifies the name of the SQL Performance Analyzer task execution (type COMPARE PERFORMANCE) from which the provided filters will be applied. If NULL, then assume the most recent COMPARE PERFORMANCE execution.</td>
</tr>
</tbody>
</table>
Table 160-43 (Cont.) SELECT_SQLPA_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level_filter</td>
<td>Specifies which subset of SQL statements to include. Same format as DBMS_SQLPA.REPORT_ANALYSIS_TASK.LEVEL, with some possible strings removed.</td>
</tr>
<tr>
<td>basic filter</td>
<td>Specifies the SQL predicate to filter the SQL in addition to the level filters.</td>
</tr>
<tr>
<td>object_filter</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>Defines the SQL statement attributes to return in the result. Possible values are:</td>
</tr>
<tr>
<td>Return Values</td>
<td>This function returns a SQL tuning set object.</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>For example, you can use this function to create a SQL tuning set containing the subset of SQL statements that regressed during a SQL Performance Analyzer (SPA) experiment. You can also specify other arbitrary filters.</td>
</tr>
</tbody>
</table>

160.5.38 SELECT_SQLSET Function

This is a table function that reads the contents of a SQL tuning set.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group.
Syntax

DBMS_SQLTUNE.SELECT_SQLSET (
    sqlset_name      IN   VARCHAR2,  
    basic_filter     IN   VARCHAR2 := NULL,  
    object_filter    IN   VARCHAR2 := NULL,  
    ranking_measure1 IN   VARCHAR2 := NULL,  
    ranking_measure2 IN   VARCHAR2 := NULL,  
    ranking_measure3 IN   VARCHAR2 := NULL,  
    result_percentage IN   NUMBER   := 1,  
    result_limit     IN   NUMBER   := NULL,  
    attribute_list   IN   VARCHAR2 := NULL,  
    plan_filter      IN   VARCHAR2 := NULL,  
    sqlset_owner     IN   VARCHAR2 := NULL,  
    recursive_sql    IN   VARCHAR2 := HAS_RECURSIVE_SQL)
RETURN sys.sqlset PIPELINED;

Parameters

Table 160-44  SELECT_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set to query.</td>
</tr>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate to filter the SQL from the SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>defined on attributes of the SQLSET_ROW.</td>
</tr>
<tr>
<td>object_filter</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>ranking_measure(n)</td>
<td>Specifies an ORDER BY clause on the selected SQL.</td>
</tr>
<tr>
<td>result_percentage</td>
<td>Specifies a filter that picks the top n% according to the supplied ranking</td>
</tr>
<tr>
<td></td>
<td>measure. Note that this parameter applies only if one ranking measure is</td>
</tr>
<tr>
<td></td>
<td>supplied.</td>
</tr>
<tr>
<td>result_limit</td>
<td>The top limit SQL from the filtered source, ranked by the ranking measure.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>Defines the SQL statement attributes to return in the result.</td>
</tr>
<tr>
<td></td>
<td>The possible values are:</td>
</tr>
<tr>
<td></td>
<td>• BASIC — Returns all attributes (such as execution statistics and binds)</td>
</tr>
<tr>
<td></td>
<td>except the plans. The execution context is included in the result.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL — Returns BASIC plus the SQL plan, but without row source</td>
</tr>
<tr>
<td></td>
<td>statistics and without the object reference list. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• ALL — Returns all attributes.</td>
</tr>
<tr>
<td></td>
<td>• Comma-separated list of attribute names. This value enables the</td>
</tr>
<tr>
<td></td>
<td>function to return only a subset of SQL attributes:</td>
</tr>
<tr>
<td></td>
<td>– EXECUTION_STATISTICS</td>
</tr>
<tr>
<td></td>
<td>– SQL_BINDS</td>
</tr>
<tr>
<td></td>
<td>– SQL_PLAN_STATISTICS (similar to SQL_PLAN plus row source</td>
</tr>
<tr>
<td></td>
<td>statistics)</td>
</tr>
</tbody>
</table>
Table 160-44  (Cont.) SELECT_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_filter</td>
<td>Specifies the plan filter. This parameter enables you to select a single plan when a statement has multiple plans. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• LAST_GENERATED — Returns the plan with the most recent timestamp.</td>
</tr>
<tr>
<td></td>
<td>• FIRST_GENERATED — Returns the plan with the least recent timestamp.</td>
</tr>
<tr>
<td></td>
<td>• LAST_LOADED — Returns the plan with the most recent FIRST_LOAD_TIME statistical information.</td>
</tr>
<tr>
<td></td>
<td>• FIRST_LOADED — Returns the plan with the least recent FIRST_LOAD_TIME statistical information.</td>
</tr>
<tr>
<td></td>
<td>• MAX_ELAPSED TIME — Returns the plan with the maximum elapsed time.</td>
</tr>
<tr>
<td></td>
<td>• MAX_BUFFER_GETS — Returns the plan with the maximum buffer gets.</td>
</tr>
<tr>
<td></td>
<td>• MAX_DISK_READS — Returns the plan with the maximum disk reads.</td>
</tr>
<tr>
<td></td>
<td>• MAX_DIRECT_WRITES — Returns the plan with the maximum direct writes.</td>
</tr>
<tr>
<td></td>
<td>• MAX_OPTIMIZER_COST — Returns the plan with the maximum optimizer cost value.</td>
</tr>
<tr>
<td>sqset_owner</td>
<td>Specifies the owner of the SQL tuning set, or NULL for the current schema owner.</td>
</tr>
<tr>
<td>recursive_sql</td>
<td>Specifies that the filter must include recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL, which is the default) or exclude it (NO_RECURSIVE_SQL).</td>
</tr>
</tbody>
</table>

Return Values

This function returns one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Usage Notes

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

Examples

```sql
-- select from a sql tuning set
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
    FROM table(dbms_sqltune.select_sqlset('my_workload')) P;
  -- Process each statement (or pass cursor to load_sqlset)
  CLOSE cur;
END;
/
```
160.5.39 SELECT_WORKLOAD_REPOSITORY Function

This function collects SQL statements from the workload repository. The overloaded forms enable you to collect SQL statements from the following sources:

- Snapshots between \texttt{begin\_snap} and \texttt{end\_snap}
- A workload repository baseline

Syntax

\begin{verbatim}
DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY (
    begin_snap        IN NUMBER,
    end_snap          IN NUMBER,
    basic_filter      IN VARCHAR2 := NULL,
    object_filter     IN VARCHAR2 := NULL,
    ranking_measure1  IN VARCHAR2 := NULL,
    ranking_measure2  IN VARCHAR2 := NULL,
    ranking_measure3  IN VARCHAR2 := NULL,
    result_percentage IN NUMBER   := 1,
    result_limit      IN NUMBER   := NULL,
    attribute_list    IN VARCHAR2 := NULL,
    recursive_sql     IN VARCHAR2 := HAS_RECURSIVE_SQL,
    dbid              IN NUMBER   := NULL)
RETURN sys.sqlset PIPELINED;
\end{verbatim}

\begin{verbatim}
DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY (
    baseline_name     IN VARCHAR2,
    basic_filter      IN VARCHAR2 := NULL,
    object_filter     IN VARCHAR2 := NULL,
    ranking_measure1  IN VARCHAR2 := NULL,
    ranking_measure2  IN VARCHAR2 := NULL,
    ranking_measure3  IN VARCHAR2 := NULL,
    result_percentage IN NUMBER   := 1,
    result_limit      IN NUMBER   := NULL,
    attribute_list    IN VARCHAR2 := NULL,
    recursive_sql     IN VARCHAR2 := HAS_RECURSIVE_SQL,
    dbid              IN NUMBER   := NULL)
RETURN sys.sqlset PIPELINED;
\end{verbatim}

Parameters

Table 160-45  SELECT_WORKLOAD_REPOSITORY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{begin_snap}</td>
<td>Defines the beginning AWR snapshot (non-inclusive).</td>
</tr>
<tr>
<td>\texttt{end_snap}</td>
<td>Defines the ending AWR snapshot (inclusive).</td>
</tr>
<tr>
<td>\texttt{baseline_name}</td>
<td>Specifies the name of the AWR baseline period.</td>
</tr>
</tbody>
</table>
### Table 160-45  (Cont.) SELECT_WORKLOAD_REPOSITORY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic_filter</td>
<td>Specifies the SQL predicate to filter the SQL from the workload repository. The filter is defined on attributes of the SQLSET_ROW. If basic_filter is not set by the caller, then the subprogram captures only statements of type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.</td>
</tr>
<tr>
<td>object_filter</td>
<td>Currently not supported.</td>
</tr>
<tr>
<td>ranking_measure(n)</td>
<td>Defines an ORDER BY clause on the selected SQL.</td>
</tr>
<tr>
<td>result_percentage</td>
<td>Specifies a filter that picks the top n% according to the supplied ranking measure. Note that this percentage applies only if one ranking measure is given.</td>
</tr>
<tr>
<td>result_limit</td>
<td>Specifies the top limit SQL from the source according to the supplied ranking measure.</td>
</tr>
<tr>
<td>attribute_list</td>
<td>Specifies the SQL statement attributes to return in the result. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL — Returns BASIC plus SQL plan (without row source statistics) and without object reference list. This is the default.</td>
</tr>
<tr>
<td></td>
<td>- BASIC — Returns all attributes (such as execution statistics and binds) are returned except the plans. The execution context is always part of the result.</td>
</tr>
<tr>
<td></td>
<td>- ALL — Returns all attributes</td>
</tr>
<tr>
<td></td>
<td>- Comma-separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN_STATISTICS (similar to SQL_PLAN plus row source statistics).</td>
</tr>
<tr>
<td>recursive_sql</td>
<td>Specifies the filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).</td>
</tr>
<tr>
<td>dbid</td>
<td>Specifies the DBID for imported or PDB-level AWR data. If NULL, then the function uses the current database DBID.</td>
</tr>
</tbody>
</table>

### Return Values

This function returns one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

### Usage Notes

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

### Examples

```sql
-- select statements from snapshots 1-2
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
  END;
```
FROM table(dbms_sqltune.select_workload_repository(1,2)) P;

-- Process each statement (or pass cursor to load_sqlset)

CLOSE cur;
END;
/

160.5.40 SET_TUNING_TASK_PARAMETER Procedures

This procedure updates the value of a SQL tuning parameter of type VARCHAR2 or NUMBER.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER (
    task_name        IN  VARCHAR2,
    parameter        IN  VARCHAR2,
    value            IN  VARCHAR2,
    database_link_to IN  VARCHAR2);

DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER (
    task_name        IN  VARCHAR2,
    parameter        IN  VARCHAR2,
    value            IN  NUMBER,
    database_link_to IN  VARCHAR2);
);

Parameters

Table 160-46 SET_TUNING_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Identifier of the task to execute</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parameter</td>
<td>Name of the parameter to set. The possible tuning parameters that can be set by this procedure using the parameter in the form VARCHAR2:</td>
</tr>
<tr>
<td></td>
<td>• APPLY_CAPTURED_COMPILEENV: indicates whether the advisor could use the compilation environment captured with the SQL statements. The default is 0 (that is, NO).</td>
</tr>
<tr>
<td></td>
<td>• BASIC_FILTER: basic filter for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>• DAYS_TO_EXPIRE: number of days until the task is deleted</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT_EXECUTION_TYPE: the task defaults to this type of execution when none is specified by the EXECUTE_TUNING_TASK Function and Procedure</td>
</tr>
<tr>
<td></td>
<td>• EXECUTION_DAYS_TO_EXPIRE: number of days until the task's execution is deleted (without deleting the task)</td>
</tr>
<tr>
<td></td>
<td>• LOCAL_TIME_LIMIT: per-statement time out (seconds)</td>
</tr>
<tr>
<td></td>
<td>• MODE: tuning scope (comprehensive, limited)</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_FILTER: object filter for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>• PLAN_FILTER: plan filter for SQL tuning set (see SELECT_SQLSET for possible values)</td>
</tr>
<tr>
<td></td>
<td>• RANK_MEASURE1: first ranking measure for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>• RANK_MEASURE2: second possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>• RANK_MEASURE3: third possible ranking measure for SQL tuning set</td>
</tr>
<tr>
<td></td>
<td>• RESUME_FILTER: a extra filter for SQL tuning sets besides BASIC_FILTER</td>
</tr>
<tr>
<td></td>
<td>• SQL_LIMIT: maximum number of SQL statements to tune</td>
</tr>
<tr>
<td></td>
<td>• SQL_PERCENTAGE: percentage filter of SQL tuning set statements</td>
</tr>
<tr>
<td></td>
<td>• TEST_EXECUTE: FULL/AUTO/OFF.</td>
</tr>
<tr>
<td></td>
<td>• FULL - test-execute for as much time as necessary, up to the local time limit for the SQL (or the global task time limit if no SQL time limit is set)</td>
</tr>
<tr>
<td></td>
<td>• AUTO - test-execute for an automatically-chosen time proportional to the tuning time</td>
</tr>
<tr>
<td></td>
<td>• OFF - do not test-execute</td>
</tr>
<tr>
<td></td>
<td>• TIME_LIMIT: global time out (seconds)</td>
</tr>
<tr>
<td></td>
<td>• USERNAME: username under which the statement is parsed</td>
</tr>
<tr>
<td>value</td>
<td>New value of the specified parameter</td>
</tr>
</tbody>
</table>
Table 160-46  (Cont.) SET_TUNING_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database_link_to</td>
<td>Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri:</td>
</tr>
<tr>
<td></td>
<td>CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';</td>
</tr>
</tbody>
</table>

Usage Notes

When setting automatic tuning task parameters, use the SET_AUTO_TUNING_TASK_PARAMETER Procedures in the DBMS_AUTO_SQLTUNE package.

160.5.41 SQLTEXT_TO_SIGNATURE Function

This function returns a SQL text's signature. The signature can be used to identify SQL text in dba_sql_profiles.

See Also:

DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```sql
DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE (  
  sql_text    IN CLOB,  
  force_match IN BOOLEAN := FALSE)  
RETURN NUMBER;
```
Parameters

### Table 160-47  SQLTEXT_TO_SIGNATURE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_text</td>
<td>SQL text whose signature is required. Required.</td>
</tr>
<tr>
<td>force_match</td>
<td>If TRUE, this returns a signature that supports SQL matching with literal values transformed into bind variables. If FALSE, returns the signature based on the text with literals not transformed.</td>
</tr>
</tbody>
</table>

Return Values

This function returns the signature of the specified SQL text.

160.5.42 UNPACK_STGTAB_SQLPROF Procedure

This procedure copies profile data stored in the staging table to create profiles on the system.

![See Also:](#)

**DBMS_SQLTUNE SQL Profile Subprograms** for other subprograms in this group

Syntax

```sql
DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF (  
    profile_name          IN VARCHAR2 := '%',  
    profile_category      IN VARCHAR2 := 'DEFAULT',  
    replace               IN BOOLEAN,  
    staging_table_name    IN VARCHAR2,  
    staging_schema_owner  IN VARCHAR2 := NULL);  
```

Parameters

### Table 160-48  UNPACK_STGTAB_SQLPROF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile_name</td>
<td>The name of the profile to unpack (% wildcards acceptable, case-sensitive)</td>
</tr>
<tr>
<td>profile_category</td>
<td>The category from which to unpack profiles (% wildcards acceptable, case-sensitive)</td>
</tr>
<tr>
<td>replace</td>
<td>The option to replace profiles if they already exist. Note that profiles cannot be replaced if one in the staging table has the same name as an active profile in a different SQL statement. If FALSE, this function raises errors if you try to create a profile that already exists.</td>
</tr>
<tr>
<td>staging_table_name</td>
<td>The name of the table on which to perform the remap operation (case-insensitive unless double quoted). Required.</td>
</tr>
</tbody>
</table>
Table 160-48  (Cont.) UNPACK_STGTAB_SQLPROF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>staging_schem</td>
<td>The schema where the table resides, or NULL for current schema (case-</td>
</tr>
<tr>
<td>a_owner</td>
<td>insensitive unless double quoted)</td>
</tr>
</tbody>
</table>

Usage Notes

Using this procedure requires the CREATE ANY SQL PROFILE privilege and the SELECT privilege on staging table.

Examples

-- Unpack all profiles stored in a staging table.
BEGIN
  DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(
    replace => FALSE,
    staging_table_name => 'PROFILE_STGTAB');
END;

-- If there is a failure during the unpack operation, you can find the
-- that caused the error and perform a remap_stgtab_sqlprof operation
-- targeting it.
-- You can resume the unpack operation by setting replace to TRUE so that
-- the profiles that were already created are replaced.
BEGIN
  DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(
    replace => TRUE,
    staging_table_name => 'PROFILE_STGTAB');
END;

160.5.43 UNPACK_STGTAB_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the staging table
into the SQL tuning sets schema, making them proper SQL tuning sets.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this

group

Syntax

DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET (   
  sqllset_name IN VARCHAR2 := '%',
  sqlset_owner IN VARCHAR2 := NULL,
  replace IN BOOLEAN,
staging_table_name IN VARCHAR2,
staging_schema_owner IN VARCHAR2 := NULL);

Parameters

The parameters are identical for DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET and
DBMS_SQLSET.UNPACK_STGTAB.

Table 160-49  UNPACK_STGTAB_SQLSET and UNPACK_STGTAB Procedure
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the tuning set to unpack (not null). Wildcard characters (%) are supported to</td>
</tr>
<tr>
<td></td>
<td>unpack multiple tuning sets in a single call. For example, specify % to unpack all tuning sets from</td>
</tr>
<tr>
<td></td>
<td>the staging table.</td>
</tr>
<tr>
<td>sqlset_owner</td>
<td>Specifies the name of tuning set owner, or NULL for the current schema owner. Wildcard characters (%)</td>
</tr>
<tr>
<td></td>
<td>are supported.</td>
</tr>
<tr>
<td>replace</td>
<td>Specifies whether to replace an existing SQL tuning set. If FALSE, then this procedure raises errors</td>
</tr>
<tr>
<td></td>
<td>when you try to create a tuning set that already exists.</td>
</tr>
<tr>
<td>staging_table</td>
<td>Specifies the name of the staging table, moved after a call to the DBMS_SQLTUNE.PACK_STGTAB_SQLSET</td>
</tr>
<tr>
<td>_name</td>
<td>or DBMS_SQLSET.PACK_STGTAB_SQLSET procedure (case-sensitive).</td>
</tr>
<tr>
<td>staging_schema</td>
<td>Specifies the name of staging table owner, or NULL for the current schema owner (case-sensitive).</td>
</tr>
<tr>
<td>owner</td>
<td></td>
</tr>
</tbody>
</table>

Examples

-- unpack all STS in the staging table
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name         => '%', -
                                        sqlset_owner        => '%', -
                                        replace             => FALSE, -
                                        staging_table_name  => 'STGTAB_SQLSET');

-- errors can arise during STS unpack when a STS in the staging table has the
-- same name/owner as STS on the system. In this case, users should call
-- remap_stgtab_sqlset to patch the staging table and with which to call unpack
-- Replace set to TRUE.
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name         => '%', -
                                        sqlset_owner        => '%', -
                                        replace             => TRUE, -
                                        staging_table_name  => 'STGTAB_SQLSET');
160.5.44 UPDATE_SQLSET Procedures

This overloaded procedure updates selected fields for SQL statements in a SQL tuning set.

See Also:

DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

DBMS_SQLTUNE.UPDATE_SQLSET (sqlset_name IN VARCHAR2, sql_id IN VARCHAR2, attribute_name IN VARCHAR2, attribute_value IN VARCHAR2 := NULL);

DBMS_SQLTUNE.UPDATE_SQLSET (sqlset_name IN VARCHAR2, sql_id IN VARCHAR2, attribute_name IN VARCHAR2, attribute_value IN NUMBER := NULL);

Parameters

Table 160-50  UPDATE_SQLSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Specifies the name of the SQL tuning set.</td>
</tr>
<tr>
<td>sql_id</td>
<td>Specifies the identifier of the SQL statement to be updated.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Specifies the hash value of the execution plan for a SQL statement.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Specifies the name of the attribute to be modified.</td>
</tr>
<tr>
<td>attribute_value</td>
<td>Specifies the new value of the attribute.</td>
</tr>
</tbody>
</table>

You can update the text field for MODULE, ACTION, PARSING_SCHEMA_NAME, and OTHER. The only numerical field that you can update is PRIORITY. If a statement has multiple plans, then the procedure changes the attribute value for all plans.
The DBMS_STAT_FUNCS package provides statistical functions.

This chapter contains the following topic:

• Summary of DBMS_STAT_FUNCS Subprograms

161.1 Summary of DBMS_STAT_FUNCS Subprograms

This table lists and briefly describes the DBMS_STAT_FUNCS subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPONENTIAL_DIST_FIT</td>
<td>Tests how well a sample of values fits an exponential distribution</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>NORMAL_DIST_FIT</td>
<td>Tests how well a sample of values fits a normal distribution</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>POISSON_DIST_FIT</td>
<td>Tests how well a sample of values fits a Poisson distribution</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>SUMMARY Procedure</td>
<td>Summarizes a numerical column of a table</td>
</tr>
<tr>
<td>UNIFORM_DIST_FIT</td>
<td>Tests how well a sample of values fits a uniform distribution</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
<tr>
<td>WEIBULL_DIST_FIT</td>
<td>Tests how well a sample of values fits a Weibull distribution</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
</tbody>
</table>

161.1.1 EXPONENTIAL_DIST_FIT Procedure

This procedure tests how well a sample of values fits an exponential distribution.

Syntax

DBMS_STAT_FUNCS.EXPONENTIAL_DIST_FIT (  
  ownername    IN    VARCHAR2,  
  tablename    IN    VARCHAR2,  
  columnname   IN    VARCHAR2,  
  test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIROV',  
  lambda       IN    NUMBER,  
  mu           IN    NUMBER,  
  sig          OUT   NUMBER);  

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>The schema where the table resides.</td>
</tr>
</tbody>
</table>
Table 161-2  (Cont.) EXPONENTIAL_DIST_FIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>The table where the column resides.</td>
</tr>
<tr>
<td>columnname</td>
<td>The column of the table against which to run the test.</td>
</tr>
<tr>
<td>test_type</td>
<td>The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV', 'ANDERSON_DARLING'.</td>
</tr>
<tr>
<td>lambda</td>
<td>The scale parameter.</td>
</tr>
<tr>
<td>mu</td>
<td>The location parameter.</td>
</tr>
<tr>
<td>sig</td>
<td>The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the exponential distribution. A number close to 1 indicates a close match.</td>
</tr>
</tbody>
</table>

161.1.2 NORMAL_DIST_FIT Procedure

This procedure tests how well a sample of values fits a normal distribution.

**Syntax**

```sql
DBMS_STAT_FUNCS.NORMAL_DIST_FIT (
    ownername    IN    VARCHAR2,
    tablename    IN    VARCHAR2,
    columnname   IN    VARCHAR2,
    test_type    IN    VARCHAR2 DEFAULT 'SHAPIRO_WILKS',
    mean         IN    NUMBER,
    stdev        IN    NUMBER,
    sig          OUT   NUMBER);
```

**Parameters**

Table 161-3  NORMAL_DIST_FIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>The schema where the table resides.</td>
</tr>
<tr>
<td>tablename</td>
<td>The table where the column resides.</td>
</tr>
<tr>
<td>columnname</td>
<td>The column of the table against which to run the test.</td>
</tr>
<tr>
<td>test_type</td>
<td>The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV', 'ANDERSON_DARLING', 'SHAPIRO_WILKS'.</td>
</tr>
<tr>
<td>mean</td>
<td>The mean of the distribution against which to compare.</td>
</tr>
<tr>
<td>stdev</td>
<td>The standard deviation of the distribution against which to compare.</td>
</tr>
<tr>
<td>sig</td>
<td>The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the normal distribution. A number close to 1 indicates a close match.</td>
</tr>
</tbody>
</table>
161.3 POISSON_DIST_FIT Procedure

This procedure tests how well a sample of values fits a Poisson distribution.

Syntax

```sql
DBMS_STAT_FUNCS.POISSON_DIST_FIT (  
    ownername    IN    VARCHAR2,  
    tablename    IN    VARCHAR2,  
    columnname   IN    VARCHAR2,  
    test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',  
    lambda       IN    NUMBER,  
    sig          OUT   NUMBER);
```

Parameters

**Table 161-4** POISSON_DIST_FIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>The schema where the table resides.</td>
</tr>
<tr>
<td>tablename</td>
<td>The table where the column resides.</td>
</tr>
<tr>
<td>columnname</td>
<td>The column of the table against which to run the test.</td>
</tr>
<tr>
<td>test_type</td>
<td>The type of test to use: 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.</td>
</tr>
<tr>
<td>lambda</td>
<td>The lambda parameter is the shape parameter.</td>
</tr>
<tr>
<td>sig</td>
<td>The goodness of fit value, based on test type. A small value indicates a</td>
</tr>
<tr>
<td></td>
<td>significant difference between the sample and the Poisson distribution.</td>
</tr>
<tr>
<td></td>
<td>A number close to 1 indicates a close match.</td>
</tr>
</tbody>
</table>

161.4 SUMMARY Procedure

This procedure summarizes the numerical column specified in the `columnname` of `tablename`.

The summary is returned as a Summary Type. Note that most of the output of `SUMMARY` can be obtained with currently available SQL.

Syntax

```sql
DBMS_STAT_FUNCS.SUMMARY (  
    ownername    IN    VARCHAR2,  
    tablename    IN    VARCHAR2,  
    columnname   IN    VARCHAR2,  
    sigma_value  IN    NUMBER DEFAULT 3,  
    s            OUT   SummaryType);
```

Parameters

**Table 161-5** SUMMARY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>The schema where the table resides.</td>
</tr>
</tbody>
</table>
Table 161-5  (Cont.) SUMMARY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>The table where the column resides.</td>
</tr>
<tr>
<td>columnname</td>
<td>The column of the table to be summarized.</td>
</tr>
<tr>
<td>sigma_value</td>
<td>The number of sigmas for the set of extreme values, defaults to 3.</td>
</tr>
<tr>
<td>s</td>
<td>The Record containing summary information about given column.</td>
</tr>
</tbody>
</table>

Definition of SummaryType

```sql
TYPE n_arr IS VARRAY(5) of NUMBER;
TYPE num_table IS TABLE of NUMBER;
TYPE summaryType IS RECORD (
  count NUMBER,
  min   NUMBER,
  max   NUMBER,
  range NUMBER,
  mean  NUMBER,
  cmode num_table,
  variance NUMBER,
  stddev NUMBER,
  quantile_5 NUMBER,
  quantile_25 NUMBER,
  median NUMBER,
  quantile_75 NUMBER,
  quantile_95 NUMBER,
  plus_x_sigma NUMBER,
  minus_x_sigma NUMBER,
  extreme_values num_table,
  top_5_values n_arr,
  bottom_5_values n_arr);
```

161.1.5 UNIFORM_DIST_FIT Procedure

This procedure tests well a sample of values fits a uniform distribution.

Syntax

```sql
DBMS_STAT_FUNCS.UNIFORM_DIST_FIT (  
  ownername    IN    VARCHAR2,  
  tablename    IN    VARCHAR2,  
  columnname   IN    VARCHAR2,  
  var_type     IN    VARCHAR2 DEFAULT 'CONTINUOUS',  
  test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',  
  paramA       IN    NUMBER,  
  paramB       IN    NUMBER,  
  sig          OUT   NUMBER);
```
Parameters

161.1.6 WEIBULL_DIST_FIT Procedure

This procedure tests how well a sample of values fits a Weibull distribution.

Syntax

```sql
DBMS_STAT_FUNCS.WEIBULL_DIST_FIT (  
    ownername    IN    VARCHAR2,  
    tablename    IN    VARCHAR2,  
    columnname   IN    VARCHAR2,  
    test_type    IN    VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',  
    alpha        IN    NUMBER,  
    mu           IN    NUMBER,  
    beta         IN    NUMBER,  
    sig          OUT   NUMBER);  
```

Parameters

Table 161-7  WEIBULL_DIST_FIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>The schema where the table resides.</td>
</tr>
<tr>
<td>tablename</td>
<td>The table where the column resides.</td>
</tr>
<tr>
<td>columnname</td>
<td>The column of the table against which to run the test.</td>
</tr>
<tr>
<td>test_type</td>
<td>The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.</td>
</tr>
<tr>
<td>alpha</td>
<td>The scale parameter.</td>
</tr>
<tr>
<td>mu</td>
<td>The location parameter.</td>
</tr>
<tr>
<td>beta</td>
<td>The slope/shape parameter.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>sig</td>
<td>The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the Weibull distribution. A number close to 1 indicates a close match.</td>
</tr>
</tbody>
</table>
With the `DBMS_STATS` package you can view and modify optimizer statistics gathered for database objects.

Users can also collect statistics on Global Temporary Tables (GTTs) using the `DBMS_STATS` package. However, `DBMS_STATS` cannot collect statistics on Private Temporary Tables (PTTs).

This chapter contains the following topics:

- DBMS_STATS Overview
- DBMS_STATS Deprecated Subprograms
- DBMS_STATS Types
- DBMS_STATS Constants
- DBMS_STATS Operational Notes
- DBMS_STATS Data Structures
- Summary of DBMS_STATS Subprograms

**See Also:**

- Oracle Database SQL Tuning Guide
- Oracle Database Administrator's Guide

### 162.1 DBMS_STATS Overview

To improve performance, the database enables you to collect statistics of many different types. This package is concerned with optimizer statistics only. By default, the database collects statistics automatically, so this package is intended only for specialized cases.

Optimizer statistics can reside in the data dictionary or in a table created in the user's schema. You can also collect and manage user-defined statistics for tables and domain indexes using this package. For example, if you invoke the `DELETE_COLUMN_STATS` procedure on a column for which an association is defined, the database deletes both user-defined and standard statistics for this column.

Only statistics stored in the dictionary have an effect on the cost-based optimizer. You can also use `DBMS_STATS` to gather statistics in parallel.

Starting in Oracle Database 12c Release 2 (12.2), Optimizer Statistics Advisor inspects the statistics gathering process, automatically diagnoses problems in the existing practices for gathering statistics, and then generates a report of findings and recommendations. The advisor task runs automatically in the maintenance window, but you can also run it on demand.
162.2 DBMS_STATS Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms. Support for deprecated features is for backward compatibility only.

Starting with Oracle Database 11g Release 2 (11.2), the following subprograms are obsolete:

- **GET_PARAM Function**
  Instead, use **GET_PREFS Function**
- **SET_PARAM Procedure**
  Instead, use **SET_GLOBAL_PREFS Procedure**
- **RESET_PARAM_DEFAULTS Procedure**
  Instead use **RESET_GLOBAL_PREF_DEFAULTS Procedure**

The following subprogram is deprecated with Oracle Database 12c and later:

- **GENERATE_STATS**
  This procedure is replaced by the **GATHER_INDEX_STAT procedure**.

162.3 DBMS_STATS Types

### Histograms

Types for the minimum and maximum values and histogram endpoints include the following:

```
TYPE numarray  IS VARRAY(2050) OF NUMBER;
TYPE datearray IS VARRAY(2050) OF DATE;
TYPE chararray IS VARRAY(2050) OF VARCHAR2(4000);
TYPE rawarray  IS VARRAY(2050) OF RAW(2000);
TYPE fltarray  IS VARRAY(2050) OF BINARY_FLOAT;
TYPE dblarray  IS VARRAY(2050) OF BINARY_DOUBLE;
```

### Stale Tables

Types for listing stale tables include the following:

```
TYPE ObjectElem IS RECORD {
    ownname     VARCHAR2(30),     -- owner
}
```
Statistics Difference Reports

Use the following type to display a statistics difference report:

```
TYPE DiffRepElem IS RECORD (  
  report      CLOB,            -- stats difference report  
  maxdiffpct  NUMBER);         -- max stats difference (percentage)
```

```
TYPE DiffRepTab IS TABLE OF DiffRepElem;
```

Optimizer Statistics Advisor

The following type represents database objects for which you can gather statistics:

```
TYPE ObjectElem IS RECORD (  
  ownname     dbms_quoted_id, -- owner  
  objtype     VARCHAR2(6), -- 'TABLE' or 'INDEX'  
  objname     dbms_quoted_id, -- table/index  
  partname    dbms_quoted_id, -- partition  
  subpartname dbms_quoted_id -- subpartition
);
```

```
TYPE ObjectTab IS TABLE OF ObjectElem;
```

Note:

Make sure to maintain `satisfy_obj_filter` when the `ObjectElem` type is changed

The following type represents an operation:

```
TYPE StatsAdvOpr IS RECORD (  
  name   VARCHAR2(64), -- name of the operation  
  param  VARCHAR2(4000)  
); -- XML containing parameters and their values
```

```
TYPE StatsAdvOprTab IS TABLE OF StatsAdvOpr;
```

The following type represents a filter list:

```
TYPE StatsAdvFilter IS RECORD (  
  rulename  VARCHAR2(64), -- rule name  
  objlist   ObjectTab, -- object filter list  
  oprlist   StatsAdvOprTab, -- operation filter list  
  include  BOOLEAN); -- include/exclude elements in the list
```

```
TYPE StatsAdvFilterTab IS TABLE OF StatsAdvFilter;
```
162.4 DBMS_STATS Constants

The `DBMS_STATS` package defines several constants to use specifying parameter values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_GLOBAL_PREFS</td>
<td>NUMBER</td>
<td>Copies global preferences</td>
</tr>
<tr>
<td>AUTOCASCADE</td>
<td>BOOLEAN</td>
<td>Lets Oracle decide whether to collect statistics for indexes or not</td>
</tr>
<tr>
<td>AUTODEGREE</td>
<td>NUMBER</td>
<td>Lets Oracle select the degree of parallelism based on size of the object, number of CPUs and initialization parameters</td>
</tr>
<tr>
<td>AUTOINVALIDATE</td>
<td>BOOLEAN</td>
<td>Lets Oracle decide when to invalidate dependent cursors</td>
</tr>
<tr>
<td>AUTOSAMPLE_SIZE</td>
<td>NUMBER</td>
<td>Indicates that auto-sample size algorithms should be used</td>
</tr>
<tr>
<td>PURGE_ALL</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>A flag that can be passed to the <code>PURGE_STATS Procedure</code> and unconditionally deletes all the history statistics. The deletion uses TRUNCATE statements on the various dictionary statistics tables holding the history of statistics.</td>
</tr>
<tr>
<td>RECLAIM_SYNOPSIS</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>A constant used for reclaiming synopsis table space.</td>
</tr>
</tbody>
</table>

162.5 DBMS_STATS Operational Notes

Observe these operational notes when using the `DBMS_STATS` package.

The `DBMS_STATS` subprograms perform the following general operations:

- Gathering Optimizer Statistics
- Setting or Getting Statistics
- Deleting Statistics
- Transferring Statistics
- Locking or Unlocking Statistics
- Restoring and Purging Statistics History
- User-Defined Statistics
- Pending Statistics
- Comparing Statistics
- Extended Statistics
- Optimizer Statistics Advisor
Most of the `DBMS_STATS` procedures include the three parameters `statown`, `stattab`, and `statid`. These parameters enable you to store statistics in your own tables (outside of the dictionary), which does not affect the optimizer. Therefore, you can maintain and experiment with sets of statistics.

The `stattab` parameter specifies the name of a table in which to hold statistics, and it is assumed that it resides in the same schema as the object for which statistics are collected (unless the `statown` parameter is specified). You can create multiple tables with different `stattab` identifiers to hold separate sets of statistics.

Additionally, you can maintain different sets of statistics within a single `stattab` by using the `statid` parameter, which avoids cluttering the user's schema.

For the `SET` and `GET` procedures, if `stattab` is not provided (that is, `NULL`), then the operation works directly on the dictionary statistics; therefore, you do not need to create these statistics tables if they only plan to modify the dictionary directly. However, if `stattab` is not `NULL`, then the `SET` or `GET` operation works on the specified user statistics table, and not the dictionary.

You can change the default values of some of the parameters of `DBMS_STATS` procedures using the `SET_DATABASE_PREFS` Procedure, `SET_GLOBAL_PREFS` Procedure, `SET_SCHEMA_PREFS` Procedure and `SET_TABLE_PREFS` Procedure.

Most procedures in this package commit the current transaction, perform the operation, and then commit again.

Most of the procedures have a `force` parameter that enables you to override a lock on statistics. Whenever statistics in dictionary are modified, old versions of statistics are saved automatically for future restoring.

Gathering Optimizer Statistics

Use the following subprograms to gather certain classes of optimizer statistics, with possible performance improvements over the `ANALYZE` command:

- `GATHER_DATABASE_STATS` Procedures
- `GATHER_DICTIONARY_STATS` Procedure
- `GATHER_FIXED_OBJECTS_STATS` Procedure
- `GATHER_INDEX_STATS` Procedure
- `GATHER_SCHEMA_STATS` Procedures
- `GATHER_SYSTEM_STATS` Procedure
- `GATHER_TABLE_STATS` Procedure

The `GATHER_*` procedures also collect user-defined statistics for columns and domain indexes.

The `statown`, `stattab`, and `statid` parameters instruct the package to back up current statistics in the specified table before gathering new statistics.

Oracle also provides the following procedure for generating statistics for derived objects when you have sufficient statistics on related objects:

- `GENERATE_STATS` Procedure

Setting or Getting Statistics

Use the following subprograms to store and retrieve individual column-related, index-related, and table-related statistics:
In the special versions of the `SET_` _stats_ procedures for setting user-defined statistics, the following, if provided, are stored in the dictionary or user statistics table:

- User-defined statistics
- Owner of statistics type
- Name of statistics type

The user-defined statistics and the corresponding statistics type are inserted into the `USTATS$` dictionary table. You can specify user-defined statistics without specifying the statistics type name.

The special versions of the `GET_` _stats_ procedures return user-defined statistics and the statistics type owner and name as `OUT` arguments corresponding to the schema object specified. If user-defined statistics are not collected, `NULL` values are returned.

### Deleting Statistics

The `DELETE_` _stats_ procedures delete both user-defined statistics and the standard statistics for the given schema object.

- `DELETE_COLUMN_STATS Procedure`
- `DELETE_DATABASE_STATS Procedure`
- `DELETE_DICTIONARY_STATS Procedure`
- `DELETE_FIXED_OBJECTS_STATS Procedure`
- `DELETE_INDEX_STATS Procedure`
- `DELETE_SCHEMA_STATS Procedure`
- `DELETE_SYSTEM_STATS Procedure`
- `DELETE_TABLE_STATS Procedure`

Note that `DELETE_TABLE_STATS`, `DELETE_DICTIONARY_STATS`, `DELETE_DATABASE_STATS`, and `DELETE_SCHEMA_STATS` have a parameter `stat_category` which specifies which statistics to delete. The parameter accepts multiple values separated by comma. The supported values are `'OBJECT_STATS'` (table statistics, column statistics and index statistics) and `'SYNOPSES'` (auxiliary statistics created when statistics are incrementally maintained). The default is `'OBJECT_STATS, SYNOPSES'`.

### Transferring Statistics

Use the following procedures for creating and dropping the user statistics table.

- `CREATE_STAT_TABLE Procedure`
DROP_STAT_TABLE Procedure

Use the following procedures to transfer statistics

- from the dictionary to a user statistics table (**EXPORT_***)
- from a user statistics table to the dictionary (**IMPORT_***)

**EXPORT_COLUMN_STATS Procedure**
**EXPORT_DATABASE_STATS Procedure**
**EXPORT_DICTIONARY_STATS Procedure**
**EXPORT_FIXED_OBJECTS_STATS Procedure**
**EXPORT_INDEX_STATS Procedure**
**EXPORT_SCHEMA_STATS Procedure**
**EXPORT_SYSTEM_STATS Procedure**
**EXPORT_TABLE_STATS Procedure**

**IMPORT_COLUMN_STATS Procedure**
**IMPORT_DATABASE_STATS Procedure**
**IMPORT_DICTIONARY_STATS Procedure**
**IMPORT_FIXED_OBJECTS_STATS Procedure**
**IMPORT_INDEX_STATS Procedure**
**IMPORT_SCHEMA_STATS Procedure**
**IMPORT_SYSTEM_STATS Procedure**
**IMPORT_TABLE_STATS Procedure**

**Note:**

Oracle does not support export or import of statistics across databases of different character sets.

Locking or Unlocking Statistics

Use the following procedures to lock and unlock statistics on objects.

**LOCK_PARTITION_STATS Procedure**
**LOCK_SCHEMA_STATS Procedure**
**LOCK_TABLE_STATS Procedure**

**UNLOCK_PARTITION_STATS Procedure**
**UNLOCK_SCHEMA_STATS Procedure**
**UNLOCK_TABLE_STATS Procedure**

The **LOCK_*** procedures either freeze the current set of the statistics or to keep the statistics untouched. When statistics on a table are locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.

Restoring and Purging Statistics History

Use the following procedures to restore statistics as of a specified timestamp. This is useful in case newly collected statistics leads to some sub-optimal execution plans and the administrator wants to revert to the previous set of statistics.

**RESET_GLOBAL_PREF_DEFAULTS Procedure**
Whenever statistics in dictionary are modified, old versions of statistics are saved automatically for future restoring. The old statistics are purged automatically at regular intervals based on the statistics history retention setting and the time of recent statistics gathering performed in the system. Retention is configurable using the `ALTER_STATS_HISTORY_RETENTION Procedure`.

The other `DBMS_STATS` procedures related to restoring statistics are:

- **PURGE_STATS Procedure**: This procedure lets you manually purge old versions beyond a time stamp.
- **GET_STATS_HISTORY_RETENTION Function**: This function gets the current statistics history retention value.
- **GET_STATS_HISTORY_AVAILABILITY Function**: This function gets the oldest time stamp where statistics history is available. Users cannot restore statistics to a time stamp older than the oldest time stamp.

`RESTORE_*` operations are not supported for user defined statistics.

**User-Defined Statistics**

The `DBMS_STATS` package supports operations on user-defined statistics. When a domain index or column is associated with a statistics type (using the `associate` statement), operations on the index or column manipulate user-defined statistics. For example, gathering statistics for a domain index (for which an association with a statistics type exists) using the `GET_INDEX_STATS Procedures` invokes the user-defined statistics collection method of the associated statistics type. Similarly, delete, transfer, import, and export operations manipulate user-defined statistics.

`SET_*` and `GET_*` operations for user-defined statistics are also supported using a special version of the `SET` and `GET` interfaces for columns and indexes.

`EXPORT_*`, `IMPORT_*`, and `RESTORE_*` operations are not supported for user defined statistics.

**Pending Statistics**

The package gathers statistics and stores it in the dictionary by default. User's can store these statistics in the system's private area instead of the dictionary by turning the `PUBLISH` option to `FALSE` using the `SET*PREFS` procedures. The default value for `PUBLISH` is `TRUE`. The statistics stored in private area are not used by Cost Based Optimizer unless parameter `optimizer_use_pending_statistics` is set to `TRUE`. The default value of this parameter is `FALSE` and this boolean parameter can be set at the session/system level. Users can verify the impact of the new statistics on query plans by using the pending statistics on a session.

Pending statistics provide a mechanism to verify the impact of the new statistics on query plans before making them available for general use. There are two scenarios to verify the query plans:
• Export the pending statistics (use the EXPORT_PENDING_STATS Procedure) to a test system, then run the query workload and check the performance or plans.

• Set optimizer_use_pending_statistics to TRUE in a session on the system where pending statistics have been gathered, run the workload, and check the performance or plans.

After the performance or query plans have been verified, you can publish the pending statistics using the PUBLISH_PENDING_STATS Procedure if the performance is acceptable, or delete the pending statistics using DELETE_PENDING_STATS Procedure if it is not.

Pending statistics can be published, exported, or deleted. The following procedures are provided to manage pending statistics:

• DELETE_PENDING_STATS Procedure
• EXPORT_PENDING_STATS Procedure
• PUBLISH_PENDING_STATS Procedure

Comparing Statistics

You can use the DIFF_TABLE_STATS_* statistics to compare statistics for a table from two different sources. The statistics can be from:

• Two different user statistics tables
• A single user statistics table containing two sets of statistics that can be identified using statids
• A user statistics table and dictionary history
• Pending statistics

The functions also compare the statistics of the dependent objects (indexes, columns, partitions). They display statistics of the objects from both sources if the difference between those statistics exceeds a certain threshold. The threshold can be specified as an argument to the function, with a default of 10%. The database uses the statistics corresponding to the first source (stattab1 or timel) as a basis for computing the differential percentage.

Extended Statistics

This package enables you to collect statistics for column groups and expressions. The statistics collected for column groups and expressions are called "extended statistics".

Statistics on column groups are used by optimizer for accounting correlation between columns. For example, if a query has predicates c1=1 and c2=1 and if there are statistics on column group (c1, c2), the optimizer uses these statistics for estimating the combined cardinality of the predicates. The optimizer uses the expression statistics to estimate cardinality of predicates on those expressions. The extended statistics are similar to column statistics. The procedures that take columns names accept extended statistics names in place of column names.

Related subprograms:

• CREATE_EXTENDED_STATS Function
• DROP_EXTENDED_STATS Procedure
• SHOW_EXTENDED_STATS_NAME Function
• REPORT_COL_USAGE Function
• SEED_COL_USAGE Procedure
Optimizer Statistics Advisor

Optimizer Statistics Advisor is built-in diagnostic software that helps use to use best practices to manage optimizer statistics. The advisor analyzes how you are currently gathering statistics (using manual scripts, explicitly setting parameters, and so on), the effectiveness of existing statistics gathering jobs, and the quality of the gathered statistics. The advisor generates findings for any issues it finds. Based on these findings, the advisor provides recommendations, which it stores in DBA_ADVISOR_RECOMMENDATIONS.

The advisor organizes rules into the following classes:

- **System**
  - This class checks the preferences for statistics collection, status of the automated statistics gathering job, use of SQL plan directives, and so on. Rules in this class have the value `SYSTEM` in `V$STATS_ADVISOR_RULES.RULE_TYPE`.

- **Operation**
  - This class checks whether statistics collection uses the defaults, test statistics are created using the `SET_.*_STATS` procedures, and so on. Rules in this class have the value `OPERATION` in `V$STATS_ADVISOR_RULES.RULE_TYPE`.

- **Object**
  - This class checks for the quality of the statistics, staleness of statistics, unnecessary collection of statistics, and so on. Rules in this class have the value `OBJECT` in `V$STATS_ADVISOR_RULES.RULE_TYPE`.

All Optimizer Statistics Advisor subprograms require the `ADVISOR` privilege. All procedures and functions execute using the invoker's privilege for the operation instead of the task owner's privilege. For example, if a user without the `ANALYZE ANY DICTIONARY` privilege creates a task `t1`, and if a DBA then executes this task, then the task execution checks for SYS objects. Another example is a task that is executed by `user1`, interrupted, and then resumed by `user2`. In this case, the checks of the resumed execution are based on the privilege of `user2` rather than `user1`.

You can use the following subprograms to manage Optimizer Statistics Advisor:

- **CANCEL_ADVISOR_TASK Procedure**
- **CONFIGURE_ADVISOR_FILTER Function**
- **CONFIGURE_ADVISOR_OBJ_FILTER Function**
- **CONFIGURE_ADVISOR_OPR_FILTER Functions**
- **CONFIGURE_ADVISOR_RULE_FILTER Function**
- **CREATE_ADVISOR_TASK Function**
- **DROP_ADVISOR_TASK Procedure**
- **EXECUTE_ADVISOR_TASK Function**
- **GET_ADVISOR_OPR_FILTER Procedure**
- **GET_ADVISOR_RECS Function**
- **IMPLEMENT_ADVISOR_TASK Function**
162.6 DBMS_STATS Data Structures

The DBMS_STATS package defines a RECORD type.

RECORD Types

• STAT_REC Record Type

162.6.1 DBMS_STATS STAT_REC Record Type

This record type is provided for users in case they want to set column statistics manually. Its fields allow specifying column min/max values, as well as a histogram for a column.

Syntax

```
TYPE STATREC IS RECORD (
  epc    NUMBER,
  minval RAW(2000),
  maxval RAW(2000),
  bkvals NUMARRAY,
  novals NUMARRAY,
  chvals CHARARRAY,
  eavals RAWARRAY,
  rpcnts NUMARRAY,
  eavs   NUMBER);
```

Fields of the Record type COMPARISON_TYPE (STAT_REC Attributes)

Table 162-2  STAT_REC Attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>epc</td>
<td>Number of buckets in histogram</td>
</tr>
<tr>
<td>minval</td>
<td>Minimum value</td>
</tr>
<tr>
<td>maxval</td>
<td>Maximum value</td>
</tr>
<tr>
<td>bkvals</td>
<td>Array of bucket numbers</td>
</tr>
</tbody>
</table>
### 162.7 Summary of DBMS_STATS Subprograms

This table lists the DBMS_STATS subprograms and briefly describes them.

#### Table 162-3 DBMS_STATS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_STATS_HISTORIES_RETENTION Procedure</td>
<td>Changes the statistics history retention value</td>
</tr>
<tr>
<td>CANCEL_ADVISOR_TASK Procedure</td>
<td>Cancels an Optimizer Statistics Advisor execution</td>
</tr>
<tr>
<td>CONFIGURE_ADVISOR_FILTER Function</td>
<td>Configures the filter list for an Optimizer Statistics Advisor task</td>
</tr>
<tr>
<td>CONFIGURE_ADVISOR_OBJ_FILTER Function</td>
<td>Configures an object filter for an Optimizer Statistics Advisor task</td>
</tr>
<tr>
<td>CONFIGURE_ADVISOR_OPR_FILTER Functions</td>
<td>Configures an operation filter for an Optimizer Statistics Advisor task</td>
</tr>
<tr>
<td>CONFIGURE_ADVISOR_RULE_FILTER Function</td>
<td>Configures a rule filter for an Optimizer Statistics Advisor task</td>
</tr>
<tr>
<td>CREATE_ADVISOR_TASK Function</td>
<td>Creates an advisor task for the Optimizer Statistics Advisor</td>
</tr>
<tr>
<td>CONVERT_RAW_VALUE Procedures</td>
<td>Converts the internal representation of a minimum value, maximum value, or histogram endpoint actual value into a datatype-specific value</td>
</tr>
<tr>
<td>CONVERT_RAW_VALUE_NVARCHAR Procedure</td>
<td>Converts the internal representation of a minimum value, maximum value, or histogram endpoint actual value into a datatype-specific value</td>
</tr>
<tr>
<td>CONVERT_RAW_VALUE_ROWID Procedure</td>
<td>Converts the internal representation of a minimum value, maximum value, or histogram endpoint actual value into a datatype-specific value</td>
</tr>
<tr>
<td>COPY_TABLE_STATS Procedure</td>
<td>Copies the statistics of the source [sub] partition to the destination [sub] partition after scaling</td>
</tr>
<tr>
<td>CREATE_EXTENDEDSTATS Function</td>
<td>Creates a virtual column for a user specified column group or an expression in a table</td>
</tr>
<tr>
<td>CREATE_STAT_TABLE Procedure</td>
<td>Creates a table with name \texttt{stattab} in \texttt{ownname}'s schema which is capable of holding statistics</td>
</tr>
</tbody>
</table>

---

Table 162-2 (Cont.) STAT_REC Attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>novals</td>
<td>Array of normalized end point values</td>
</tr>
<tr>
<td>chvals</td>
<td>Array of dumped end point values</td>
</tr>
<tr>
<td>eavals</td>
<td>Array of end point actual values</td>
</tr>
<tr>
<td>rpcnts</td>
<td>Array of end point value frequencies</td>
</tr>
<tr>
<td>eavs</td>
<td>A number indicating whether actual end point values are needed in the histogram. If using the \texttt{PREPARE_COLUMN_VALUES} Procedures, this field will be automatically filled.</td>
</tr>
</tbody>
</table>
### Table 162-3  (Cont.) DBMS_STATS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE_COLUMN_STATS Procedure</td>
<td>Deletes column-related statistics</td>
</tr>
<tr>
<td>DELETE_DATABASE_PREFERENCES Procedure</td>
<td>Deletes the statistics preferences of all the tables</td>
</tr>
<tr>
<td>DELETE_DATABASE_STATS Procedure</td>
<td>Deletes statistics for the entire database</td>
</tr>
<tr>
<td>DELETE_DICTIONARY_STATS Procedure</td>
<td>Deletes statistics for all dictionary schemas (SYS,'SYSTEM' and database component schemas)</td>
</tr>
<tr>
<td>DELETE_FIXED_OBJECTS_STATS Procedure</td>
<td>Deletes statistics of all fixed tables</td>
</tr>
<tr>
<td>DELETE_INDEX_STATS Procedure</td>
<td>Deletes index-related statistics</td>
</tr>
<tr>
<td>DELETE_PENDING_STATS Procedure</td>
<td>Deletes the private statistics that have been collected but have not been published</td>
</tr>
<tr>
<td>DELETE_PROCESSING_RATE Procedure</td>
<td>Deletes the processing rate of a given statistics source. If the source is not specified, it deletes the statistics of all the sources</td>
</tr>
<tr>
<td>DELETE_SCHEMA_PREFERENCES Procedure</td>
<td>Deletes the statistics preferences of all the tables owned by the specified owner name</td>
</tr>
<tr>
<td>DELETE_SCHEMA_STATS Procedure</td>
<td>Deletes schema-related statistics</td>
</tr>
<tr>
<td>DELETE_SYSTEM_STATS Procedure</td>
<td>Deletes system statistics</td>
</tr>
<tr>
<td>DELETE_TABLE_PREFERENCES Procedure</td>
<td>Deletes statistics preferences of the specified table in the specified schema</td>
</tr>
<tr>
<td>DELETE_TABLE_STATS Procedure</td>
<td>Deletes table-related statistics</td>
</tr>
<tr>
<td>DIFF_TABLE_STATS_IN_HISTORY Function</td>
<td>Compares statistics for a table from two timestamps in past and compare the statistics as of that timestamps</td>
</tr>
<tr>
<td>DIFF_TABLE_STATS_IN_PENDING Function</td>
<td>Compares pending statistics and statistics as of a timestamp or statistics from dictionary</td>
</tr>
<tr>
<td>DIFF_TABLE_STATS_IN_STATTAB Function</td>
<td>Compares statistics for a table from two different sources</td>
</tr>
<tr>
<td>DROP_ADVISOR_TASK Procedure</td>
<td>Drops the specified Optimizer Statistics Advisor task</td>
</tr>
<tr>
<td>DROP_EXTENDED_STATS Procedure</td>
<td>Drops the statistics entry that is created for the user specified extension</td>
</tr>
<tr>
<td>DROP_STAT_TABLE Procedure</td>
<td>Drops a user statistics table created by CREATE_STAT_TABLE</td>
</tr>
<tr>
<td>EXECUTE_ADVISOR_TASK Function</td>
<td>Executes a previously created Optimizer Statistics Advisor task</td>
</tr>
<tr>
<td>EXPORT_COLUMN_STATS Procedure</td>
<td>Retrieves statistics for a particular column and stores them in the user statistics table identified by stattab</td>
</tr>
<tr>
<td>EXPORT_DATABASE_PREFERENCES Procedure</td>
<td>Exports the statistics preferences of all the tables</td>
</tr>
<tr>
<td>EXPORT_DATABASE_STATS Procedure</td>
<td>Retrieves statistics for all objects in the database and stores them in the user statistics table identified by statown.stattab</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>EXPORT_DICTIONARY_SSTATS Procedure</strong></td>
<td>Retrieves statistics for all dictionary schemas (SYS', 'SYSTEM' and RDBMS component schemas) and stores them in the user statistics table identified by stattab</td>
</tr>
<tr>
<td><strong>EXPORT_FIXED_OBJECTS_STATS Procedure</strong></td>
<td>Retrieves statistics for fixed tables and stores them in the user statistics table identified by stattab</td>
</tr>
<tr>
<td><strong>EXPORT_INDEX_STATS Procedure</strong></td>
<td>Retrieves statistics for a particular index and stores them in the user statistics table identified by stattab</td>
</tr>
<tr>
<td><strong>EXPORT_PENDING_STATS Procedure</strong></td>
<td>Exports the statistics gathered and stored as pending</td>
</tr>
<tr>
<td><strong>EXPORT_SCHEMA_PREFS Procedure</strong></td>
<td>Exports the statistics preferences of all the tables owned by the specified owner name</td>
</tr>
<tr>
<td><strong>EXPORT_SCHEMA STAT S Procedure</strong></td>
<td>Retrieves statistics for all objects in the schema identified by ownname and stores them in the user statistics table identified by stattab</td>
</tr>
<tr>
<td><strong>EXPORT_SYSTEM_STATS Procedure</strong></td>
<td>Retrieves system statistics and stores them in the user statistics table</td>
</tr>
<tr>
<td><strong>EXPORT_TABLE_PREFS Procedure</strong></td>
<td>Exports statistics preferences of the specified table in the specified schema into the specified statistics table</td>
</tr>
<tr>
<td><strong>EXPORT_TABLE_STATS Procedure</strong></td>
<td>Retrieves statistics for a particular table and stores them in the user statistics table</td>
</tr>
<tr>
<td><strong>FLUSH_DATABASE_MONITORING_INFO Procedure</strong></td>
<td>Flushes in-memory monitoring information for all the tables to the dictionary</td>
</tr>
<tr>
<td><strong>GATHER_DATABASE_STATS Procedures</strong></td>
<td>Gathers statistics for all objects in the database</td>
</tr>
<tr>
<td><strong>GATHER_DICTIONARY_STATS Procedure</strong></td>
<td>Gathers statistics for dictionary schemas 'SYS', 'SYSTEM' and schemas of RDBMS components</td>
</tr>
<tr>
<td><strong>GATHER_FIXED_OBJECTS_STATS Procedure</strong></td>
<td>Gathers statistics of fixed objects</td>
</tr>
<tr>
<td><strong>GATHER_INDEX_STATS Procedure</strong></td>
<td>Gathers index statistics</td>
</tr>
<tr>
<td><strong>GATHER_PROCESSING_RATE Procedure</strong></td>
<td>Starts the job of gathering the processing rates which end after interval defined in minutes</td>
</tr>
<tr>
<td><strong>GATHER_SCHEMA_STATS Procedures</strong></td>
<td>Gathers statistics for all objects in a schema</td>
</tr>
<tr>
<td><strong>GATHER_SYSTEM_STATS Procedure</strong></td>
<td>Gathers system statistics</td>
</tr>
<tr>
<td><strong>GATHER_TABLE_STATS Procedure</strong></td>
<td>Gathers table and column (and index) statistics</td>
</tr>
<tr>
<td><strong>GENERATE_STATS Procedure</strong></td>
<td>Generates object statistics from previously collected statistics of related objects</td>
</tr>
<tr>
<td><strong>GET_ADVISOR_OPR_FILTER Procedure</strong></td>
<td>Creates an operation filter for an Optimizer Statistics Advisor operation</td>
</tr>
<tr>
<td><strong>GET_ADVISOR_RECS Function</strong></td>
<td>Generates a recommendation report for the specified item</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>GET_COLUMN_STATS</strong> Procedures</td>
<td>Gets all column-related information</td>
</tr>
<tr>
<td><strong>GET_INDEX_STATS</strong> Procedures</td>
<td>Gets all index-related information</td>
</tr>
<tr>
<td><strong>GET_PARAM Function</strong></td>
<td>Gets the default value of parameters of DBMS_STATS procedures (see DBMS_STATS Deprecated Subprograms)</td>
</tr>
<tr>
<td><strong>GET_PREFS Function</strong></td>
<td>Gets the default value of the specified preference</td>
</tr>
<tr>
<td><strong>GET_STATS_HISTORY_AVAILABILITY Function</strong></td>
<td>Gets the oldest timestamp where statistics history is available</td>
</tr>
<tr>
<td><strong>GET_STATS_HISTORY_RETENTION Function</strong></td>
<td>Returns the current statistics history retention value</td>
</tr>
<tr>
<td><strong>GET_SYSTEM_STATS Procedure</strong></td>
<td>Gets system statistics from statab, or from the dictionary if statab is NULL</td>
</tr>
<tr>
<td><strong>GET_TABLE_STATS Procedure</strong></td>
<td>Gets all table-related information</td>
</tr>
<tr>
<td><strong>IMPLEMENT_ADVISOR_TASK Function</strong></td>
<td>Implements the recommendations made by Optimizer Statistics Advisor</td>
</tr>
<tr>
<td><strong>IMPORT_COLUMN_STATS Procedure</strong></td>
<td>Retrieves statistics for a particular column from the user statistics table identified by statab and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>IMPORT_DATABASE_PREFS Procedure</strong></td>
<td>Imports the statistics preferences of all the tables</td>
</tr>
<tr>
<td><strong>IMPORT_DATABASE_STATS Procedure</strong></td>
<td>Retrieves statistics for all objects in the database from the user statistics table and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>IMPORT_DICTIONARY_STATS Procedure</strong></td>
<td>Retrieves statistics for all dictionary schemas (SYS, SYSTEM and RDBMS component schemas) from the user statistics table and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>IMPORT_FIXED_OBJECTS_STATS Procedure</strong></td>
<td>Retrieves statistics for fixed tables from the user statistics table identified by statab and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>IMPORT_INDEX_STATS Procedure</strong></td>
<td>Retrieves statistics for a particular index from the user statistics table identified by statab and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>IMPORT_SCHEMA_PREFS Procedure</strong></td>
<td>Imports the statistics preferences of all the tables owned by the specified owner name</td>
</tr>
<tr>
<td><strong>IMPORT_SCHEMA_STATS Procedure</strong></td>
<td>Retrieves statistics for all objects in the schema identified by ownname from the user statistics table and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>IMPORT_SYSTEM_STATS Procedure</strong></td>
<td>Retrieves system statistics from the user statistics table and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>IMPORT_TABLE_PREFS Procedure</strong></td>
<td>Sets the statistics preferences of the specified table in the specified schema</td>
</tr>
<tr>
<td><strong>IMPORT_TABLE_STATS Procedure</strong></td>
<td>Retrieves statistics for a particular table from the user statistics table identified by statab and stores them in the dictionary</td>
</tr>
<tr>
<td><strong>INTERRUPT_ADVISOR_TASK Procedure</strong></td>
<td>Interrupts a currently executing Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td><strong>LOCK_PARTITION_STATS Procedure</strong></td>
<td>Locks statistics for a partition</td>
</tr>
</tbody>
</table>
### Table 162-3  (Cont.) DBMS_STATS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCK_SCHEMA_STATS Procedure</td>
<td>Locks the statistics of all tables of a schema</td>
</tr>
<tr>
<td>LOCK_TABLE_STATS Procedure</td>
<td>Locks the statistics on the table</td>
</tr>
<tr>
<td>MERGE_COL_USAGE Procedure</td>
<td>Merges column usage information from a source database, by means of a dblink, into the local database</td>
</tr>
<tr>
<td>PREPARE_COLUMN_VALUES Procedures</td>
<td>Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle’s internal representation for future storage using the SEED_COL_USAGE Procedure</td>
</tr>
<tr>
<td>PREPARE_COLUMN_VALUES_NVARCHAR Procedure</td>
<td>Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle’s internal representation for future storage using the SEED_COL_USAGE Procedure</td>
</tr>
<tr>
<td>PREPARE_COLUMN_VALUES_ROWID Procedure</td>
<td>Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle’s internal representation for future storage using the SEED_COL_USAGE Procedure</td>
</tr>
<tr>
<td>PUBLISH_PENDING_STATS Procedure</td>
<td>Publishes the statistics gathered and stored as pending</td>
</tr>
<tr>
<td>PURGE_STATS Procedure</td>
<td>Purges old versions of statistics saved in the dictionary</td>
</tr>
<tr>
<td>REMAP_STAT_TABLE Procedure</td>
<td>Remaps the names of objects in the user statistics table</td>
</tr>
<tr>
<td>REPORT_ADVISOR_TASK Function</td>
<td>Reports the results of an Optimizer Advisor Task</td>
</tr>
<tr>
<td>REPORT_COL_USAGE Function</td>
<td>Reports the recorded column (group) usage information</td>
</tr>
<tr>
<td>REPORT_GATHER_AUTO_STATS Function</td>
<td>Runs the auto statistics gathering job in reporting mode</td>
</tr>
<tr>
<td>REPORT_GATHER_DATABASE_STATS Functions</td>
<td>Runs the GATHER_DATABASE_STATS Procedures in reporting mode</td>
</tr>
<tr>
<td>REPORT_GATHER_DICTIONARY_STATS Functions</td>
<td>Runs the GATHER_DICTIONARY_STATS Procedure in reporting mode</td>
</tr>
<tr>
<td>REPORT_GATHER_FIXED_OBJ_STATS Function</td>
<td>Runs the GATHER_FIXEDOBJECTS_STATS Procedure in reporting mode</td>
</tr>
<tr>
<td>REPORT_GATHER_SCHEMA_STATS Functions</td>
<td>Runs the GATHER_SCHEMA_STATS Procedures in reporting mode</td>
</tr>
<tr>
<td>REPORT_GATHER_TABLE_STATS Function</td>
<td>Runs the GATHER_TABLE_STATS Procedure in reporting mode</td>
</tr>
<tr>
<td>REPORT_STATS_OPERATIONS Function</td>
<td>Generates a report of all statistics operations that take place between two timestamps which may or may not have been provided</td>
</tr>
<tr>
<td>RESET_ADVISOR_TASK Procedure</td>
<td>Resets an Optimizer Statistics Advisor task execution to its initial state. Only reset a task that is not currently executing</td>
</tr>
<tr>
<td>RESET_COL_USAGE Procedure</td>
<td>Resets the recorded column (group) usage information</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>RESET_GLOBAL_PREF_DEFAULTS</code></td>
<td>Procedure resets the default values of all parameters to Oracle recommended values</td>
</tr>
<tr>
<td><code>RESET_PARAM_DEFAULTS</code></td>
<td>Procedure resets global preferences to default values (see DBMS_STATS Deprecated Subprograms)</td>
</tr>
<tr>
<td><code>RESTORE_DICITIONARY_STATS</code></td>
<td>Procedure restores statistics of all dictionary tables (tables of 'SYS', 'SYSTEM' and RDBMS component schemas) as of a specified timestamp</td>
</tr>
<tr>
<td><code>RESTORE_FIXED_OBJECTS_STATS</code></td>
<td>Procedure restores statistics of all fixed tables as of a specified timestamp</td>
</tr>
<tr>
<td><code>RESTORE_SCHEMA_SCHEMA</code></td>
<td>Procedure restores statistics of all tables of a schema as of a specified timestamp</td>
</tr>
<tr>
<td><code>RESTORE_SYSTEM_SCHEMA</code></td>
<td>Procedure restores statistics of all tables of a schema as of a specified timestamp</td>
</tr>
<tr>
<td><code>RESTORE_TABLE_STATS</code></td>
<td>Procedure restores statistics of a table as of a specified timestamp (as_of_timestamp), as well as statistics of associated indexes and columns</td>
</tr>
<tr>
<td><code>RESUME_ADVISOR_TASK</code></td>
<td>Procedure resumes an interrupted task. It only resumes the execution that was most recently interrupted</td>
</tr>
<tr>
<td><code>SCRIPT_ADVISOR_TASK</code></td>
<td>Function retrieves the script that implements the recommended actions for the problems found by Optimizer Statistics Advisor</td>
</tr>
<tr>
<td><code>SEED_COL_USAGE</code></td>
<td>Procedure iterates over the SQL statements in the specified SQL tuning set, compiles them and seeds column usage information for the columns that appear in these statements</td>
</tr>
<tr>
<td><code>SET_ADVISOR_TASK_PARAMETER</code></td>
<td>Procedure updates the value of an Optimizer Statistics Advisor task parameter</td>
</tr>
<tr>
<td><code>SET_COLUMNS_STATS</code></td>
<td>Procedures sets column-related information</td>
</tr>
<tr>
<td><code>SET_DATABASE_PREFS</code></td>
<td>Procedure sets the statistics preferences of all the tables</td>
</tr>
<tr>
<td><code>SET_GLOBAL_PREFS</code></td>
<td>Procedure sets the global statistics preferences</td>
</tr>
<tr>
<td><code>SET_INDEX_STATS</code></td>
<td>Procedures sets index-related information</td>
</tr>
<tr>
<td><code>SET_PARAM</code></td>
<td>Procedure sets default values for parameters of DBMS_STATS procedures (see DBMS_STATS Deprecated Subprograms)</td>
</tr>
<tr>
<td><code>SET_PROCESSING_RATE</code></td>
<td>Procedure sets the value of rate of processing for a given operation</td>
</tr>
<tr>
<td><code>SET_SCHEMA_PREFS</code></td>
<td>Procedure sets the statistics preferences of all the tables owned by the specified owner name</td>
</tr>
<tr>
<td><code>SET_SYSTEM_STATS</code></td>
<td>Procedure sets system statistics</td>
</tr>
<tr>
<td><code>SET_TABLE_PREFS</code></td>
<td>Procedure sets the statistics preferences of the specified table in the specified schema</td>
</tr>
<tr>
<td><code>SET_TABLESTATS</code></td>
<td>Procedure sets table-related information</td>
</tr>
<tr>
<td><code>SHOW_EXTENDED_STAT</code></td>
<td>Function returns the name of the virtual column that is created for the user-specified extension</td>
</tr>
</tbody>
</table>
Table 162-3  (Cont.) *DBMS_STATS Package Subprograms*

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSFER_STATS Procedure</td>
<td>Transfers statistics for specified table(s) from a remote database specified by dblink to the local database</td>
</tr>
<tr>
<td>UNLOCK_PARTITION_STATS Procedure</td>
<td>Unlocks the statistics for a partition</td>
</tr>
<tr>
<td>UNLOCK_SCHEMA_STATS Procedure</td>
<td>Unlocks the statistics on all the tables in schema</td>
</tr>
<tr>
<td>UNLOCK_TABLE_STATS Procedure</td>
<td>Unlocks the statistics on the table</td>
</tr>
<tr>
<td>UPGRADE_STAT_TABLE Procedure</td>
<td>Upgrades user statistics on an older table</td>
</tr>
</tbody>
</table>

162.7.1 ALTER_STATS_HISTORY_RETENTION Procedure

This procedure changes the statistics history retention value.

Statistics history retention is used by both the automatic purge and PURGE_STATS Procedure.

**Syntax**

```sql
DBMS_STATS.ALTER_STATS_HISTORY_RETENTION (retention IN NUMBER);
```

**Parameters**

Table 162-4  ALTER_STATS_HISTORY_RETENTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| retention | The retention time in days. The statistics history will be retained for at least these many number of days. The valid range is [1,365000]. Also you can use the following values for special purposes:  
  - -1: Statistics history is never purged by automatic purge  
  - 0: Old statistics are never saved. The automatic purge will delete all statistics history  
  - NULL: Change statistics history retention to default value |

**Usage Notes**

To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

**Exceptions**

ORA-20000: Insufficient privileges
162.7.2 CANCEL_ADVISOR_TASK Procedure

This procedure cancels an Optimizer Statistics Advisor execution. The advisor removes all intermediate results of the current execution from the task.

Syntax

```
DBMS_STATS.CANCEL_ADVISOR_TASK (
    task_name IN VARCHAR2);
```

Parameters

Table 162-5  CANCEL_ADVISOR_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.

Consider a case in which a task is executed by one user, interrupted, and then resumed by a different user. In this case, Optimizer Statistics Advisor bases its checks of the resumed execution on the privilege of the user who resumed the task.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
- ORA-20012: Optimizer Statistics Advisor errors

Usage Notes

To be canceled or interrupted, the specified task must be currently executing.

Example 162-1  Canceling an Optimizer Statistics Advisor

In this example, you start a SQL*Plus session, and then create and execute an advisor task named `my_task`:

```
DECLARE
    v_tname   VARCHAR2(128) := 'my_task';
    v_ename   VARCHAR2(128) := NULL;
BEGIN
    -- create a task
    v_tname := DBMS_STATS.CREATE_ADVISOR_TASK(v_tname);
    -- execute the task
```
v_ename := DBMS_STATS.EXECUTE_ADVISOR_TASK(v_tname);
END;
/

In a separate terminal, you start a second SQL*Plus session, and then execute the following program:

EXEC DBMS_STATS.CANCEL_ADVISOR_TASK('my_task');

The first session returns an ORA-13632 to indicate the cancelation of the task:
ORA-13632: The user cancelled the current operation.

162.7.3 CONFIGURE_ADVISOR_FILTER Function

This function configures the filter list for an Optimizer Statistics Advisor task. Filters are useful for excluding irrelevant findings from a report.

Syntax

DBMS_STATS.CONFIGURE_ADVISOR_FILTER (  
  task_name IN VARCHAR2,  
  stats_adv_opr_type IN VARCHAR2,  
  configuration_type IN VARCHAR2,  
  filter IN StatsAdvFilterTab := NULL)  
RETURN CLOB;

Parameters

Table 162-6  CONFIGURE_ADVISOR_FILTER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td>stats_adv_opr_type</td>
<td>The type of operation to configure. Possible values are EXECUTE, REPORT, SCRIPT, and IMPLEMENT. The function permits you to specify a combination of operation types by using the plus (+) operator, for example, EXECUTE +REPORT. If this parameter is null, then the filter applies to all types of advisor operations.</td>
</tr>
<tr>
<td>configuration_type</td>
<td>The type of configuration. Possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td>• SET: Sets the specified filter list values. The submitted filter overrides existing filter values.</td>
</tr>
<tr>
<td></td>
<td>• CLEAR: Clears the existing values for the specified filter.</td>
</tr>
<tr>
<td></td>
<td>• SHOW: Shows the current values of the specified filter.</td>
</tr>
<tr>
<td>filter</td>
<td>The list of filter items for the script.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

• To execute this subprogram, you must have the ADVISOR privilege.
• You must be the owner of the task.
• This subprogram executes using invoker’s rights.

Return Values
This function returns a CLOB that contains the configuration of the provided filter in XML format.

Exceptions
• ORA-20000: Insufficient privileges
• ORA-20001: Invalid input values
• ORA-20012: Optimizer Statistics Advisor errors

Usage Notes
To provide fine-grained control and a unified interface across all procedures, DBMS_STATS provides the StatsAdvFilter type. You can use this data type to instantiate and construct a table of filters. You can then pass a parameter of type StatsAdvFilter to CONFIGURE_ADVISOR_FILTER along with a Boolean variable that specifies either of the following:

• Inclusion list
  Only include these objects in the check.
• Exclusion list
  Do not include these objects in the check.

You can also pass in a parameter specifying whether to replace the existing list. This list only filters object-level and operation-level items. The advisor always checks system-level rules.

You can create the following types of filters:

• Rule filter
  This filter takes a rule name as input. Obtain rule names from the \$STATS_ADVISOR_RULES view.
• Operation filter
  This filter is an exact match filter that takes in the name of the operation and an XML string representation of all the parameter values in the call. To obtain the XML, see the notes section of the DBA_OPTSTAT_OPERATIONS view. To obtain the filter for an operation, use DBMS_STATS.GET_ADVISOR_OPR_FILTER.
• Object filter
  This filter accept an owner name and an object name. Wildcards (%) are supported in the owner name and object name. When an object name is null or %, it means a filter for all the objects in the specified schema. If the owner name is also null or %, it means a default filter for all objects in the system.

If none of the filters is specified, then the function recognizes the filter as setting the global default value of filtering (include or exclude). During the check, if no filter has been specified for a rule, operation, or object, then the function uses the default value to determine whether to include or exclude it.
Example 162-2   Enabling and Disabling Rules

You may want to turn off checks for all rules except for a specific rule. In this example, you want to check whether SQL plan directives have been disabled.

DECLARE
  v_task_name VARCHAR2(128) := 'my_task';
  v_ret VARCHAR2(128);
  filter DBMS_STATS.StatsAdvFilter := null;
  filterTab DBMS_STATS.StatsAdvFilterTab := null;
  v_counter NUMBER := 0;
  v_filterReport CLOB;
BEGIN
  -- Create the advisor task
  v_ret := DBMS_STATS.CREATE_ADVISOR_TASK(v_task_name);

  -- Initialize the filter table
  filterTab := DBMS_STATS.StatsAdvFilterTab();

  -- First filter: set filters to be FALSE by default
  filter.include := FALSE;

  -- Add this filter to the filter table
  v_counter := v_counter + 1;
  filterTab.extend;
  filterTab(v_counter) := filter;

  -- Second filter: turn on filter for one rule
  filter.include := TRUE;
  filter.rulename := 'TurnOnSQLPlanDirective';

  -- Add the SQL plan directive filter to the filter table
  v_counter := v_counter + 1;
  filterTab.extend;
  filterTab(v_counter) := filter;

  v_filterReport := DBMS_STATS.CONFIGURE_ADVISOR_FILTER(
    task_name          => v_task_name,
    stats_adv_opr_type => NULL,
    configuration_type => 'SET',
    filter             => filterTab);

  -- Drop the task
  DBMS_STATS.DROP_ADVISOR_TASK(v_task_name);
END;

Example 162-3   Configuring an Operations Filter

In this example, your shop uses customized scripts to gather statistics for a table. If you do not want to see a specific statistics operation in the report, then you can specify an operations filter.

DECLARE
  v_task_name VARCHAR2(128) := 'my_task';
  v_ret VARCHAR2(128);
  filter DBMS_STATS.StatsAdvFilter := null;
BEGIN
  -- Create the advisor task
  v_ret := DBMS_STATS.CREATE_ADVISOR_TASK(v_task_name);

  -- Initialize filter table
  filterTab := DBMS_STATS.StatsAdvFilterTab();

  -- Initialize operations filter
  oprTab := DBMS_STATS.StatsAdvOprTab();

  SELECT ID
  BULK COLLECT INTO opr_tab
  FROM WRI$OPTSTAT_OPR
  WHERE OPERATION = 'set_table_stats'
  AND TARGET = 'HR.EMPLOYEES';

  -- Populate the operations table
  FOR i IN 1..opr_tab.count LOOP
    -- Use the procedure GET_ADVISOR_OPR_FILTER to construct
    -- an operation filter
    DBMS_STATS.GET_ADVISOR_OPR_FILTER(opr_tab(i), opr);
    v_oprCnt := v_oprCnt + 1;
    oprTab.extend;
    oprTab(v_oprCnt) := opr;
  END LOOP;

  filter.include := FALSE;
  filter.oprlist := oprTab;

  -- Add to filter table
  filterTab.extend;
  filterTab(1) := filter;

  v_filterReport := DBMS_STATS.CONFIGURE_ADVISOR_FILTER(
    task_name => v_task_name,
    stats_adv_opr_type => NULL,
    configuration_type => 'SET',
    filter => filterTab);

  -- Drop the task
  DBMS_STATS.DROP_ADVISOR_TASK(v_task_name);
END;
Example 162-4  Reporting on a Specific Schema

In this example, you want to generate a report only for the sh schema. Also, you want to skip the sh.products table. You create an object filter as follows:

```sql
DECLARE
    v_task_name     VARCHAR2(128) := 'my_task';
    v_ret           VARCHAR2(128);
    filter          DBMS_STATS.StatsAdvFilter := null;
    filterTab       DBMS_STATS.StatsAdvFilterTab := null;
    v_filterReport  CLOB;
    v_counter       NUMBER := 0;
    obj             DBMS_STATS.ObjectElem;
    objTab          DBMS_STATS.ObjectTab;
    v_objCnt        NUMBER := 0;
BEGIN
    -- Create the advisor task
    v_ret := DBMS_STATS.CREATE_ADVISOR_TASK(v_task_name);

    -- Initialize filter table
    filterTab := DBMS_STATS.StatsAdvFilterTab();

    -- Set object filter to be off by default
    filter.include := FALSE;
    objTab := DBMS_STATS.ObjectTab();
    obj.ownname := NULL;
    obj.objname := NULL;

    -- Add to the object table
    v_objCnt := v_objCnt + 1;
    objTab.extend;
    objTab(v_objCnt) := obj;

    filter.objlist := objTab;

    -- Add the object filter to the filter table
    v_counter := v_counter + 1;
    filterTab.extend;
    filterTab(v_counter) := filter;

    -- In filter 1, turn on the check only for schema SH
    filter.include := TRUE;
    objTab := DBMS_STATS.ObjectTab();
    v_objCnt := 0;
    obj.ownname := 'SH';
    obj.objname := NULL;

    -- add to the object table
    v_objCnt := v_objCnt + 1;
```
objTab.extend;
objTab(v_objCnt) := obj;

filter.objlist := objTab;

-- Add the object filter to the filter table
v_counter := v_counter + 1;
filterTab.extend;
filterTab(v_counter) := filter;

-- In filter 2, exclude the check for object sh.products
filter.include := FALSE;

objTab := dbms_stats.ObjectTab();
v_objCnt := 0;

-- Specify another object filter for sh.products
obj.ownname := 'SH';
obj.objname := 'PRODUCTS';

-- Add to the object table
v_objCnt := v_objCnt + 1;
objTab.extend;
objTab(v_objCnt) := obj;

filter.objlist := objTab;

-- Add the object filter to the filter table
v_counter := v_counter + 1;
filterTab.extend;
filterTab(v_counter) := filter;

v_filterReport :=
DBMS_STATS.CONFIGURE_ADVISOR_FILTER(
    task_name => v_task_name,
    stats_adv_opr_type => NULL,
    configuration_type => 'SET',
    filter => filterTab);

-- Drop the task
DBMS_STATS.DROP_ADVISOR_TASK(v_task_name);
END;
162.7.4 CONFIGURE_ADVISOR_OBJ_FILTER Function

This function configures an object filter for an Optimizer Statistics Advisor task.

Syntax

```sql
DBMS_STATS.CONFIGURE_ADVISOR_OBJ_FILTER (  
    task_name             IN VARCHAR2,  
    stats_adv_opr_type    IN VARCHAR2,  
    rule_name             IN VARCHAR2,  
    ownname               IN VARCHAR2,  
    tabname               IN VARCHAR2,  
    action                IN VARCHAR2)  
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td>stats_adv_opr_type</td>
<td>The type of operation to configure. Possible values are EXECUTE, REPORT, SCRIPT, and IMPLEMENT. See &quot;CONFIGURE_ADVISOR_RULE_FILTER Function&quot;.</td>
</tr>
<tr>
<td>rule_name</td>
<td>The name of the rule to configure. If null, the function applies the filter to all operation-level rules.</td>
</tr>
<tr>
<td>ownname</td>
<td>The owner name of the operation target. If null, the function applies the filter to all owner names.</td>
</tr>
<tr>
<td>tabname</td>
<td>The table name of the operation target.</td>
</tr>
<tr>
<td>action</td>
<td>The configuration action to take for the specified rule. See &quot;CONFIGURE_ADVISOR_RULE_FILTER Function&quot;.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.

Return Values

This function returns an XML CLOB that contains the updated values of the filter.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
- ORA-20012: Optimizer Statistics Advisor errors
162.7.5 CONFIGURE_ADVISOR_OPR_FILTER Functions

This overloaded function configures an operation filter for an Optimizer Statistics Advisor task.

Syntax

```sql
DBMS_STATS.CONFIGURE_ADVISOR_OPR_FILTER (    task_name             IN VARCHAR2,
    stats_adv_opr_type    IN VARCHAR2,
    rule_name             IN VARCHAR2,
    operation_name        IN VARCHAR2,
    action                IN VARCHAR2)    RETURN CLOB;
```

```sql
DBMS_STATS.CONFIGURE_ADVISOR_OPR_FILTER (    task_name             IN VARCHAR2,
    stats_adv_opr_type    IN VARCHAR2,
    rule_name             IN VARCHAR2,
    operation_name        IN VARCHAR2,
    ownname               IN VARCHAR2,
    tabname               IN VARCHAR2,
    action                IN VARCHAR2)    RETURN CLOB;
```

```sql
DBMS_STATS.CONFIGURE_ADVISOR_OPR_FILTER (    task_name             IN VARCHAR2,
    stats_adv_opr_type    IN VARCHAR2,
    rule_name             IN VARCHAR2,
    operation_id          IN NUMBER,
    action                IN VARCHAR2)    RETURN CLOB;
```

Parameters

Table 162-8  CONFIGURE_ADVISOR_OPR_FILTER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td>stats_adv_opr_type</td>
<td>The type of operation to configure. Possible values are EXECUTE, REPORT, SCRIPT, and IMPLEMENT. See &quot;CONFIGURE_ADVISOR_RULE_FILTER Function&quot;.</td>
</tr>
<tr>
<td>rule_name</td>
<td>The name of the rule to configure. If null, the function applies the filter to all operation-level rules.</td>
</tr>
<tr>
<td>operation_name</td>
<td>The name of the operation. For example, an operation name could be gather_table_stats. This value cannot be null.</td>
</tr>
</tbody>
</table>
Table 162-8  (Cont.) CONFIGURE_ADVISOR_OPR_FILTER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation_id</td>
<td>The ID of the operation to configure. The filter applies to any operation with the same signature as the specified operation ID. If two operations have the same signature, then they have the same value for every parameter. View the operation ID in DBA_OPSTAT_OPERATIONS.ID. This value cannot be null.</td>
</tr>
<tr>
<td>ownname</td>
<td>The owner name of the operation target. This value cannot be null.</td>
</tr>
<tr>
<td>tabname</td>
<td>The table name of the operation target.</td>
</tr>
<tr>
<td>action</td>
<td>The configuration action to take for the specified rule. See &quot;CONFIGURE_ADVISOR_RULE_FILTER Function&quot;.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.

Return Values

This function returns an XML CLOB that contains the updated values of the filter.

Exceptions

- ORA-20000: Insufficient privileges  
- ORA-20001: Invalid input values  
- ORA-20012: Optimizer Statistics Advisor errors

Example 162-5  Excluding Operations for Gathering Table Statistics

In this example, your goal is to exclude operations that gather table statistics in the hr schema. User account stats has been granted the DBA role, ADVISOR privilege, and SELECT ON DBA_OPTSTAT_OPERATIONS privilege. You perform the following steps:

1. Log in to the database as stats.

2. Drop any existing task named opt_adv_task1.

   DECLARE
   v_tname VARCHAR2(32767);
   BEGIN
   v_tname := 'opt_adv_task1';
   DBMS_STATS DROP_ADVISOR_TASK(v_tname);
   END;
   /
3. Create a procedure named `opr_filter` that configures a task to advise on all operations except those that gather statistics for tables in the `hr` schema.

```sql
CREATE OR REPLACE PROCEDURE opr_filter(p_tname IN VARCHAR2) IS
  v_retc CLOB;
BEGIN
  -- For all rules, prevent the advisor from operating
  -- on the operations selected in the following query
  FOR rec IN
    (SELECT ID FROM DBA_OPTSTAT_OPERATIONS WHERE OPERATION = 'gather_table_stats' AND TARGET LIKE 'HR.%')
  LOOP
    v_retc := DBMS_STATS.CONFIGURE_ADVISOR_OPR_FILTER(
      task_name          => p_tname,
      stats_adv_opr_type => NULL,
      rule_name          => NULL,
      operation_id       => rec.id,
      action             => 'DISABLE');
  END LOOP;
END;
/
SHOW ERRORS
```

4. Create a task named `opt_adv_task1`, and then execute the `opr_filter` procedure for this task.

```sql
DECLARE
  v_tname VARCHAR2(32767);
  v_ret VARCHAR2(32767);
BEGIN
  v_tname := 'opt_adv_task1';
  v_ret   := DBMS_STATS.CREATE_ADVISOR_TASK(v_tname);
  opr_filter(v_tname);
END;
/
```

5. Execute the task `opt_adv_task1`.

```sql
DECLARE
  v_tname VARCHAR2(32767);
  v_ret VARCHAR2(32767);
BEGIN
  v_tname := 'opt_adv_task1';
  v_ret   := DBMS_STATS.EXECUTE_ADVISOR_TASK(v_tname);
END;
/
```

6. Print the report.

```sql
SPOOL /tmp/rep.txt
SET LONG 1000000
COLUMN report FORMAT A200
SET LINESIZE 250
SET PAGESIZE 1000
```
162.7.6 CONFIGURE_ADVISOR_RULE_FILTER Function

This function configures a rule filter for an Optimizer Statistics Advisor task.

Syntax

```sql
DBMS_STATS.CONFIGURE_ADVISOR_RULE_FILTER (  
    task_name            IN   VARCHAR2,  
    stats_adv_opr_Type   IN   VARCHAR2,  
    rule_name            IN   VARCHAR2,  
    action               IN   VARCHAR2)
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td>stats_adv_opr_Type</td>
<td>The type of operation to configure. Possible values are EXECUTE, REPORT, SCRIPT, and IMPLEMENT. You can specify a combination of operation types, for example, EXECUTE +REPORT. If this parameter is null, then the filter applies to all types of Optimizer Statistics Advisor operations.</td>
</tr>
<tr>
<td>rule_name</td>
<td>The name of the rule to configure. If null, the function applies the filter to all rules.</td>
</tr>
<tr>
<td>action</td>
<td>The configuration action to take for the specified rule. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• ENABLE: Enables the filter</td>
</tr>
<tr>
<td></td>
<td>• DISABLE: Disables the filter</td>
</tr>
<tr>
<td></td>
<td>• DELETE: Deletes the filter</td>
</tr>
<tr>
<td></td>
<td>• SHOW: Shows the current filter value</td>
</tr>
</tbody>
</table>
Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.

Return Values

This function returns an XML CLOB that contains the updated values of the filter.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
- ORA-20012: Optimizer Statistics Advisor errors

162.7.7 CONVERT_RAW_VALUE Procedures

This procedure converts the internal representation of a minimum value, maximum value, or histogram endpoint actual value into a datatype-specific value.

The minval, maxval, and eavals fields of the StatRec structure as filled in by GET_COLUMN_STATS or PREPARE_COLUMN_VALUES are appropriate values for input.

Syntax

```sql
DBMS_STATS.CONVERT_RAW_VALUE (    rawval RAW,    resval OUT BINARY_FLOAT);
```

```sql
DBMS_STATS.CONVERT_RAW_VALUE (    rawval RAW,    resval OUT BINARY_DOUBLE);
```

```sql
DBMS_STATS.CONVERT_RAW_VALUE (    rawval RAW,    resval OUT DATE);
```

```sql
DBMS_STATS.CONVERT_RAW_VALUE (    rawval RAW,    resval OUT NUMBER);
```

```sql
DBMS_STATS.CONVERT_RAW_VALUE (    rawval RAW,    resval OUT VARCHAR2);
```
Parameters

Table 162-10  CONVERT_RAW_VALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawval</td>
<td>Raw representation of a column minimum, maximum, histogram end point actual value</td>
</tr>
<tr>
<td>resval</td>
<td>Converted, type-specific value</td>
</tr>
</tbody>
</table>

Usage Notes

No special privilege or role is needed to invoke this procedure.

162.7.8 CONVERT_RAW_VALUE_NVARCHAR Procedure

This procedure converts the internal representation of a minimum value, maximum value, or histogram end point actual value.

The minval, maxval and eavals fields of the StatRec structure as filled in by GET_COLUMN_STATS or PREPARE_COLUMN_VALUES are appropriate values for input.

Syntax

```sql
DBMS_STATS.CONVERT_RAW_VALUE_NVARCHAR (  
  rawval RAW,  
  resval OUT NVARCHAR2);  
```

Parameters

Table 162-11  CONVERT_RAW_VALUE_NVARCHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawval</td>
<td>The raw representation of a column minimum or maximum datatype-specific output parameters</td>
</tr>
<tr>
<td>resval</td>
<td>The converted, type-specific value</td>
</tr>
</tbody>
</table>

Usage Notes

No special privilege or role is needed to invoke this procedure.

162.7.9 CONVERT_RAW_VALUE_ROWID Procedure

This procedure converts the internal representation of a minimum value, maximum value, or histogram end point actual value.

The minval, maxval and eavals fields of the StatRec structure as filled in by GET_COLUMN_STATS or PREPARE_COLUMN_VALUES are appropriate values for input.
Syntax

DBMS_STATS.CONVERT_RAW_VALUE_ROWID (  
  rawval    RAW,  
  resval OUT ROWID);

Pragmas

pragma restrict_references(convert_raw_value_rowid, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 162-12 CONVERT_RAW_VALUE_ROWID Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawval</td>
<td>The raw representation of a column minimum or maximum datatype-specific output parameters</td>
</tr>
<tr>
<td>resval</td>
<td>The converted, type-specific value</td>
</tr>
</tbody>
</table>

Usage Notes

No special privilege or role is needed to invoke this procedure.

162.7.10 COPY_TABLE_STATS Procedure

This procedure copies statistics of all dependent object such as columns and local indexes. If the statistics for source are not available then nothing is copied. It can optionally scale the statistics (such as the number of blks, or number of rows) based on the given scale_factor.

Syntax

DBMS_STATS.COPY_TABLE_STATS (  
  ownname      VARCHAR2,  
  tabname      VARCHAR2,  
  srcpartname  VARCHAR2,  
  dstpartname  VARCHAR2,  
  scale_factor VARCHAR2 DEFAULT 1,  
  flags        NUMBER DEFAULT NULL,  
  force        BOOLEAN DEFAULT FALSE);

Parameters

Table 162-13 COPY_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of the table of source and destination [sub] partitions</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name of source and destination [sub] partitions</td>
</tr>
<tr>
<td>srcpartname</td>
<td>Source [sub] partition</td>
</tr>
<tr>
<td>dstpartname</td>
<td>Destination [sub] partition</td>
</tr>
<tr>
<td>scale_factor</td>
<td>Scale factor to scale nblks, nrows etc. in dstpartname</td>
</tr>
</tbody>
</table>
Table 162-13  (Cont.) COPY_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>For internal Oracle use (should be left as NULL)</td>
</tr>
<tr>
<td>force</td>
<td>When value of this argument is TRUE copy statistics even if the destination</td>
</tr>
<tr>
<td></td>
<td>[sub]partition is locked</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Exceptions

ORA-20000: Invalid [sub]partition name

ORA-20001: Bad input value

Usage Notes

This procedure updates the minimum and maximum values of destination partition for the first partitioning column as follows:

- If the partitioning type is HASH, then the minimum and maximum values of the destination partition are same as that of the source partition.

- If the partitioning type is LIST, then the behavior depends on the setting of the destination partition:
  - If the destination partition is a NOT DEFAULT partition, then the following statements are true:
    * The minimum value of the destination partition is set to the minimum value of the value list that describes the destination partition.
    * The maximum value of the destination partition is set to the maximum value of the value list that describes the destination partition.
  - Alternatively, if the destination partition is a DEFAULT partition, then the following statements are true:
    * The minimum value of the destination partition is set to the minimum value of the source partition.
    * The maximum value of the destination partition is set to the maximum value of the source partition.

- If the partitioning type is RANGE, then the following statements are true:
  - The minimum value of the destination partition is set to the high bound of previous partition unless the destination partition is the first partition. For the first partition, the minimum value is set to the high bound of the destination partition.
  - The maximum value of the destination partition is set to the high bound of the destination partition unless the high bound of the destination partition is...
MAXVALUE, in which case the maximum value of the destination partition is set to the high bound of the previous partition.

- If the source partition column's minimum value is equal to its maximum value, and if both are equal to the source partition's lower bound, and if it has a single distinct value, then the destination partition column's minimum and maximum values are both set to the destination partition's lower bound. This is done for all partitioning columns.

If the above condition does not apply, second and subsequent partitioning columns are updated as follows. The destination partition column's maximum value is set to the greater of the destination partition upper bound and the source partition column's maximum value, with one exception. If the destination partition is \( D \) and its preceding partition is \( D-1 \) and the key column to be adjusted is \( C_n \), the maximum value for \( C_n \) is set to the upper bound of \( D \) (ignoring the maximum value of the source partition column) provided that the upper bounds of the previous key column \( C_{n-1} \) are the same in partitions \( D \) and \( D-1 \).

- If the minimum and maximum values are different for a column after modifications, and if the number of distinct values is less than 1, then the number of distinct values is updated as 2.
- If the source or destination is a partition of a composite partitioned table, then this procedure does not copy statistics of the underlying subpartitions.

### 162.7.11 CREATE_ADVISOR_TASK Function

This function creates an advisor task for the Optimizer Statistics Advisor.

**Syntax**

```sql
DBMS_STATS.CREATE_ADVISOR_TASK (  
    task_name IN VARCHAR2 := NULL)  
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>task_name</code></td>
<td>Name of the task. If the task name is already specified, then the function uses the specified task name. Otherwise, the function generates a new task name automatically.</td>
</tr>
</tbody>
</table>

**Security Model**

Note the following:

- To execute this subprogram, you must have the `ADVISOR` privilege.
- This subprogram executes using invoker's rights.

**Return Values**

This function returns the unique name of the Optimizer Statistics Advisor task.
Exceptions

ORA-20000: Insufficient privileges / creating extension is not supported
ORA-20001: Error when processing extension
ORA-20012: Optimizer Statistics Advisor errors

Example 162-6  Creating and Executing a Task

This example creates an Optimizer Statistics Advisor task named `my_task`, and then executes it.

```sql
DECLARE
  v_tname   VARCHAR2(128) := 'my_task';
BEGIN
  -- create a task
  v_tname := DBMS_STATS.CREATE_ADVISOR_TASK(v_tname);
  -- execute the task
  v_tname := DBMS_STATS.EXECUTE_ADVISOR_TASK(v_tname);
END;
```

Note:

*Oracle Database SQL Tuning Guide* to learn how manage Optimizer Statistics Advisor

162.7.12 CREATE_EXTENDED_STATS Function

This function creates a column statistics entry in the system for a user-specified column group or an expression in a table.

The database gathers statistics for this extension when a user-initiated or automatic statistics gathering job gathers statistics for the table. Statistics for such an extension are called **extended statistics**. This function returns the name of this newly created entry for the extension.

This second form of this function creates statistics extensions based on the column group usage recorded by the `SEED_COL_USAGE` Procedure. This function returns a report of extensions created.

**Syntax**

```sql
DBMS_STATS.CREATE_EXTENDED_STATS (  
    ownname    VARCHAR2,  
    tabname    VARCHAR2,  
    extension  VARCHAR2)  
RETURN VARCHAR2;

DBMS_STATS.CREATE_EXTENDED_STATS (  
    ownname    VARCHAR2,  
    tabname    VARCHAR2)  
RETURN CLOB;
```
Parameters

Table 162-15  CREATE_EXTENDED_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name of a table</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
<tr>
<td>extension</td>
<td>Can be either a column group or an expression. Suppose the specified table</td>
</tr>
<tr>
<td></td>
<td>has columns c1, c2. An example column group is &quot;(c1, c2)&quot;. An example</td>
</tr>
<tr>
<td></td>
<td>expression is &quot;(c1 + c2)&quot;.</td>
</tr>
</tbody>
</table>

Return Values

This function returns the name of this newly created entry for the extension.

Exceptions

ORA-20000: Insufficient privileges / creating extension is not supported
ORA-20001: Error when processing extension
ORA-20007: Extension already exists
ORA-20008: Reached the upper limit on number of extensions

Usage Notes

To invoke this procedure you must be owner of the table, or have the ANALYZE ANY privilege. For objects owned by SYS, you must be either the owner of the table, or have either the ANALYZE ANY DICTIONARY or SYSDBA privilege.

The extension has the following restrictions:

• The extension cannot contain a virtual column.
• Extensions cannot be created on tables owned by SYS.
• Extensions cannot be created on cluster tables, index organized tables, temporary tables, or external tables.
• The total number of extensions in a table cannot be greater than a maximum of (20, 10% of number of non-virtual columns in the table).
• The number of columns in a column group must be in the range [2, 32].
• A column cannot appear more than once in a column group.
• The extension can contain an expression only if a corresponding virtual column has been created.
• An expression must contain at least one column.
• An expression cannot contain a subquery.
• The COMPATIBLE parameter must be 11.0.0.0.0 or greater.
162.7.13 CREATE_STAT_TABLE Procedure

This procedure creates a table with name `stattab` in `ownname`'s schema which is capable of holding statistics. The columns and types that compose this table are not relevant as it should be accessed solely through the procedures in this package.

Syntax

```
DBMS_STATS.CREATE_STAT_TABLE (        
  ownname             VARCHAR2,        
  stattab             VARCHAR2,        
  tblspace            VARCHAR2 DEFAULT NULL,        
  global_temporary    BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ownname</code></td>
<td>Name of the schema</td>
</tr>
<tr>
<td><code>stattab</code></td>
<td>Name of the table to create. This value should be passed as the <code>stattab</code> parameter to other procedures when the user does not want to modify the dictionary statistics directly.</td>
</tr>
<tr>
<td><code>tblspace</code></td>
<td>Tablespace in which to create the statistics tables. If none is specified, then they are created in the user's default tablespace.</td>
</tr>
<tr>
<td><code>global_temporary</code></td>
<td>Whether or not the table should be created as a global temporary table</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you need whichever privileges are required for creating a table in the specified schema.

Exceptions

- ORA-20000: Table already exists or insufficient privileges
- ORA-20001: Tablespace does not exist

162.7.14 DELETE_COLUMN_STATS Procedure

This procedure deletes column-related statistics.

Syntax

```
DBMS_STATS.DELETE_COLUMN_STATS (        
  ownname        VARCHAR2,        
  tabname        VARCHAR2,        
  colname        VARCHAR2,        
  partname       VARCHAR2 DEFAULT NULL,        
  stattab        VARCHAR2 DEFAULT NULL,        
  statid         VARCHAR2 DEFAULT NULL,        
```

Chapter 162
Summary of DBMS_STATS Subprograms

162-38
cascade_parts BOOLEAN DEFAULT TRUE,
statown VARCHAR2 DEFAULT NULL,
no_invalidate BOOLEAN DEFAULT to_no_invalidate_type {
    get_param('NO_INVALIDATE')},
force BOOLEAN DEFAULT FALSE,
col_stat_type VARCHAR2 DEFAULT 'ALL');

Parameters

Table 162-17  DELETE_COLUMN_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column or extension</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition for which to delete the statistics. If the</td>
</tr>
<tr>
<td></td>
<td>table is partitioned and if partname is NULL, then global column statistics</td>
</tr>
<tr>
<td></td>
<td>are deleted.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to delete the</td>
</tr>
<tr>
<td></td>
<td>statistics. If stattab is NULL, then the statistics are deleted directly</td>
</tr>
<tr>
<td></td>
<td>from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td></td>
<td>(Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>cascade_parts</td>
<td>If the table is partitioned and if partname is NULL, then setting this to</td>
</tr>
<tr>
<td></td>
<td>true causes the deletion of statistics for this column for all underlying</td>
</tr>
<tr>
<td></td>
<td>partitions as well.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure</td>
</tr>
<tr>
<td></td>
<td>invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>When value of this argument is TRUE, deletes column statistics even if locked</td>
</tr>
<tr>
<td>col_stat_type</td>
<td>Type of column statistics to be deleted. This argument takes the following</td>
</tr>
<tr>
<td></td>
<td>values:</td>
</tr>
<tr>
<td></td>
<td>- HISTOGRAM - delete column histogram only</td>
</tr>
<tr>
<td></td>
<td>- ALL - delete base column statistics and histogram</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20005: Object statistics are locked

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.
162.7.15 DELETE_DATABASE_PREFS Procedure

This procedure deletes the statistics preferences set for all non-system tables. You can include system tables by passing TRUE for the add_sys parameter.

Syntax

```sql
DBMS_STATS.DELETE_DATABASE_PREFS (
    pname IN VARCHAR2,
    add_sys IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 162-18  DELETE_DATABASE_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>Preference name. The existing value for following preferences can be deleted and default preference values will be used:</td>
</tr>
<tr>
<td></td>
<td>· APPROXIMATE_NDV_ALGORITHM</td>
</tr>
<tr>
<td></td>
<td>· AUTO_STAT_EXTENSIONS</td>
</tr>
<tr>
<td></td>
<td>· CASCADE</td>
</tr>
<tr>
<td></td>
<td>· DEGREE</td>
</tr>
<tr>
<td></td>
<td>· ESTIMATE_PERCENT</td>
</tr>
<tr>
<td></td>
<td>· GLOBAL_TEMP_TABLE_STATS</td>
</tr>
<tr>
<td></td>
<td>· GRANULARITY</td>
</tr>
<tr>
<td></td>
<td>· INCREMENTAL</td>
</tr>
<tr>
<td></td>
<td>· INCREMENTAL_LEVEL</td>
</tr>
<tr>
<td></td>
<td>· INCREMENTAL_STALENESS</td>
</tr>
<tr>
<td></td>
<td>· METHOD_OPT</td>
</tr>
<tr>
<td></td>
<td>· NO_INVALIDATE</td>
</tr>
<tr>
<td></td>
<td>· OPTIONS</td>
</tr>
<tr>
<td></td>
<td>· PREFERENCE_OVERRIDES_PARAMETER</td>
</tr>
<tr>
<td></td>
<td>· PUBLISH</td>
</tr>
<tr>
<td></td>
<td>· STALE_PERCENT</td>
</tr>
<tr>
<td></td>
<td>· STAT_CATEGORY</td>
</tr>
<tr>
<td></td>
<td>· TABLE_CACHED_BLOCKS</td>
</tr>
<tr>
<td>add_sys</td>
<td>Determines whether SYS tables will be included.</td>
</tr>
</tbody>
</table>
Table 162-19  Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPROXIMATE_NDV_ALGORITHM</td>
<td>Specifies the synopsis generation algorithm. A synopsis is a special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. Consider a synopsis as an internal management structure that samples distinct values.</td>
</tr>
<tr>
<td></td>
<td>You can specify the following preferences:</td>
</tr>
<tr>
<td></td>
<td>• REPEAT OR HYPERLOGLOG</td>
</tr>
<tr>
<td></td>
<td>This is the default. If INCREMENTAL is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format. This approach is attractive when existing performance is acceptable, and you do not want to incur the performance cost of reformatting legacy content.</td>
</tr>
<tr>
<td></td>
<td>• ADAPTIVE_SAMPLING</td>
</tr>
<tr>
<td></td>
<td>The database uses the adaptive sampling algorithm for all synopses. This is the most conservative option.</td>
</tr>
<tr>
<td></td>
<td>• HYPERLOGLOG</td>
</tr>
<tr>
<td></td>
<td>The database uses the HyperLogLog algorithm for all new and stale synopses. In contrast to dynamic sampling, the HyperLogLog algorithm uses a randomization technique. The advantages of HyperLogLog over adaptive sampling are:</td>
</tr>
<tr>
<td></td>
<td>– The accuracy of the new algorithm is similar to the original algorithm.</td>
</tr>
<tr>
<td></td>
<td>– The memory required is significantly lower, which typically leads to huge reductions in synopsis size.</td>
</tr>
<tr>
<td>AUTO_STAT_EXTENSIONS</td>
<td>Controls the automatic creation of extensions when database statistics are gathered.</td>
</tr>
<tr>
<td></td>
<td>You can set the following values:</td>
</tr>
<tr>
<td></td>
<td>• ON — When applicable, a SQL plan directive can trigger the creation of column group statistics based on usage of columns in the predicates in the workload.</td>
</tr>
<tr>
<td></td>
<td>• OFF— The database does not create column group statistics automatically. The database creates them only when the CREATE_EXTENDED_STATS function is executed, or when extended statistics are specified explicitly in the METHOD_OPT clause of DBMS_STATS. This is the default.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Determines whether index statistics are collected as part of gathering table statistics.</td>
</tr>
<tr>
<td>DEGREE</td>
<td>Determines degree of parallelism used for gathering statistics.</td>
</tr>
<tr>
<td>ESTIMATE_PERCENT</td>
<td>Determines the percentage of rows to estimate.</td>
</tr>
<tr>
<td>GLOBAL_TEMP_TABLE_STATS</td>
<td>Controls whether the statistics gathered for a global temporary table should be stored as shared statistics or session statistics.</td>
</tr>
</tbody>
</table>
Table 162-19  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANULARITY</td>
<td>Determines the granularity of statistics to collect. This value is only relevant for partitioned tables.</td>
</tr>
<tr>
<td>INCREMENTAL</td>
<td>Determines whether the global statistics of a partitioned table will be maintained without doing a full table scan.</td>
</tr>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>Controls which synopses to collect when INCREMENTAL preference is set to TRUE.</td>
</tr>
</tbody>
</table>
Table 162-19  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_STALENESS</td>
<td>Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT,USE_LOCKED_STATS,ALLOW_MIXED_FORMAT'. The parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>• USE_STALE_PERCENT—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference. For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics. Assume a different case in which STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.</td>
</tr>
<tr>
<td></td>
<td>• USE_LOCKED_STATS—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes. For example, assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS, USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.</td>
</tr>
<tr>
<td></td>
<td>• ALLOW_MIXED_FORMAT—Adaptive sampling synopses and HyperLogLog synopses are permitted to coexist.</td>
</tr>
<tr>
<td></td>
<td>• NULL—A partition or subpartition is considered stale when it has any DML changes. For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics.</td>
</tr>
</tbody>
</table>

Note that the following two executions are different:

```
EXEC DBMS_STATS.SET_TABLE_PREFS
    ('sh', 'sales', 'INCREMENTAL_STALENESS', 'NULL');

EXEC DBMS_STATS.SET_TABLE_PREFS
    ('sh', 'sales', 'INCREMENTAL_STALENESS', null);
```

The first execution uses single quotes to set the preference to the value NULL, whereas the second sets the preference to the default, which is ALLOW_MIXED_FORMAT.
### Table 162-19  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD_OPT</td>
<td>Controls column statistics collection and histogram creation. When setting preference on global, schema, database or dictionary level, only 'FOR ALL' syntax is allowed.</td>
</tr>
<tr>
<td>NO_INVALIDATE</td>
<td>Controls the invalidation of dependent cursors of the tables for which statistics are being gathered.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Determines the options parameter used in the GATHER_TABLE_STATS Procedure.</td>
</tr>
<tr>
<td>PREFERENCE_OVERRIDES_PARAMETER</td>
<td>Determines whether to override the input value of a parameter with the preference value of that parameter for a statistics operation. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• TRUE — Ignores input parameter values, and uses the value of the corresponding preference.</td>
</tr>
<tr>
<td></td>
<td>• FALSE — Obeys input parameter values. Specifying this preference does not change the order of precedence of table, global, and default.</td>
</tr>
<tr>
<td>PUBLISH</td>
<td>Determines whether the database publishes newly gathered statistics after the gathering job completes. You can gather statistics without publishing them immediately. This technique enables you to test new statistics before publishing them.</td>
</tr>
<tr>
<td>STALE_PERCENT</td>
<td>Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered.</td>
</tr>
<tr>
<td>STAT_CATEGORY</td>
<td>Specifies which statistics to import or export, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_STATS - table statistics, column statistics and index statistics (default)</td>
</tr>
<tr>
<td></td>
<td>• SYNOPSES - information to support incremental statistics</td>
</tr>
<tr>
<td></td>
<td>The value 'OBJECT_STATS, SYNOPSES' specifies table statistics, column statistics, index statistics, and synopses.</td>
</tr>
<tr>
<td>TABLE_CACHED_BLOCKS</td>
<td>Specifies the average number of blocks assumed to be cached in the buffer cache when calculating the index clustering factor.</td>
</tr>
</tbody>
</table>

### Security Model

To run this procedure, you must have the SYSDBA role or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

### Exceptions

ORA-20000: Insufficient privileges

ORA-200001: Invalid or Illegal input values

### Usage Notes

All pname arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.
Example 162-7  Examples

DBMS_STATS.DELETE_DATABASE_PREFS('CASCADE', FALSE);
DBMS_STATS.DELETE_DATABASE_PREFS('ESTIMATE_PERCENT', TRUE);

See Also:

*Oracle Database SQL Tuning Guide* to learn how to manage optimizer statistics preferences

162.7.16 DELETE_DATABASE_STATS Procedure

This procedure deletes statistics for all the tables in a database.

Syntax

```
DBMS_STATS.DELETE_DATABASE_STATS (  
  stattab          VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (  
    get_param('NO_INVALIDATE'),  
  ),
  force            BOOLEAN  DEFAULT FALSE,
  stat_category    VARCHAR2 DEFAULT DEFAULT_DEL_STAT_CATEGORY);
```

Parameters

Table 162-20  DELETE_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>When the value of this argument is TRUE, deletes statistics of tables in a database even if they are locked</td>
</tr>
</tbody>
</table>
Table 162-20  (Cont.) DELETE_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| stat_category | Statistics to delete. It accepts multiple values separated by comma:  
  - `OBJECT_STATS` - table statistics, column statistics and index statistics  
  - `SYNOPSES` - information to support incremental statistics  
  The default is `'OBJECT_STATS, SYNOPSES'` |

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

To run this procedure, you need to have the SYSDBA role or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

162.7.17 DELETE_DICTIONARY_STATS Procedure

This procedure deletes statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas).

Syntax

```sql
DBMS_STATS.DELETE_DICTIONARY_STATS (  
  stattab VARCHAR2 DEFAULT NULL,  
  statid VARCHAR2 DEFAULT NULL,  
  statown VARCHAR2 DEFAULT NULL,  
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (  
    get_param('NO_INVALIDATE')),  
  stattype VARCHAR2 DEFAULT 'ALL',  
  force BOOLEAN DEFAULT FALSE,  
  stat_category VARCHAR2 DEFAULT DEFAULT_DEL_STAT_CATEGORY);  
```

Parameters

Table 162-21  DELETE_DICTIONARY_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
</tbody>
</table>
Table 162-21  (Cont.) DELETE_DICTIONARY_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>stattype</td>
<td>Statistics type</td>
</tr>
<tr>
<td>force</td>
<td>When the value of this argument is TRUE, deletes statistics of tables in a database even if they are locked</td>
</tr>
<tr>
<td>stat_category</td>
<td>Statistics to delete. It accepts multiple values separated by comma:</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_STATS - table statistics, column statistics and index statistics</td>
</tr>
<tr>
<td></td>
<td>• SYNOPSES - information to support incremental statistics</td>
</tr>
<tr>
<td></td>
<td>The default is 'OBJECT_STATS, SYNOPSES'</td>
</tr>
</tbody>
</table>

Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table, may need to upgrade it

162.7.18 DELETE_FIXED_OBJECTS_STATS Procedure

This procedure deletes statistics of all fixed tables.

Syntax

```
DBMS_STATS.DELETE_FIXED_OBJECTS_STATS (  
  stattab VARCHAR2 DEFAULT NULL,  
  statid VARCHAR2 DEFAULT NULL,  
  statown VARCHAR2 DEFAULT NULL,  
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (  
    get_param('NO_INVALIDATE')),  
  force BOOLEAN DEFAULT FALSE);  
```
Parameters

Table 162-22  DELETE_FIXED_OBJECTS_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>The user statistics table identifier describing from where to delete the current statistics. If stattab is NULL, the statistics will be deleted directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>The (optional) identifier to associate with these statistics within stattab. This only applies if stattab is not NULL.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure</td>
</tr>
<tr>
<td>force</td>
<td>Ignores the statistics lock on objects and deletes the statistics if set to TRUE</td>
</tr>
</tbody>
</table>

Usage Notes

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.

Exceptions

ORA-20000: Insufficient privileges
ORA-20002: Bad user statistics table, may need to upgrade it

162.7.19 DELETE_INDEX_STATS Procedure

This procedure deletes index-related statistics.

Syntax

```sql
DBMS_STATS.DELETE_INDEX_STATS (    ownname VARCHAR2,    indname VARCHAR2,    partname VARCHAR2 DEFAULT NULL,    stattab VARCHAR2 DEFAULT NULL,    statid VARCHAR2 DEFAULT NULL,    cascade_parts BOOLEAN DEFAULT TRUE,    statown VARCHAR2 DEFAULT NULL,    no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (    get_param('NO_INVALIDATE')),    stattype VARCHAR2 DEFAULT 'ALL',    force BOOLEAN DEFAULT FALSE,    stat_category VARCHAR2 DEFAULT DEFAULT_DEL_STAT_CATEGORY);```
Parameters

Table 162-23  DELETE_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the index partition for which to delete the statistics. If the index is partitioned and if partname is NULL, then index statistics are deleted at the global level.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>cascade_parts</td>
<td>If the index is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this index for all underlying partitions as well.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>stattype</td>
<td>Statistics type</td>
</tr>
<tr>
<td>force</td>
<td>When value of this argument is TRUE, deletes index statistics even if locked</td>
</tr>
</tbody>
</table>
| stat_category | Statistics to delete. It accepts multiple values separated by comma:

- OBJECT_STATS - table statistics, column statistics and index statistics
- SYNOPSES - information to support incremental statistics

The default is 'OBJECT_STATS, SYNOPSES'

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20005: Object statistics are locked

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.
162.7.20 DELETE_PENDING_STATS Procedure

This procedure is used to delete the pending statistics that have been collected but have not been published.

Syntax

```
DBMS_STATS.DELETE_PENDING_STATS (
    ownname    IN  VARCHAR2  DEFAULT USER,
    tabname    IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
</tbody>
</table>

Security Model

To run this procedure, you need to have the same privilege for gathering statistics on the tables that will be affected by this procedure. The default owner is the user who runs the procedure.

Exceptions

ORA-20000: Insufficient privileges

Usage Notes

If the parameter `tabname` is NULL delete applies to all tables of the specified schema.

Examples

```
DBMS_STATS.DELETE_PENDING_STATS('SH', 'SALES');
```

162.7.21 DELETE_PROCESSING_RATE Procedure

This procedure deletes the processing rate of a given statistics source. If the source is not specified, it deletes the statistics of all the sources.

Syntax

```
DBMS_STATS.DELETE_PROCESSING_RATE (
    stat_source IN  VARCHAR2 DEFAULT NULL);
```
Parameters

Table 162-25  DELETE_PROCESSING_RATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| stat_source | Source of processing rates:  
  - 'MANUAL': values set by the user manually using the SET_PROCESSING_RATE Procedure  
  - 'CALIBRATION': values collected by the calibration GATHER_PROCESSING_RATE Procedure run explicitly by the user  
  - 'FEEDBACK': values obtained by time feedback |

Usage Notes

You require the OPTIMIZER_PROCESSING_RATE role to run this procedure since AUTO DOP uses processing rates to determine the optimal degree of parallelism for a SQL statement.

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid or illegal input value

162.7.22 DELETE_SCHEMA_PREFS Procedure

This procedure is used to delete the statistics preferences of all the tables owned by the specified owner name.

Syntax

```sql
DBMS_STATS.DELETE_SCHEMA_PREFS (  
    ownername IN VARCHAR2,  
    pname IN VARCHAR2);
```

Parameters

Table 162-26  DELETE_SCHEMA_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownername</td>
<td>Owner name</td>
</tr>
</tbody>
</table>
Table 162-26  (Cont.) DELETE_SCHEMA_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| pname     | Preference name. The existing value for following preferences can be deleted and default preference values will be used:  
  - APPROXIMATE_NDV_ALGORITHM  
  - AUTO_STAT_EXTENSIONS  
  - CASCADE  
  - DEGREE  
  - ESTIMATE_PERCENT  
  - GLOBAL_TEMP_TABLE_STATS  
  - GRANULARITY  
  - INCREMENTAL  
  - INCREMENTAL_LEVEL  
  - INCREMENTAL_STALENESS  
  - METHOD_OPT  
  - NO_INVALIDATE  
  - OPTIONS  
  - PREFERENCE_OVERRIDES_PARAMETER  
  - PUBLISH  
  - STALE_PERCENT  
  - TABLE_CACHED_BLOCKS |

Table 162-27  Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
</table>
| APPROXIMATE_NDV_ALGORITHM | Specifies the synopsis generation algorithm. A synopsis is special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. You can consider a synopsis as an internal management structure that samples distinct values.  
  You can set the following values:  
  - REPEAT OR HYPERLOGLOG  
    - This is the default. If INCREMENTAL is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format.  
  - ADAPTIVE_SAMPLING  
    - The database uses the adaptive sampling algorithm for all synopses.  
  - HYPERLOGLOG  
    - The database uses the HyperLogLog algorithm for all new and stale synopses. |
<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_STAT_EXTENSIONS</td>
<td>Controls the automatic creation of extensions when database statistics are gathered.</td>
</tr>
<tr>
<td></td>
<td>You can set the following values:</td>
</tr>
<tr>
<td></td>
<td>- <strong>ON</strong> — When applicable, a SQL plan directive can trigger the creation of column group statistics based on usage of columns in the predicates in the workload.</td>
</tr>
<tr>
<td></td>
<td>- <strong>OFF</strong> — The database does not create column group statistics automatically. The database creates them only when the CREATE_EXTENDED_STATS function is executed, or when extended statistics are specified explicitly in the METHOD_OPT clause of DBMS_STATS. This is the default.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Determines whether index statistics are collected as part of gathering table statistics.</td>
</tr>
<tr>
<td>COORDINATOR_TRIGGER_SHARD</td>
<td>User of each shard uses this preference to determine whether to allow shard coordinator to interact with the statistics gathering in each shards.</td>
</tr>
<tr>
<td></td>
<td>While gathering the statistics in shard coordinator, if the statistics in one of the shards are not up to date, the shard coordinator will try to trigger the statistics gathering in that shard. By using this preference, user can execute or ignore that command from the shard coordinator.</td>
</tr>
<tr>
<td></td>
<td>You can set the following values:</td>
</tr>
<tr>
<td></td>
<td>- <strong>TRUE</strong> — Allows the shard coordinator trigger the statistics gathering on sharded table in local shard if the statistics on local shard are stale.</td>
</tr>
<tr>
<td></td>
<td>- <strong>FALSE</strong> — Ignores the statistics gathering command triggered from the shard coordinator.</td>
</tr>
<tr>
<td></td>
<td>The default value is <strong>FALSE</strong>.</td>
</tr>
<tr>
<td>DEGREE</td>
<td>Determines the degree of parallelism used for gathering statistics.</td>
</tr>
<tr>
<td>ESTIMATE_PERCENT</td>
<td>The value determines the percentage of rows to estimate.</td>
</tr>
<tr>
<td>METHOD_OPT</td>
<td>Controls column statistics collection and histogram creation. When setting preferences at the global, schema, database, or dictionary level, only FOR ALL syntax is allowed.</td>
</tr>
<tr>
<td>NO_INVALIDATE</td>
<td>The value controls the invalidation of dependent cursors of the tables for which statistics are being gathered.</td>
</tr>
<tr>
<td>GRANULARITY</td>
<td>The value determines granularity of statistics to collect (only pertinent if the table is partitioned)</td>
</tr>
<tr>
<td>PUBLISH</td>
<td>This value determines whether or not newly gathered statistics will be published once the gather job has completed.</td>
</tr>
<tr>
<td>INCREMENTAL</td>
<td>This value determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan.</td>
</tr>
</tbody>
</table>
Table 162-27  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>This value controls what synopses to collect when INCREMENTAL preference is set to TRUE.</td>
</tr>
</tbody>
</table>
Table 162-27  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_STALENESS</td>
<td>Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT,USE_LOCKED_STATS,ALLOW_MIXED_FORMAT'. The parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>- <strong>USE_STALE_PERCENT</strong>—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference. For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics. Assume a different case in which STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.</td>
</tr>
<tr>
<td></td>
<td>- <strong>USE_LOCKED_STATS</strong>—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes. For example, assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS, USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.</td>
</tr>
<tr>
<td></td>
<td>- <strong>ALLOW_MIXED_FORMAT</strong>—Adaptive sampling synopses and HyperLogLog synopses are permitted to coexist.</td>
</tr>
<tr>
<td></td>
<td>- <strong>NULL</strong>—A partition or subpartition is considered stale when it has any DML changes. For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics. Note that the following two executions are different:</td>
</tr>
</tbody>
</table>

EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'sales', 'INCREMENTAL_STALENESS', 'NULL');

EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'sales', 'INCREMENTAL_STALENESS', null);
## Table 162-27  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
</table>
| ROOT_TRIGGER_PDB      | The application PDB user, uses this preference to determine whether to allow the application root to interact with the statistics gathering in PDB. During the statistics gathering of a metadata linked table in the application root, if the statistics in a PDB are in stale state, the application root triggers the statistics gathering for the particular PDB. Using this preference, the user can either execute or ignore the command from the application root. You can set the following values:  
  - TRUE—Allows the application root trigger the statistics gathering on metadata linked table in application PDB if the statistics on PDB are stale.  
  - FALSE—Ignores the statistics gathering command triggered from application root.  
  The default value is FALSE. |
| STALE_PERCENT         | This value determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered.                                                      |
| GLOBAL_TEMP_TABLE_STATS| This controls whether the statistics gathered for a global temporary table should be stored as shared statistics or session statistics.                                                                           |
| TABLE_CACHED_BLOCKS   | The average number of blocks cached in the buffer cache for any table we can assume when gathering the index clustering factor.                                                                               |
| OPTIONS               | Determines the options parameter used in the GATHER_TABLE_STATS Procedure.                                                                                                                               |

### Note:

CDB root, different from application root, never triggers statistics gathering on the PDBs and it is not controlled by this preference.

**Security Model**

To run this procedure, you must be the object owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.
Exceptions

ORA-20000: Insufficient privileges / Schema "<schema>" does not exist

ORA-20001: Invalid or Illegal input values

Usage Notes

All arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

Examples

DBMS_STATS.DELETE_SCHEMA_PREFS('SH', 'CASCADE');
DBMS_STATS.DELETE_SCHEMA_PREFS('SH', 'ESTIMATE_PERCENT');
DBMS_STATS.DELETE_SCHEMA_PREFS('SH', 'DEGREE');

See Also:

Oracle Database SQL Tuning Guide to learn how to manage optimizer statistics preferences

162.7.23 DELETE_SCHEMA_STATS Procedure

This procedure deletes statistics for an entire schema.

Syntax

DBMS_STATS.DELETE_SCHEMA_STATS (ownname VARCHAR2,
stattab VARCHAR2 DEFAULT NULL,
statid VARCHAR2 DEFAULT NULL,
statown VARCHAR2 DEFAULT NULL,
no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (get_param('NO_INVALIDATE')),
force BOOLEAN DEFAULT FALSE);

Parameters

Table 162-28  DELETE_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>
### Table 162-28  (Cont.) DELETE_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>When value of this argument is TRUE, deletes statistics of tables in a schema even if locked</td>
</tr>
</tbody>
</table>

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

### 162.7.24 DELETE_SYSTEM_STATS Procedure

This procedure deletes workload statistics (collected using the 'INTERVAL' or 'START' and 'STOP' options) and resets the default to noworkload statistics (collected using 'NOWORKLOAD' option), if stattab is not specified. If stattab is specified, the subprogram deletes all system statistics with the associated statid from the stattab.

#### Syntax

```sql
DBMS_STATS.DELETE_SYSTEM_STATS (
    stattab VARCHAR2 DEFAULT NULL,
    statid  VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Identifier of the user statistics table where the statistics will be saved</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics saved in the stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
</tbody>
</table>

#### Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20002: Bad user statistics table; may need to be upgraded

Usage Notes

To run this procedure, you need the GATHER_SYSTEM_STATISTICS role.

162.7.25 DELETE_TABLE_PREFS Procedure

This procedure deletes the optimizer statistics preferences of the specified table in the specified schema.

Syntax

```sql
DBMS_STATS.DELETE_TABLE_PREFS ( 
    ownname    IN  VARCHAR2, 
    tabname    IN  VARCHAR2, 
    pname      IN  VARCHAR2);
```

Parameters

Table 162-30  DELETE_TABLE_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
</tbody>
</table>
| pname       | Preference name. The existing value for following preferences can be deleted and default preference values will be used:  
  - APPROXIMATE_NDV_ALGORITHM  
  - AUTO_STAT_EXTENSIONS  
  - CASCADE  
  - DEGREE  
  - ESTIMATE_PERCENT  
  - GRANULARITY  
  - GLOBAL_TEMP_STATS  
  - INCREMENTAL  
  - INCREMENTAL_LEVEL  
  - INCREMENTAL_STALENESS  
  - METHOD_OPT  
  - NO_INVALIDATE  
  - OPTIONS  
  - PREFERENCE_OVERRIDES_PARAMETER  
  - PUBLISH  
  - STALE_PERCENT  
  - STAT_CATEGORY  
  - TABLE_CACHED_BLOCKS |
### Table 162-31  Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
</table>
| APPROXIMATE_NDV_ALGORITHM          | Specifies the synopsis generation algorithm. A synopsis is special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. Consider a synopsis as an internal management structure that samples distinct values. You can specify the following preferences:  
  - REPEAT OR HYPERLOGLOG  
    - This is the default. If INCREMENTAL is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format. This approach is attractive when existing performance is acceptable, and you do not want to incur the performance cost of reformatting legacy content.  
  - ADAPTIVE_SAMPLING  
    - The database uses the adaptive sampling algorithm for all synopses. This is the most conservative option.  
  - HYPERLOGLOG  
    - The database uses the HyperLogLog algorithm for all new and stale synopses. In contrast to dynamic sampling, the HyperLogLog algorithm uses a randomization technique. The advantages of HyperLogLog over adaptive sampling are:  
      - The accuracy of the new algorithm is similar to the original algorithm.  
      - The memory required is significantly lower, which typically leads to huge reductions in synopsis size. |
| AUTO_STAT_EXTENSIONS                | Controls the automatic creation of extensions when database statistics are gathered. You can set the following values:  
  - ON  
    - When applicable, a SQL plan directive can trigger the creation of column group statistics based on usage of columns in the predicates in the workload.  
  - OFF  
    - The database does not create column group statistics automatically. The database creates them only when the CREATE_EXTENDED_STATS function is executed, or when extended statistics are specified explicitly in the METHOD_OPT clause of DBMS_STATS. This is the default. |
| CASCADE                            | Determines whether index statistics are collected as part of gathering table statistics.                                                                                                                   |
| DEGREE                             | Determines the degree of parallelism used for gathering statistics.                                                                                                                                       |
| ESTIMATE_PERCENT                   | Determines the percentage of rows to estimate.                                                                                                                                                            |
| GRANULARITY                        | Determines granularity of statistics to collect. This value is only relevant for partitioned tables.                                                                                                       |
Table 162-31  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_TEMP_TABLE_STATS</td>
<td>Controls whether the statistics gathered for a global temporary table should be stored as shared statistics or session statistics.</td>
</tr>
<tr>
<td>INCREMENTAL</td>
<td>This value determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan.</td>
</tr>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>This value controls what synopses to collect when INCREMENTAL preference is set to TRUE.</td>
</tr>
</tbody>
</table>
Table 162-31  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
</table>
| INCREMENTAL_STALENESS | Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT,USE_LOCKED_STATS,ALLOW_MIXED_FORMAT'. The parameter accepts the following values:  
  - USE_STALE_PERCENT—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference.  
    For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics.  
    Assume a different case in which STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS,USE_STALE_PERCENT' for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.  
  - USE_LOCKED_STATS—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes.  
    For example, assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS,USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.  
  - ALLOW_MIXED_FORMAT—Adaptive sampling synopses and HyperLogLog synopses are permitted to coexist.  
  - NULL—A partition or subpartition is considered stale when it has any DML changes.  
    For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics. |
Table 162-31  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note that the following two executions are different:</strong></td>
<td></td>
</tr>
<tr>
<td><code>EXEC DBMS_STATS.SET_TABLE_PREFS</code></td>
<td></td>
</tr>
<tr>
<td>(<code>'sh', 'sales', 'INCREMENTAL_STALENESS', 'NULL')</code></td>
<td></td>
</tr>
<tr>
<td><code>EXEC DBMS_STATS.SET_TABLE_PREFS</code></td>
<td></td>
</tr>
<tr>
<td>(<code>'sh', 'sales', 'INCREMENTAL_STALENESS', null)</code></td>
<td></td>
</tr>
<tr>
<td>The first execution uses single quotes to set the preference to the value NULL, whereas the second sets the preference to the default, which is ALLOW_MIXED_FORMAT.</td>
<td></td>
</tr>
<tr>
<td><strong>METHOD_OPT</strong></td>
<td>Controls column statistics collection and histogram creation. When setting preference at the global, schema, database, or dictionary level, only 'FOR ALL' syntax is allowed.</td>
</tr>
<tr>
<td><strong>NO_INVALIDATE</strong></td>
<td>The value controls the invalidation of dependent cursors of the tables for which statistics are being gathered.</td>
</tr>
<tr>
<td><strong>OPTIONS</strong></td>
<td>Determines the options parameter used in the GATHER_TABLE_STATS Procedure.</td>
</tr>
<tr>
<td><strong>PREFERENCE_OVERRIDES_PARAMETER</strong></td>
<td>Determines whether to override the input value of a parameter with the preference value of that parameter for a statistics operation. Possible values are:</td>
</tr>
<tr>
<td>- <strong>TRUE</strong> — Ignores input parameter values, and uses the value of the corresponding preference.</td>
<td></td>
</tr>
<tr>
<td>- <strong>FALSE</strong> — Obeys input parameter values.</td>
<td></td>
</tr>
<tr>
<td>Specifying this preference does not change the order of precedence of table, global, and default.</td>
<td></td>
</tr>
<tr>
<td><strong>PUBLISH</strong></td>
<td>Determines whether newly gathered statistics will be published after the statistics gathering job has completed.</td>
</tr>
<tr>
<td><strong>STALE_PERCENT</strong></td>
<td>Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered.</td>
</tr>
<tr>
<td><strong>STAT_CATEGORY</strong></td>
<td>Specifies which statistics to import or export, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td>- <strong>OBJECT_STATS</strong> - table statistics, column statistics and index statistics (default)</td>
<td></td>
</tr>
<tr>
<td>- <strong>SYNOPSES</strong> - information to support incremental statistics</td>
<td></td>
</tr>
<tr>
<td>The value 'OBJECT_STATS, SYNOPSES' specifies table statistics, column statistics, index statistics, and synopses.</td>
<td></td>
</tr>
</tbody>
</table>
Table 162-31  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_CACHED_BLOCKS</td>
<td>Specifies the average number of blocks assumed to be in the buffer cache when calculating the index clustering factor.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Invalid or Illegal input values

Usage Notes

- To run this procedure, you need to connect as owner of the table, be granted ANALYZE privilege on the table, or ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

Examples

DBMS_STATS.DELETE_TABLE_PREFS('SH', 'SALES', 'CASCADE');
DBMS_STATS.DELETE_TABLE_PREFS('SH', 'SALES', 'DEGREE');

See Also:

Oracle Database SQL Tuning Guide to learn how to manage optimizer statistics preferences

162.7.26 DELETE_TABLE_STATS Procedure

This procedure deletes table-related statistics.

Syntax

```sql
DBMS_STATS.DELETE_TABLE_STATS (  
    ownername VARCHAR2,  
    tabname VARCHAR2,  
    partname VARCHAR2 DEFAULT NULL,  
    stattab VARCHAR2 DEFAULT NULL,  
    statid VARCHAR2 DEFAULT NULL,  
    cascade_parts BOOLEAN DEFAULT TRUE,  
    cascade_columns BOOLEAN DEFAULT TRUE,  
    cascade_indexes BOOLEAN DEFAULT TRUE,  
    statown VARCHAR2 DEFAULT NULL,  
    no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (  
        get_param('NO_INVALIDATE') ),  
    force BOOLEAN DEFAULT FALSE);
```
Parameters

**Table 162-32  DELETE_TABLE_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table [sub]partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>cascade_parts</td>
<td>If the table is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this table for all underlying partitions as well</td>
</tr>
<tr>
<td>cascade_columns</td>
<td>Indicates that DELETE_COLUMN_STATS should be called for all underlying columns (passing the cascade_parts parameter)</td>
</tr>
<tr>
<td>cascade_indexes</td>
<td>Indicates that DELETE_INDEX_STATS should be called for all underlying indexes (passing the cascade_parts parameter)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>When value of this argument is TRUE, deletes table statistics even if locked</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table, may need to upgrade it

ORA-20005: Object statistics are locked

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.
162.7.27 DIFF_TABLE_STATS_IN_HISTORY Function

This function can be used to compare statistics for a table from two timestamps in past and compare the statistics as of that timestamps.

Syntax

```
DBMS_STATS.DIFF_TABLE_STATS_IN_HISTORY(
    ownname        IN  VARCHAR2,
    tabname        IN  VARCHAR2,
    time1          IN  TIMESTAMP WITH TIME ZONE,
    time2          IN  TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    pctthreshold   IN  NUMBER  DEFAULT 10)
RETURN DiffRepTab pipelined;
```

Parameters

Table 162-33  DIFF_TABLE_STATS_IN_HISTORY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner of the table. Specify NULL for current schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Table for which statistics are to be compared</td>
</tr>
<tr>
<td>time1</td>
<td>First timestamp 1</td>
</tr>
<tr>
<td>time2</td>
<td>Second timestamp 2</td>
</tr>
<tr>
<td>pctthreshold</td>
<td>The function reports difference in statistics only if it exceeds this limit. The default value is 10.</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Usage Notes

If the second timestamp is NULL, then the function compares the current statistics in dictionary with the statistics as of the other timestamp.

162.7.28 DIFF_TABLE_STATS_IN_PENDING Function

This function compares pending statistics to either the current statistics in the data dictionary, or user-specified historical statistics.

Syntax

```
DBMS_STATS.DIFF_TABLE_STATS_IN_PENDING(
    ownname        IN  VARCHAR2,
    tabname        IN  VARCHAR2,
    timestamp      IN  TIMESTAMP WITH TIME ZONE,
```


Parameters

Table 162-34  DIFF_TABLE_STATS_IN_PENDING Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner of the table. Specify NULL for the current schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Table for which statistics are to be compared.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Timestamp in the statistics history that corresponds to the desired statistics. If the timestamp is NULL, then this function compares the current statistics in the dictionary with the pending statistics (default).</td>
</tr>
<tr>
<td>pctthreshold</td>
<td>Limit for reporting. The function reports difference in statistics only if it exceeds the specified limit. The default value is 10.</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table, or you must have the ANALYZE ANY privilege. For objects owned by SYS, you must be either the owner of the table, or you must have either the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

162.7.29 DIFF_TABLE_STATS_IN_STATTAB Function

This function can be used to compare statistics for a table from two different sources.

The statistics can be drawn from

- two different user statistics tables
- a single user statistics table containing 2 sets of statistics that can be identified using statids
- a user statistics table and dictionary

The function also compares the statistics of the dependent objects (indexes, columns, partitions) as well. It displays statistics of the object(s) from both sources if the difference between those statistics exceeds a certain threshold (%). The threshold can be specified as an argument to the function. The statistics corresponding to the first source (stattab1 or statid1) will be used as basis for computing the difference percentage.

Syntax

```sql
DBMS_STATS.DIFF_TABLE_STATS_IN_STATTAB(
    ownname IN VARCHAR2,
    tabname IN VARCHAR2,
    stattab1 IN VARCHAR2,
    stattab2 IN VARCHAR2 DEFAULT NULL,
    pctthreshold IN NUMBER DEFAULT 10,
    statid1 IN VARCHAR2 DEFAULT NULL,
    statid2 IN VARCHAR2 DEFAULT NULL,
    stattab1own IN VARCHAR2 DEFAULT NULL,
    stattab2own IN VARCHAR2 DEFAULT NULL)
RETURN DiffRepTab pipelined;
```
Parameters

Table 162-35  DIFF_TABLE_STATS_IN_STATTAB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner of the table. Specify NULL for current schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Table for which statistics are to be compared</td>
</tr>
<tr>
<td>stattab1</td>
<td>User statistics table 1</td>
</tr>
<tr>
<td>stattab2</td>
<td>User statistics table 2. If NULL, statistics in stattab1 is compared with current statistics in dictionary. This is the default. Specify same table as stattab1 to compare two sets within the statistics table (see statid below).</td>
</tr>
<tr>
<td>pctthreshold</td>
<td>The function reports difference in statistics only if it exceeds this limit. The default value is 10.</td>
</tr>
<tr>
<td>stadid1</td>
<td>(optional) Identifies statistics set within stattab1.</td>
</tr>
<tr>
<td>stadid2</td>
<td>(optional) Identifies statistics set within stattab2</td>
</tr>
<tr>
<td>stattabown</td>
<td>Schema containing stattab1 (if other than ownname)</td>
</tr>
<tr>
<td>stattab2own</td>
<td>Schema containing stattab2 (if other than ownname)</td>
</tr>
</tbody>
</table>

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

162.7.30 DROP_ADVISOR_TASK Procedure

This procedure drops the specified Optimizer Statistics Advisor task.

Syntax

```sql
DBMS_STATS.DROP_ADVISOR_TASK (  
    task_name IN VARCHAR2);
```

Parameters

Table 162-36  DROP_ADVISOR_TASK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
This subprogram executes using invoker's rights.

Exceptions

- **ORA-20000**: Insufficient privileges
- **ORA-20001**: Invalid input values
- **ORA-20012**: Optimizer Statistics Advisor errors

Example 162-8  Dropping an Optimizer Statistics Advisor Task

This example drops the Optimizer Statistics Advisor task named `my_task`:

```sql
EXEC DBMS_STATS.DROP_ADVISOR_TASK('my_task');
```

162.7.31 DROP_EXTENDED_STATS Procedure

This function drops the statistics entry that is created for the user specified extension. This cancels the effects of the `CREATE_EXTENDED_STATS` Function.

**Syntax**

```sql
DBMS_STATS.DROP_EXTENDED_STATS ( 
    ownname    VARCHAR2, 
    tabname    VARCHAR2, 
    extension  VARCHAR2); 
```

**Parameters**

**Table 162-37  DROP_EXTENDED_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name of a table</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
<tr>
<td>extension</td>
<td>Can be either a column group or an expression. Suppose the specified table</td>
</tr>
</tbody>
</table>
|             | has two column c1, c2. An example column group can be 
|             | "(c1, c2)" and an example expression can be "(c1 + c2)".                    |

**Exceptions**

- **ORA-20000**: Insufficient privileges or extension does not exist
- **ORA-20001**: Error when processing extension

**Usage Notes**

- To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
- If no extended statistics set is created for the extension, this function throws an error.
162.7.32 DROP_STAT_TABLE Procedure

This procedure drops a user statistics table.

Syntax

```
DBMS_STATS.DROP_STAT_TABLE (  
    ownname VARCHAR2,  
    stattab VARCHAR2);  
```

Parameters

Table 162-38  DROP_STAT_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Table does not exists or insufficient privileges.

Usage Notes

To invoke this procedure you need the privileges for dropping the specified table.

162.7.33 EXECUTE_ADVISOR_TASK Function

This function executes a previously created Optimizer Statistics Advisor task.

Syntax

```
DBMS_STATS.EXECUTE_ADVISOR_TASK (  
    task_name        IN   VARCHAR2,  
    execution_name   IN   VARCHAR2 := NULL)  
RETURN VARCHAR2;  
```

Parameters

Table 162-39  EXECUTE_ADVISOR_TASK Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>Name of the Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td>execution_name</td>
<td>A name that qualifies and identifies an advisor execution. If not specified, then the advisor automatically generates it. If the specified execution conflicts with the name of an existing execution, then the function returns an error.</td>
</tr>
</tbody>
</table>
Security Model

Note the following:

- To execute this subprogram, you must have the **ADVISOR** privilege.
- You must be the owner of the task.
- You can execute this subprogram for **AUTO_STATS_ADVISOR_TASK**, which is predefined.
- This subprogram executes using invoker’s rights.

The results of performing this task depend on the privileges of the executing user:

- **SYSTEM level**
  - Only users with both the **ANALYZE ANY** and **ANALYZE ANY DICTIONARY** privileges can perform this task on system-level rules.
- **Operation level**
  - The results depend on the following privileges:
    - Users with both the **ANALYZE ANY** and **ANALYZE ANY DICTIONARY** privileges can perform this task for all statistics operations.
    - Users with the **ANALYZE ANY** privilege but not the **ANALYZE ANY DICTIONARY** privilege can perform this task for statistics operations related to any schema except **SYS**.
    - Users with the **ANALYZE ANY DICTIONARY** privilege but not the **ANALYZE ANY** privilege can perform this task for statistics operations related to their own schema and the **SYS** schema.
    - Users with neither the **ANALYZE ANY** nor the **ANALYZE ANY DICTIONARY** privilege can only perform this operation for statistics operations relating to their own schema.
- **Object level**
  - Users can perform this task for any object for which they have statistics collection privileges.

Exceptions

- **ORA-20000**: Insufficient privileges
- **ORA-20001**: Invalid input values
- **ORA-20012**: Optimizer Statistics Advisor errors

Returns

This function returns the name of the new execution.

Usage Notes

The results of the execution depend on user privileges and the type of rules:

- **System**
  - To perform the operation on system-level rules, you must have both the **ANALYZE ANY** and **ANALYZE ANY DICTIONARY** privileges.
- **Operation**
If you have the `ANALYZE ANY` and `ANALYZE ANY DICTIONARY` privileges, then you can execute this function for all operations. If you have only the `ANALYZE ANY` privilege, then you can execute this function for operations related to any schemas except `SYS`. If you have only the `ANALYZE ANY DICTIONARY` privilege, then you can execute this function for operations related to any schemas, including `SYS`. If you have neither the `ANALYZE ANY` nor the `ANALYZE ANY DICTIONARY` privilege, then you can execute this function only for operations in your own schema.

- **Object**

  If you have the privilege to collect statistics for an object, then you can execute this function for the object.

**Example 162-9  Creating and Executing a Task**

This example creates an Optimizer Statistics Advisor task named `my_task`, and then executes it.

```sql
DECLARE
  v_tname   VARCHAR2(128) := 'my_task';
BEGIN
  -- create a task
  v_tname := DBMS_STATS.CREATE_ADVISOR_TASK(v_tname);
  -- execute the task
  v_tname := DBMS_STATS.EXECUTE_ADVISOR_TASK(v_tname);
END;
```

### 162.7.34 EXPORT_COLUMN_STATS Procedure

This procedure exports statistics for a specified column and stores them in the user statistics table identified by `stattab`.

**Syntax**

```sql
DBMS_STATS.EXPORT_COLUMN_STATS (  
  ownname  VARCHAR2,  
  tabname  VARCHAR2,  
  colname  VARCHAR2,  
  partname VARCHAR2 DEFAULT NULL,  
  stattab  VARCHAR2,  
  statid   VARCHAR2 DEFAULT NULL,  
  statown  VARCHAR2 DEFAULT NULL);
```

**Parameters**

**Table 162-40  EXPORT_COLUMN_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column or extension</td>
</tr>
</tbody>
</table>
Table 162-40  (Cont.) EXPORT_COLUMN_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partname</td>
<td>Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition column statistics are exported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table or have the ANALYZE ANY privilege.
For objects owned by SYS, you must be either the owner of the table or have either the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

Oracle Database does not support export or import of statistics across databases of different character sets.

162.7.35 EXPORT_DATABASE_PREFS Procedure

This procedure is used to export the statistics preferences of all the tables, excluding the tables owned by Oracle. These tables can by included by passing TRUE for the add_sys parameter.

Syntax

```sql
DBMS_STATS.EXPORT_DATABASE_PREFS (
    stattab IN VARCHAR2,
    statid IN VARCHAR2 DEFAULT NULL,
    statown IN VARCHAR2 DEFAULT NULL,
    add_sys IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 162-41  EXPORT_DATABASE_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Statistics table name to where statistics should be exported</td>
</tr>
<tr>
<td>statid</td>
<td>(Optional) Identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if other than ownname)</td>
</tr>
<tr>
<td>add_sys</td>
<td>Value TRUE will include the Oracle-owned tables</td>
</tr>
</tbody>
</table>
Exceptions

ORA-20000: Insufficient privileges

Usage Notes

- To run this procedure, you need to have the SYSDBA role, or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

Examples

DBMS_STATS.EXPORT_DATABASE_PREFS ('STATTAB', statown=>'SH');

162.7.36 EXPORT_DATABASE_STATS Procedure

This procedure exports statistics for all objects in the database and stores them in the user statistics tables identified by statown.stattab.

Syntax

DBMS_STATS.EXPORT_DATABASE_STATS (stattab VARCHAR2, statid VARCHAR2 DEFAULT NULL, statown VARCHAR2 DEFAULT NULL, stat_category VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);

Parameters

Table 162-42   EXPORT_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>stat_category</td>
<td>Specifies what statistics to import, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_STATS - table statistics, column statistics and index statistics (Default)</td>
</tr>
<tr>
<td></td>
<td>• SYNOPSIS - information to support incremental statistics</td>
</tr>
<tr>
<td></td>
<td>If 'OBJECT_STATS, SYNOPSIS' is specified, table statistics, column statistics, index statistics and synopses are deleted.</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table or have the ANALYZE ANY privilege. For objects owned by SYS, you must be either the owner of the table or have either the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.
Exceptions
ORA-20000: Object does not exist or insufficient privileges

Usage Notes
Oracle Database does not support export or import of statistics across databases of different character sets.

162.7.37 EXPORT_DICTIONARY_STATS Procedure

This procedure exports statistics for all data dictionary schemas (SYS, SYSTEM, and RDBMS component schemas) and stores them in the user statistics table identified by stattab.

Syntax

```
DBMS_STATS.EXPORT_DICTIONARY_STATS (  
    stattab         VARCHAR2,  
    statid          VARCHAR2 DEFAULT NULL,  
    statown         VARCHAR2 DEFAULT NULL,  
    stat_category   VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

Parameters

Table 162-43  EXPORT_DICTIONARY_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>stat_category</td>
<td>Specifies what statistics to import, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_STATS - table statistics, column statistics and index statistics (Default)</td>
</tr>
<tr>
<td></td>
<td>• SYNOPTES - information to support incremental statistics</td>
</tr>
<tr>
<td></td>
<td>If 'OBJECT_STATS, SYNOPTES' is specified, table statistics, column statistics, index statistics and synopses are deleted.</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table or have the ANALYZE ANY privilege. For objects owned by SYS, you must be either the owner of the table or have either the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Exceptions
ORA-20000: Object does not exist or insufficient privileges
ORA-20002: Bad user statistics table, may need to upgrade it
Usage Notes

Oracle Database does not support export or import of statistics across databases of different character sets.

162.7.38 EXPORT_FIXED_OBJECTS_STATS Procedure

This procedure exports statistics for fixed tables and stores them in the user statistics table identified by `stattab`.

Syntax

```sql
DBMS_STATS.EXPORT_FIXED_OBJECTS_STATS (
    stattab  VARCHAR2,
    statid   VARCHAR2 DEFAULT NULL,
    statown  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within <code>stattab</code></td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing <code>stattab</code> (if different from current schema)</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table or have the `ANALYZE ANY` privilege. For objects owned by `SYS`, you must be either the owner of the table or have either the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.

Exceptions

- ORA-20000: Object does not exist or insufficient privileges
- ORA-20002: Bad user statistics table, may need to upgrade it

Usage Notes

Oracle Database does not support export or import of statistics across databases of different character sets.

162.7.39 EXPORT_INDEX_STATS Procedure

This procedure retrieves statistics for a particular index and stores them in the user statistics table identified by `stattab`.

Syntax

```sql
DBMS_STATS.EXPORT_INDEX_STATS (
    ownname  VARCHAR2,
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner of the index</td>
</tr>
</tbody>
</table>

...
Parameters

Table 162-45  EXPORT_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the index partition. If the index is partitioned and if partname is NULL, then global and partition index statistics are exported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

- To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
- Oracle does not support export or import of statistics across databases of different character sets.

162.7.40 EXPORT_PENDING_STATS Procedure

This procedure is used to export the statistics gathered and stored as pending.

Syntax

```sql
DBMS_STATS.EXPORT_PENDING_STATS (    ownername IN VARCHAR2 DEFAULT USER,    tabname IN VARCHAR2,    stattab IN VARCHAR2,    statid IN VARCHAR2 DEFAULT NULL,    statown IN VARCHAR2 DEFAULT NULL,    );
```

Parameters

Table 162-46  EXPORT_PENDING_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
</tbody>
</table>
Table 162-46  (Cont.) EXPORT_PENDING_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Statistics table name to where to export the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>(Optional) Identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if other than ownname)</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

- If the parameter tablename is NULL then export applies to all tables of the specified schema.
- The default owner/schema is the user who runs the procedure.
- To run this procedure, you need to have the same privilege for gathering statistics on the tables that will be touched by this procedure.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

Examples

```
DBMS_STATS.EXPORT_PENDING_STATS(NULL, NULL, 'MY_STAT_TABLE');
```

162.7.41 EXPORT_SCHEMA_PREFS Procedure

This procedure is used to export the statistics preferences of all the tables owned by the specified owner name.

Syntax

```
DBMS_STATS.EXPORT_SCHEMA_PREFS (  
    ownname    IN  VARCHAR2,  
    stattab    IN  VARCHAR2,  
    statid     IN  VARCHAR2 DEFAULT NULL,  
    statown    IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 162-47  EXPORT_SCHEMA_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>stattab</td>
<td>Statistics table name to where to export the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>(Optional) Identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>
Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

- To run this procedure, you need to connect as owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

Examples

DBMS_STATS.EXPORT_SCHEMA_PREFS('SH', 'STAT');

162.7.42 EXPORT_SCHEMA_STATS Procedure

This procedure retrieves statistics for all objects in the schema identified by ownname and stores them in the user statistics tables identified by stattab.

Syntax

DBMS_STATS.EXPORT_SCHEMA_STATS (  
  ownname         VARCHAR2,  
  stattab         VARCHAR2,  
  statid          VARCHAR2 DEFAULT NULL,  
  statown         VARCHAR2 DEFAULT NULL,  
  stat_category   VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);  

Parameters

Table 162-48  EXPORT_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>
| stat_category | Specifies what statistics to import, accepting multiple values separated by a comma. Values supported:  
  • OBJECT_STATS - table statistics, column statistics and index statistics (Default)  
  • SYNOPTES - information to support incremental statistics  
  If 'OBJECT_STATS, SYNOPTES' is specified, table statistics, column statistics, index statistics and synopses are deleted. |

Exceptions

ORA-20000: Object does not exist or insufficient privileges
Usage Notes

- To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
- Oracle does not support export or import of statistics across databases of different character sets.

162.7.43 EXPORT_SYSTEM_STATS Procedure

This procedure retrieves system statistics and stores them in the user statistics table, identified by `stattab`.

Syntax

```sql
DBMS_STATS.EXPORT_SYSTEM_STATS (
    stattab       VARCHAR2,
    statid        VARCHAR2 DEFAULT NULL,
    statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 162-49  EXPORT_SYSTEM_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Identifier of the user statistics table that describes where the statistics will be stored</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics stored from the <code>stattab</code></td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing <code>stattab</code> (if different from current schema)</td>
</tr>
</tbody>
</table>

Security Model

To run this procedure, you must have the `GATHER_SYSTEM_STATISTICS` role.

Exceptions

- ORA-20000: Object does not exist or insufficient privileges
- ORA-20002: Bad user statistics table; may need to be upgraded
- ORA-20003: Unable to export system statistics

Usage Notes

Oracle Database does not support the export or import of statistics across databases of different character sets.
162.7.44 EXPORT_TABLE_PREFS Procedure

This procedure is used to export the statistics preferences of the specified table in the specified schema into the specified statistics table.

Syntax

DBMS_STATS.EXPORT_TABLE_PREFS (
    ownname IN VARCHAR2,
    tabname IN VARCHAR2,
    stattab IN VARCHAR2,
    statid IN VARCHAR2 DEFAULT NULL,
    statown IN VARCHAR2 DEFAULT NULL);

Parameters

Table 162-50  EXPORT_TABLE_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
<tr>
<td>stattab</td>
<td>Statistics table name where to export the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if other than ownname)</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

• To run this procedure, you need to connect as owner of the table, or have the ANALYZE ANY system privilege.
• All arguments are of type VARCHAR2 and values are enclosed in quotes.
• Oracle does not support export or import of statistics across databases of different character sets.

Examples

DBMS_STATS.EXPORT_TABLE_PREFS('SH', 'SALES', 'STAT');

162.7.45 EXPORT_TABLE_STATS Procedure

This procedure retrieves statistics for a particular table and stores them in the user statistics table. Cascade results in all index statistics associated with the specified table being exported as well.

Syntax

DBMS_STATS.EXPORT_TABLE_STATS (
    ownname VARCHAR2,
    tabname VARCHAR2,
    stattab VARCHAR2,
    statid IN VARCHAR2 DEFAULT NULL,
    statown IN VARCHAR2 DEFAULT NULL);

Chapter 162  Summary of DBMS_STATS Subprograms
Parameters

Table 162-51  EXPORT_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are exported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>cascade</td>
<td>If true, then column and index statistics for this table are also exported</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>
| stat_category  | Specifies which statistics to import or export, accepting multiple values separated by a comma. Values supported:  
|                |   - OBJECT_STATS - table statistics, column statistics and index statistics (default)  
|                |   - SYNOPSES - information to support incremental statistics  
|                | The value 'OBJECT_STATS, SYNOPSES' specifies table statistics, column statistics, index statistics, and synopses. |

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

- To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.
- Oracle does not support export or import of statistics across databases of different character sets.
162.7.46 FLUSH_DATABASE_MONITORING_INFO Procedure

This procedure saves monitoring information for all tables in the dictionary. The database immediately updates corresponding entries in the *_TAB_MODIFICATIONS, *_TAB_STATISTICS and *_IND_STATISTICS views.

Syntax

DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO;

Security Model

The ANALYZE_ANY system privilege is required to run this procedure.

Exceptions

ORA-20000: Insufficient privileges

Usage Notes

Starting in Oracle Database 12c Release 2 (12.2), you do not need to call FLUSH_DATABASE_MONITORING_INFO to view the latest information in *_TAB_STATISTICS and *_IND_STATISTICS because these views show statistics cached in the SGA and stored on disk. Because the GATHER_*_STATS procedures internally save monitoring information to disk, it is not necessary to run this procedure before gathering statistics.

See Also:

Oracle Database SQL Tuning Guide to learn how to set optimizer statistics preferences

162.7.47 GATHER_DATABASE_STATS Procedures

This procedure gathers statistics for all objects in the database.

Syntax

DBMS_STATS.GATHER_DATABASE_STATS
    (estimate_percent NUMBER DEFAULT to_estimate_percent_type
        (get_param('ESTIMATE_PERCENT')), block_sample BOOLEAN DEFAULT FALSE,
        method_opt VARCHAR2 DEFAULT get_param('METHOD_OPT'),
        degree NUMBER DEFAULT to_degree_type(get_param('DEGREE')),
        granularity VARCHAR2 DEFAULT get_param('GRANULARITY'),
        cascade BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),
        stattab VARCHAR2 DEFAULT NULL,
        statid VARCHAR2 DEFAULT NULL,
        options VARCHAR2 DEFAULT 'GATHER',
        objlist OUT ObjectTab,
        statown VARCHAR2 DEFAULT NULL,
        gather_sys VARCHAR2 DEFAULT NULL,
        no_invalidate BOOLEAN DEFAULT TRUE,
        )
get_param('NO_INVALIDATE')),

obj_filter_list ObjectTab DEFAULT NULL);

DBMS_STATS.GATHER_DATABASE_STATS (
    estimate_percent NUMBER DEFAULT to_estimate_percent_type (get_param('ESTIMATE_PERCENT')),
    block_sample     BOOLEAN DEFAULT FALSE,
    method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
    degree           NUMBER DEFAULT to_degree_type(get_param('DEGREE')),
    granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
    cascade          BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),
    stattab          VARCHAR2 DEFAULT NULL,
    statid           VARCHAR2 DEFAULT NULL,
    options          VARCHAR2 DEFAULT 'GATHER',
    statown          VARCHAR2 DEFAULT NULL,
    gather_sys       BOOLEAN DEFAULT TRUE,
    no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type (get_param('NO_INVALIDATE')),

    obj_filter_list ObjectTab DEFAULT NULL);

Parameters

Table 162-52  GATHER_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate_percent</td>
<td>Determines the percentage of rows to sample. The valid range is between 0.000001 and 100. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to enable the database to determine the appropriate sample size for optimal statistics. This is the default. You can change the default value using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Determines whether the database uses random block sampling (TRUE) or random row sampling (FALSE). The default is FALSE. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then sample values may be somewhat correlated. This parameter is only relevant when estimating statistics.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>method_opt</td>
<td>When setting preference on global, schema, database or dictionary level, only 'FOR ALL' syntax is allowed:</td>
</tr>
<tr>
<td></td>
<td>· FOR ALL [INDEXED</td>
</tr>
<tr>
<td></td>
<td>size_clause is defined as size_clause ::= SIZE {integer</td>
</tr>
<tr>
<td></td>
<td>- integer: Number of histogram buckets. Must be in the range [1,2048].</td>
</tr>
<tr>
<td></td>
<td>- REPEAT: Collects histograms only on the columns that already have histograms.</td>
</tr>
<tr>
<td></td>
<td>- AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns.</td>
</tr>
<tr>
<td></td>
<td>- SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns.</td>
</tr>
<tr>
<td></td>
<td>The default is FOR ALL COLUMNS SIZE AUTO. The value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>degree</td>
<td>Determines the degree of parallelism used for gathering statistics.</td>
</tr>
<tr>
<td></td>
<td>The default for degree is NULL. NULL means to use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Change the default using the SET_DATABASE_PREFS, SET_GLOBAL_PREFS, SET_SCHEMA_PREFS, and SET_TABLE_PREFS procedures. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement.</td>
</tr>
<tr>
<td></td>
<td>Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. The degree is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters), according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution when the size of the object does not warrant parallel execution.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| granularity | Granularity of statistics to collect (only pertinent if the table is partitioned).  
'ALL' - Gathers all (subpartition, partition, and global) statistics  
'AUTO' - Determines the granularity based on the partitioning type. This is the default value.  
'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.  
'GLOBAL' - Gathers global statistics  
'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.  
'PARTITION' - Gathers partition-level statistics  
'SUBPARTITION' - Gathers subpartition-level statistics |
| cascade | Determines whether to collect index statistics as part of gathering table statistics.  
Specifying this option is equivalent to running the GATHER_INDEX_STATS procedure on each index of the table. Use the constant DBMS_STATS.AUTO_CASCADE to enable the database to determine whether index statistics need to be collected. This is the default. You can change the default using the SET_DATABASE_PREFS, SET_GLOBAL_PREFS, SET_SCHEMA_PREFS, and SET_TABLE_PREFS procedures. |
| stattab | User statistics table identifier describing where to save the current statistics.  
The statistics table is assumed to reside in the same schema as the object being analyzed, so there must be one such table in each schema to use this option. |
| statid | Identifier (optional) to associate with these statistics within stattab. |
Table 162-52  (Cont.) GATHER_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Specifies which objects require statistics to be gathered. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>• GATHER — Gathers statistics on all objects in the database. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• GATHER AUTO — Gathers all necessary statistics automatically. The database implicitly determines which objects need new statistics and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are comp_id, no_invalidate, stattab, statid, and statown; all other parameter settings will be ignored. Also, the database returns a list of objects processed.</td>
</tr>
<tr>
<td></td>
<td>• GATHER STALE — Gathers statistics on stale objects by querying the *_TAB_MODIFICATIONS views. Also, the database returns a list of objects found to be stale.</td>
</tr>
<tr>
<td></td>
<td>• GATHER EMPTY — Gathers statistics on objects that currently have no statistics. Also, the database returns a list of objects found to have no statistics.</td>
</tr>
<tr>
<td></td>
<td>• LIST AUTO — Returns a list of objects to be processed with GATHER AUTO.</td>
</tr>
<tr>
<td></td>
<td>• LIST STALE — Returns list of stale objects as determined by looking at the *_TAB_MODIFICATIONS views.</td>
</tr>
<tr>
<td></td>
<td>• LIST EMPTY — Returns a list of objects that currently have no statistics.</td>
</tr>
<tr>
<td>objlist</td>
<td>List of objects found to be stale or empty</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>gather_sys</td>
<td>Gathers statistics on the objects owned by the SYS user.</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. The default is DBMS_STATS.AUTO_INVALIDATE, which means the database decides when to invalidate dependent cursors. If set to TRUE, then the database not invalidate dependent cursors. If set to FALSE, then the procedure invalidates dependent cursors immediately.</td>
</tr>
<tr>
<td>obj_filter_list</td>
<td>A list of object filters. When provided, GATHER_DATABASE_STATS will gather statistics only on objects which satisfy at least one object filter in the list as needed. In a single object filter, we can specify the constraints on the object attributes. The attribute values specified in the object filter are case-insensitive unless double-quoted. Wildcard is allowed in the attribute values. Suppose non-NULL values s1, s2, ... are specified for attributes a1, a2, ... in one object filter. An object o is said to satisfy this object filter if (o.a1 like s1) or (o.a2 like s2) or ... is true.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Bad input value
Usage Notes

To run this procedure, you need to have the SYSDBA role or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

If the GATHER AUTO option is used then frequency histograms will be created using a sample rather than a full scan. The following scenario shows where GATHER AUTO is used:

- A table is created like this: CREATE TABLE NEWTAB as SELECT * FROM ..... This will create statistics on NEWTAB but no histograms.
- Next, the DBA creates the histograms using GATHER AUTO on gather_table_stats.
- The FREQUENCY histograms on NEWTAB will be created using a sample rather than a full table scan.

162.7.48 GATHER_DICTIONARY_STATS Procedure

This procedure gathers statistics for dictionary schemas SYS, SYSTEM and schemas of RDBMS components.

Syntax

```
DBMS_STATS.GATHER_DICTIONARY_STATS (  
    comp_id VARCHAR2 DEFAULT NULL,  
    estimate_percent NUMBER DEFAULT to_estimate_percent_type  
        (get_param('ESTIMATE_PERCENT')),  
    block_sample BOOLEAN DEFAULT FALSE,  
    method_opt VARCHAR2 DEFAULT get_param('METHOD_OPT'),  
    degree NUMBER DEFAULT to_degree_type(get_param('DEGREE')),  
    granularity VARCHAR2 DEFAULT get_param('GRANULARITY'),  
    cascade BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),  
    stattab VARCHAR2 DEFAULT NULL,  
    statid VARCHAR2 DEFAULT NULL,  
    options VARCHAR2 DEFAULT 'GATHER AUTO',  
    objlist OUT ObjectTab,  
    statown VARCHAR2 DEFAULT NULL,  
    no_invalidate BOOLEAN DEFAULT to_no_invalidate_type  
        (get_param('NO_INVALIDATE')),  
    obj_filter_list ObjectTab DEFAULT NULL);
```

```
DBMS_STATS.GATHER_DICTIONARY_STATS (  
    comp_id VARCHAR2 DEFAULT NULL,  
    estimate_percent NUMBER DEFAULT get_param('ESTIMATE_PERCENT'),  
    block_sample BOOLEAN DEFAULT FALSE,  
    method_opt VARCHAR2 DEFAULT get_param('METHOD_OPT'),  
    degree NUMBER DEFAULT to_degree_type(get_param('DEGREE')),  
    granularity VARCHAR2 DEFAULT get_param('GRANULARITY'),  
    cascade BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),  
    stattab VARCHAR2 DEFAULT NULL,  
    statid VARCHAR2 DEFAULT NULL,  
    options VARCHAR2 DEFAULT 'GATHER AUTO',  
    statown VARCHAR2 DEFAULT NULL,  
    no_invalidate BOOLEAN DEFAULT get_param('NO_INVALIDATE'),  
    obj_filter_list ObjectTab DEFAULT NULL);
```
Parameters

**Table 162-53  GATHER_DICTIONARY_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp_id</td>
<td>Component id of the schema to analyze. NULL results in the analysis of schemas for all RDBMS components. Refer to the COMP_ID column of DBA_REGISTRY view. The procedure always gather statistics on SYS and SYSTEM schemas regardless of this argument.</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate (NULL means compute). The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Determines whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk then the sample values may be somewhat correlated. Only pertinent when performing estimate statistics.</td>
</tr>
</tbody>
</table>
| method_opt    | The method options. This parameter accepts the following values:  
  - FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]  
  - size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}  
    - integer: Number of histogram buckets. Must be in the range [1,2048].  
    - REPEAT: Collects histograms only on the columns that already have histograms.  
    - AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns.  
    - SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns.  
  The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</td>
</tr>
<tr>
<td>granularity</td>
<td>Granularity of statistics to collect (only pertinent if the table is partitioned).</td>
</tr>
<tr>
<td></td>
<td>'ALL' - Gathers all (subpartition, partition, and global) statistics</td>
</tr>
<tr>
<td></td>
<td>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</td>
</tr>
<tr>
<td></td>
<td>'GLOBAL' - Gathers global statistics</td>
</tr>
<tr>
<td></td>
<td>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</td>
</tr>
<tr>
<td></td>
<td>'PARTITION' - Gathers partition-level statistics</td>
</tr>
<tr>
<td></td>
<td>'SUBPARTITION' - gathers subpartition-level statistics</td>
</tr>
<tr>
<td>cascade</td>
<td>Gathers statistics on indexes also. Index statistics gathering will not be parallelized. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to save the current statistics</td>
</tr>
<tr>
<td>statid</td>
<td>The (optional) identifier to associate with these statistics within stattab</td>
</tr>
</tbody>
</table>
Table 162-53  (Cont.) GATHER_DICTIONARY_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Specifies which objects require statistics to be gathered. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>· GATHER AUTO — Gathers all necessary statistics automatically. This is the default. The database implicitly determines which objects need new statistics and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are comp_id, no_invalidate, stattab, statid, and statown; all other parameter settings will be ignored. Also, the database returns a list of objects processed.</td>
</tr>
<tr>
<td></td>
<td>· GATHER — Gathers statistics on all objects in the relevant schema.</td>
</tr>
<tr>
<td></td>
<td>· GATHER STALE — Gathers statistics on stale objects by querying the *_TAB_MODIFICATIONS views. Also, the database returns a list of objects found to be stale.</td>
</tr>
<tr>
<td></td>
<td>· GATHER EMPTY — Gathers statistics on objects that currently have no statistics. Also, the database returns a list of objects found to have no statistics.</td>
</tr>
<tr>
<td></td>
<td>· LIST AUTO — Returns a list of objects to be processed with GATHER AUTO.</td>
</tr>
<tr>
<td></td>
<td>· LIST STALE — Returns list of stale objects as determined by looking at the *_TAB_MODIFICATIONS views.</td>
</tr>
<tr>
<td></td>
<td>· LIST EMPTY — Returns a list of objects that currently have no statistics.</td>
</tr>
<tr>
<td>objlist</td>
<td>The list of objects found to be stale or empty.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab, if different from the current schema.</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTOINVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>obj_filter_list</td>
<td>A list of object filters. When provided, this will gather statistics only on objects which satisfy at least one object filter in the list as needed. In a single object filter, we can specify the constraints on the object attributes. The attribute values specified in the object filter are case-insensitive unless double-quoted. Wildcard is allowed in the attribute values. Suppose non-NULL values s1, s2, ... are specified for attributes a1, a2, ... in one object filter. An object o is said to satisfy this object filter if (o.a1 like s1) and (o.a2 like s2) and ... is true. See Applying an Object Filter List.</td>
</tr>
</tbody>
</table>

Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

If the GATHER_AUTO option is used then frequency histograms will be created using a sample rather than a full scan. The following scenario shows where GATHER AUTO is used:
A table is created like this:

```
CREATE TABLE NEWTAB as SELECT * FROM .....;
```

This will create statistics on NEWTAB but no histograms.

Next, the DBA creates the histograms using `GATHER AUTO` on `gather_table_stats`.

The `FREQUENCY` histograms on NEWTAB will be created using a sample rather than a full table scan.

**Exceptions**

- ORA-20000: Index does not exist or insufficient privileges
- ORA-20001: Bad input value
- ORA-20002: Bad user statistics table, may need to upgrade it

### 162.7.49 GATHER_FIXED_OBJECTS_STATS Procedure

This procedure gathers statistics for all fixed objects (dynamic performance tables).

**Syntax**

```
DBMS_STATS.GATHER_FIXED_OBJECTS_STATS (  
    stattab        VARCHAR2 DEFAULT NULL,  
    statid         VARCHAR2 DEFAULT NULL,  
    statown        VARCHAR2 DEFAULT NULL,  
    no_invalidate  BOOLEAN  DEFAULT to_no_invalidate_type (  
        get_param('NO_INVALIDATE'));
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to save the current statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier to associate with these statistics within stattab (optional)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
</tbody>
</table>

**Usage Notes**

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.
Exceptions

ORA-20000: Insufficient privileges
ORA-20001: Bad input value
ORA-20002: Bad user statistics table, may need to upgrade it

162.7.50 GATHER_INDEX_STATS Procedure

This procedure gathers index statistics. It attempts to parallelize as much of the work as possible.

Restrictions are described in the individual parameters. This operation will not parallelize with certain types of indexes, including cluster indexes, domain indexes, and bitmap join indexes. The granularity and no_invalidate arguments are not relevant to these types of indexes.

Syntax

```sql
DBMS_STATS.GATHER_INDEX_STATS (    ownname          VARCHAR2,
    indname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                         (GET_PARAM('ESTIMATE_PERCENT')),
    stattab          VARCHAR2 DEFAULT NULL,
    statid           VARCHAR2 DEFAULT NULL,
    statown          VARCHAR2 DEFAULT NULL,
    degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
    granularity      VARCHAR2 DEFAULT get_param('GRANULARITY'),
    no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type
                         (GET_PARAM('NO_INVALIDATE')),
    force            BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of index to analyze</td>
</tr>
<tr>
<td>indname</td>
<td>Name of index</td>
</tr>
<tr>
<td>partname</td>
<td>Name of partition</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate (NULL means compute). The valid range is [0.000001, 100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to save the current statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
</tbody>
</table>
### Table 162-55 (Cont.) GATHER_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statown</td>
<td>Schema containing statab (if different than ownname)</td>
</tr>
<tr>
<td>degree</td>
<td>Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</td>
</tr>
</tbody>
</table>
| granularity | Granularity of statistics to collect (only pertinent if the table is partitioned).

- 'ALL' - Gathers all (subpartition, partition, and global) statistics
- 'AUTO' - Determines the granularity based on the partitioning type. This is the default value.
- 'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.
- 'GLOBAL' - Gathers global statistics
- 'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.
- 'PARTITION' - Gathers partition-level statistics
- 'SUBPARTITION' - Gathers subpartition-level statistics. |
| no_invalidate | Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. |
| force | Gather statistics on object even if it is locked |

### Exceptions

ORA-20000: **Index does not exist or insufficient privileges**

ORA-20001: **Bad input value**
Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

162.7.51 GATHER_PROCESSING_RATE Procedure

This procedure starts the job of gathering the processing rates which end after an interval defined in minutes.

Syntax

```sql
DBMS_STATS.GATHER_PROCESSING_RATE (  gathering_mode IN VARCHAR2 DEFAULT 'START',  interval IN NUMBER DEFAULT NULL);
```

Parameters

Table 162-56 GATHER_PROCESSING_RATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gathering_mode</td>
<td>Mode: 'START' or 'END'. The mode is based on the Active Session History (ASH) data when invoked with 'START' option. It stops gathering when invoked with 'END' option. When invoked with 'START', 'interval' option can be specified optionally. If interval is not specified, its default value is set to 60 minutes.</td>
</tr>
<tr>
<td>interval</td>
<td>Time interval (number of minutes) for which the processing must be gathered</td>
</tr>
</tbody>
</table>

Usage Notes

- You require the OPTIMIZER_PROCESSING_RATE role to run this procedure.
- AUTO DOP uses processing rates to determine the optimal degree of parallelism for a SQL statement.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or illegal input value

162.7.52 GATHER_SCHEMA_STATS Procedures

This procedure gathers statistics for all objects in a schema.

Syntax

```sql
DBMS_STATS.GATHER_SCHEMA_STATS (  ownname VARCHAR2,  estimate_percent NUMBER DEFAULT to_estimate_percent_type (get_param('ESTIMATE_PERCENT')),  block_sample BOOLEAN DEFAULT FALSE,  method_opt VARCHAR2 DEFAULT get_param('METHOD_OPT'),
```


Parameters

**Table 162-57**  GATHER_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema to analyze (NULL means current schema)</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Determines the percentage of rows to sample. The valid range is between 0.000001 and 100. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to enable the database to determine the appropriate sample size for optimal statistics. This is the default. You can change the default value using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>method_opt</td>
<td>Accepts:</td>
</tr>
<tr>
<td></td>
<td>- FOR ALL [INDEXED</td>
</tr>
<tr>
<td></td>
<td>size_clause is defined as size_clause := SIZE {integer</td>
</tr>
<tr>
<td></td>
<td>REPEAT</td>
</tr>
<tr>
<td></td>
<td>- integer: Number of histogram buckets. Must be in the</td>
</tr>
<tr>
<td></td>
<td>range [1,2048].</td>
</tr>
<tr>
<td></td>
<td>- REPEAT: Collects histograms only on the columns that</td>
</tr>
<tr>
<td></td>
<td>already have histograms</td>
</tr>
<tr>
<td></td>
<td>- AUTO: Oracle determines the columns on which to collect</td>
</tr>
<tr>
<td></td>
<td>histograms based on data distribution and the workload of the</td>
</tr>
<tr>
<td></td>
<td>columns.</td>
</tr>
<tr>
<td></td>
<td>- SKEWONLY: Oracle determines the columns on which to</td>
</tr>
<tr>
<td></td>
<td>collect histograms based on the data distribution of the</td>
</tr>
<tr>
<td></td>
<td>columns.</td>
</tr>
<tr>
<td></td>
<td>The default is FOR ALL COLUMNS SIZE AUTO. The default value</td>
</tr>
<tr>
<td></td>
<td>can be changed using the SET_DATABASE_PREFS Procedure,</td>
</tr>
<tr>
<td></td>
<td>SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>degree</td>
<td>Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</td>
</tr>
</tbody>
</table>
### Table 162-57  (Cont.) GATHER_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>granularity</td>
<td>Granularity of statistics to collect (only pertinent if the table is partitioned).&lt;br&gt;  'ALL' - Gathers all (subpartition, partition, and global) statistics.&lt;br&gt;  'AUTO' - Determines the granularity based on the partitioning type. This is the default value.&lt;br&gt;  'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.&lt;br&gt;  'GLOBAL' - Gathers global statistics&lt;br&gt;  'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.&lt;br&gt;  'PARTITION' - Gathers partition-level statistics&lt;br&gt;  'SUBPARTITION' - Gathers subpartition-level statistics.</td>
</tr>
<tr>
<td>cascade</td>
<td>Gather statistics on the indexes as well. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to save the current statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
</tbody>
</table>
### Table 162-57  (Cont.) GATHER_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Specifies which objects require statistics to be gathered. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>- <strong>GATHER</strong> — Gathers statistics on all objects in the schema. This is the default.</td>
</tr>
<tr>
<td></td>
<td>- <strong>GATHER AUTO</strong> — Gathers all necessary statistics automatically. The database implicitly determines which objects need new statistics and determines how to gather those statistics. When <strong>GATHER AUTO</strong> is specified, the only additional valid parameters are comp_id, no_invalidate, stattab, statid, and statown; all other parameter settings will be ignored. Also, the database returns a list of objects processed.</td>
</tr>
<tr>
<td></td>
<td>- <strong>GATHER STALE</strong> — Gathers statistics on stale objects by querying the *_TAB_MODIFICATIONS views. Also, the database returns a list of objects found to be stale.</td>
</tr>
<tr>
<td></td>
<td>- <strong>GATHER EMPTY</strong> — Gathers statistics on objects that currently have no statistics. Also, the database returns a list of objects found to have no statistics.</td>
</tr>
<tr>
<td></td>
<td>- <strong>LIST AUTO</strong> — Returns a list of objects to be processed with <strong>GATHER AUTO</strong>.</td>
</tr>
<tr>
<td></td>
<td>- <strong>LIST STALE</strong> — Returns list of stale objects as determined by looking at the *_TAB_MODIFICATIONS views.</td>
</tr>
<tr>
<td></td>
<td>- <strong>LIST EMPTY</strong> — Returns a list of objects that currently have no statistics.</td>
</tr>
<tr>
<td>objlist</td>
<td>List of objects found to be stale or empty</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use <strong>DBMS_STATS.AUTO_INVALIDATE</strong> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <strong>SET_DATABASE_PREFS</strong>, <strong>SET_GLOBAL_PREFS</strong> Procedure, <strong>SET_SCHEMA_PREFS</strong> Procedure and <strong>SET_TABLE_PREFS</strong> Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>Gather statistics on objects even if they are locked</td>
</tr>
<tr>
<td>obj_filter_list</td>
<td>A list of object filters. When provided, <strong>GATHER_SCHEMA_STATS</strong> will gather statistics only on objects which satisfy at least one object filter in the list as needed. In a single object filter, we can specify the constraints on the object attributes. The attribute values specified in the object filter are case-insensitive unless double-quoted. Wildcard is allowed in the attribute values. Suppose non-NULL values s1, s2, ... are specified for attributes a1, a2, ... in one object filter. An object o is said to satisfy this object filter if (o.a1 like s1) and (o.a2 like s2) and ... is true.</td>
</tr>
</tbody>
</table>
Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you must be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

If the GATHER AUTO option is used then frequency histograms will be created using a sample rather than a full scan. The following scenario shows where GATHER AUTO is used:

- A table is created like this: CREATE TABLE NEWTAB as SELECT * FROM ....
  This will create statistics on NEWTAB but no histograms.
- Next, the DBA creates the histograms using GATHER AUTO on gather_table_stats.
- The FREQUENCY histograms on NEWTAB will be created using a sample rather than a full table scan.

Exceptions

ORA-20000: Schema does not exist or insufficient privileges
ORA-20001: Bad input value

Examples

Applying an Object Filter List

The following example specifies that the tables SH.SALES and SH.COSTS, if stale, will have statistics gathered upon them.

DECLARE
  filter_lst  DBMS_STATS.OBJECTTAB := DBMS_STATS.OBJECTTAB();
BEGIN
  filter_lst.extend(2);
  filter_lst(1).ownname := 'SH';
  filter_lst(1).objname := 'SALES';
  filter_lst(2).ownname := 'SH';
  filter_lst(2).objname := 'COSTS';
  DBMS_STATS.GATHER_SCHEMA_STATS(ownname=>'SH',obj_filter_list=>filter_lst);
END;

162.7.53 GATHER_SYSTEM_STATS Procedure

This procedure gathers system statistics.

Syntax

```
DBMS_STATS.GATHER_SYSTEM_STATS ( 
  gathering_mode VARCHAR2 DEFAULT 'NOWORKLOAD',
  interval INTEGER DEFAULT NULL,
  stattab VARCHAR2 DEFAULT NULL,
  statid VARCHAR2 DEFAULT NULL,
  statown VARCHAR2 DEFAULT NULL);
```
Parameters

### Table 162-58  GATHER_SYSTEM_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| gathering_mode | Specifies the mode in which the database gathers system statistics. Possible values are:<br>  
  - NOWORKLOAD<br>  
    The database captures performance characteristics of the I/O system. Gathering may take a few minutes and depends on the size of the database. During this period, the database estimates the average read seek time and transfer speed for the I/O system. This mode is suitable for all workloads.  
    To fine tune system statistics for the workload, use START and STOP or the INTERVAL option. If you gather both NOWORKLOAD and workload-specific statistics (statistics collected using INTERVAL or START and STOP), the optimizer uses the workload statistics. Collected components include cpuspeednw, ioseektim, and iotfrspeed.<br>  
  - INTERVAL<br>  
    The database captures system activity during a specified interval in minutes. This parameter works in combination with the interval parameter. The database creates or updates system statistics in the dictionary or stattab. You can use GATHER_SYSTEM_STATS (gathering_mode=>'STOP') to stop gathering earlier than scheduled. Collected components include maxthr, slavethr, cpuspeed, sreadtim, mreadtim, and mbrc.<br>  
  - START | STOP<br>  
    The database captures system activity during specified start and stop times and refreshes the dictionary or stattab with statistics for the elapsed period. The database ignores the INTERVAL value. Collected components include maxthr, slavethr, cpuspeed, sreadtim, mreadtim, and mbrc.<br>  
  - EXADATA<br>  
    In this mode, gathered system statistics take into account the unique capabilities of Oracle Exadata, such as large I/O size and high I/O throughput. The database sets multiblock read count and I/O throughput statistics along with CPU speed. |
| interval     | Specifies the number of minutes in which to gather system statistics. This parameter applies only when gathering_mode='INTERVAL'. |
| stattab      | Specifies the table in which the database stores the statistics. |
| statid       | Specifies an optional identifier associated with the statistics saved in stattab. |
| statown      | Specifies the schema containing stattab, if different from the current schema. |

### Exceptions

- **ORA-20000**: Object does not exist or insufficient privileges
- **ORA-20001**: Invalid input value
- **ORA-20002**: Bad user statistics table; may need to be upgraded
ORA-20003: Unable to gather system statistics

ORA-20004: Error in the INTERVAL mode: system parameter job_queue_processes must be >0

Usage Notes

To run this procedure, you must have the GATHER_SYSTEM_STATISTICS role.

Examples

Assume that you want to perform database application processing OLTP transactions during the day and run reports at night.

To collect daytime system statistics, gather statistics for 720 minutes. Store the statistics in the MYSTATS table.

BEGIN
  DBMS_STATS.GATHER_SYSTEM_STATS (interval => 720, stattab => 'mystats', statid => 'OLTP');
END;

To collect nighttime system statistics, gather statistics for 720 minutes. Store the statistics in the MYSTATS table.

BEGIN
  DBMS_STATS.GATHER_SYSTEM_STATS (interval => 720, stattab => 'mystats', statid => 'OLAP');
END;

Update the dictionary with the gathered statistics.

VARIABLE jobno number;
BEGIN
  DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS ('mystats','OLTP');'
    sysdate, 'sysdate + 1');
  COMMIT;
END;

BEGIN
  DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS ('mystats','OLAP');'
    sysdate + 0.5, 'sysdate + 1');
  COMMIT;
END;
162.7.54 GATHER_TABLE_STATS Procedure

This procedure gathers table, column, and index statistics. It attempts to parallelize as much work as possible, but there are some restrictions, which are described in the individual parameters.

Syntax

```
DBMS_STATS.GATHER_TABLE_STATS (  
  ownname           VARCHAR2,  
  tabname           VARCHAR2,  
  partname          VARCHAR2 DEFAULT NULL,  
  estimate_percent  NUMBER   DEFAULT to_estimate_percent_type  
                     (get_param('ESTIMATE_PERCENT')),  
  block_sample      BOOLEAN  DEFAULT FALSE,  
  method_opt        VARCHAR2 DEFAULT get_param('METHOD_OPT'),  
  degree            NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),  
  granularity       VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),  
  cascade           BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),  
  stattab           VARCHAR2 DEFAULT NULL,  
  statid            VARCHAR2 DEFAULT NULL,  
  statown           VARCHAR2 DEFAULT NULL,  
  no_invalidate     BOOLEAN  DEFAULT to_no_invalidate_type  
                     (get_param('NO_INVALIDATE')),  
  stattype          VARCHAR2 DEFAULT 'DATA',  
  force             BOOLEAN  DEFAULT FALSE,  
  context           DBMS_STATS.CCONTEXT DEFAULT NULL, -- non operative  
  options           VARCHAR2 DEFAULT get_param('OPTIONS'));
```

Parameters

Table 162-59  GATHER_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema containing the table.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the partition.</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Determines the percentage of rows to sample. The valid range is between 0.000001 and 100. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to enable the database to determine the appropriate sample size for optimal statistics. This is the default. You can change the default value using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Determines whether the database uses random block sampling (TRUE) or random row sampling (FALSE). The default is FALSE. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then sample values may be somewhat correlated. This parameter is only relevant when estimating statistics.</td>
</tr>
</tbody>
</table>

You can change the default value using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.
Table 162-59  (Cont.) GATHER_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| method_opt     | METHOD_OPT - When setting preference on global, schema, database or dictionary level, only 'FOR ALL' syntax is allowed. Other than that, method_opt accepts either of the following options, or both in combination:  
  - FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]  
  - FOR COLUMNS [column_clause] [size_clause]  
  size_clause is defined as: size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}  
  column_clause is defined as: column_clause := column_name | extension name | extension  
  - integer: Number of histogram buckets. Must be in the range [1,2048].  
  - REPEAT: Collects histograms only on the columns that already have histograms  
  - AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns.  
  - SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns.  
  - column_name: Name of a column  
  - extension: can be either a column group in the format of (column_name,Columne_name [, ...]) or an expression  
  The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. |
| degree         | Determines the degree of parallelism used for gathering statistics. The default for degree is NULL. NULL means to use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Change the default using the SET_DATABASE_PREFS, SET_GLOBAL_PREFS, SET_SCHEMA_PREFS, and SET_TABLE_PREFS procedures. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement.  
  Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO DEGREE value determines the degree of parallelism automatically. The degree is between 1 (serial execution) and DEFAULT DEGREE (the system default value based on number of CPUs and initialization parameters), according to the size of the object. When using DEGREE=>NULL, DEGREE=>n, or DEGREE=>DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution when the size of the object does not warrant parallel execution. |
Table 162-59  (Cont.) GATHER_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>granularity</td>
<td>Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - Gathers all (subpartition, partition, and global) statistics 'APPROX_GLOBAL AND PARTITION' - similar to 'GLOBAL AND PARTITION' but in this case the global statistics are aggregated from partition level statistics. This option will aggregate all statistics except the number of distinct values for columns and number of distinct keys of indexes. The existing histograms of the columns at the table level are also aggregated. The aggregation will use only partitions with statistics, so to get accurate global statistics, users should make sure to have statistics for all partitions. Global statistics are gathered if partname is NULL or if the aggregation cannot be performed (for example, if statistics for one of the partitions is missing). 'AUTO' - Determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - Gathers global statistics 'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION' - Gathers partition-level statistics 'SUBPARTITION' - Gathers subpartition-level statistics.</td>
</tr>
<tr>
<td>cascade</td>
<td>Determines whether to collect index statistics as part of gathering table statistics. Specifying this option is equivalent to running the GATHER_INDEX_STATS procedure on each index of the table. Use the constant DBMS_STATS.AUTO_CASCADE to enable the database to determine whether index statistics need to be collected. This is the default. You can change the default using the SET_DATABASE_PREFS, SET_GLOBAL_PREFS, SET_SCHEMA_PREFS, and SET_TABLE_PREFS procedures.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to save the current statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. The default is DBMS_STATS.AUTO_INVALIDATE, which means the database decides when to invalidate dependent cursors. If set to TRUE, then the database not invalidate dependent cursors. If set to FALSE, then the procedure invalidates dependent cursors immediately.</td>
</tr>
<tr>
<td>stattype</td>
<td>Statistics type. The only value allowed is DATA.</td>
</tr>
</tbody>
</table>
Table 162-59  (Cont.) GATHER_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>Gather statistics of table even if it is locked</td>
</tr>
<tr>
<td>context</td>
<td>Not used.</td>
</tr>
<tr>
<td>options</td>
<td>Determines the options parameter used in the GATHER_TABLE_STATS procedure. The preference takes the following values:</td>
</tr>
<tr>
<td></td>
<td>· GATHER — Gathers statistics for all objects in the table. This is the default.</td>
</tr>
<tr>
<td></td>
<td>· GATHER AUTO — Gathers all necessary statistics automatically. Oracle recommends setting GATHER AUTO on tables that undergo bulk load operations that gather statistics. This option is only applicable to tables that do not have INCREMENTAL enabled. Running GATHER_TABLE_STATS procedure on these tables with the GATHER AUTO option skips regathering the already fresh statistics.</td>
</tr>
</tbody>
</table>

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Index statistics collection can be paralleled except for cluster, domain and join indexes.

If the GATHER AUTO option is used then frequency histograms will be created using a sample rather than a full scan. The following scenario shows where GATHER AUTO is used:

• A table is created like this: CREATE TABLE NEWTAB as SELECT * FROM ..... This will create statistics on NEWTAB but no histograms.
• Next, the DBA creates the histograms using GATHER AUTO on gather_table_stats.
• The FREQUENCY histograms on NEWTAB will be created using a sample rather than a full table scan.

Exceptions

ORA-20000: Table does not exist or insufficient privileges
ORA-20001: Bad input value

Examples

An extension can be either a column group (see Example 1) or an expression (see Example 2).

Example 1

```
DBMS_STATS.GATHER_TABLE_STATS(
    'SH', 'SALES', method_opt => 'FOR COLUMNS (empno, deptno)');
```
Example 2

```
DBMS_STATS.GATHER_TABLE_STATS(
  'SH', 'SALES', method_opt => 'FOR COLUMNS (sal+comm)');
```

**162.7.55 GENERATE_STATS Procedure**

This *deprecated procedure* generates object statistics from previously collected statistics of related objects. The currently supported objects are b-tree and bitmap indexes.

---

**Note:**

This subprogram has been deprecated and replaced by improved technology. It is maintained only for purposes of backward compatibility. As an alternative, use the `GATHER_INDEX_STATS` procedure. See "GATHER_INDEX_STATS Procedure".

---

**Syntax**

```
DBMS_STATS.GENERATE_STATS (  
  ownname    VARCHAR2,  
  objname    VARCHAR2,  
  organized  NUMBER DEFAULT 7,  
  force      BOOLEAN default FALSE);
```

**Parameters**

**Table 162-60  GENERATE_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of object</td>
</tr>
<tr>
<td>objname</td>
<td>Name of object</td>
</tr>
<tr>
<td>organized</td>
<td>Amount of ordering associated between the index and its underlying table. A heavily organized index would have consecutive index keys referring to consecutive rows on disk for the table (the same block). A heavily disorganized index would have consecutive keys referencing different table blocks on disk. This parameter is only used for b-tree indexes. The number can be in the range of 0-10, with 0 representing a completely organized index and 10 a completely disorganized one.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, generates statistics for the target object even if it is locked</td>
</tr>
</tbody>
</table>

**Usage Notes**

To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the SYSDBA privilege.

For fully populated schemas, the gather procedures should be used instead when more accurate statistics are desired.
Exceptions

ORA-20000: Unsupported object type of object does not exist
ORA-20001: Invalid option or invalid statistics

162.7.56 GET_ADVISOR_OPR_FILTER Procedure

This procedure creates an operation filter for an Optimizer Statistics Advisor task.

Syntax

```sql
DBMS_STATS.GET_ADVISOR_OPR_FILTER
(    opr_id        IN             NUMBER,
    opr_filter    IN OUT NOCOPY  StatsAdvOpr);
```

Parameters

Table 162-61  GET_ADVISOR_OPR_FILTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opr_id</td>
<td>The ID of the statistics operation stored in the DBA_OPTSTAT_OPERATIONS view.</td>
</tr>
<tr>
<td>opr_filter</td>
<td>The Optimizer Statistics Advisor filter that is generated based on the specified statistics operation.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- This subprogram executes using invoker’s rights.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
- ORA-20012: Optimizer Statistics Advisor errors

Usage Notes

You can specify the filter using either the operation ID or the filter ID, but not both at the same time.
162.7.57 GET_ADVISOR_RECS Function

This function generates a recommendation report for the specified item.

Syntax

```
DBMS_STATS.GET_ADVISOR_RECS(
    ownname IN VARCHAR2,
    tabname IN VARCHAR2,
    rec IN VARCHAR2,
    type IN VARCHAR2 := 'TEXT')
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>The owner of the table.</td>
</tr>
<tr>
<td>tabname</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>rec</td>
<td>The Optimizer Statistics Advisor recommendation.</td>
</tr>
<tr>
<td></td>
<td>- INCREMENTAL</td>
</tr>
<tr>
<td></td>
<td>When only a small number of range partitions are modified, this option improves the performance of statistics gathering dramatically. However, it requires additional space to store synopses for maintaining incremental statistics. The report analyzes this trade-off.</td>
</tr>
<tr>
<td></td>
<td>- CONCURRENT</td>
</tr>
<tr>
<td></td>
<td>The report recommends either setting the CONCURRENT preference, or specifying AUTO_DEGREE for individual tables. If the system resources and usage satisfies the conditions, the advisor always recommends setting CONCURRENT first. The advisor only recommends AUTO_DEGREE when statistics gathering on an individual table take a long time and the CONCURRENT preference is already set.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the report: TEXT, HTML, or XML.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must have the privileges to gather statistics for the objects for which recommendations are generated.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.
Usage Notes

The advisor does not make recommendations for manual statistics gathering. The database only make recommendations for automatic statistics gathering jobs, with the main goal of finishing the job within the maintenance window. As long as the automatic job finishes, the database does not make further recommendations.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
- ORA-20012: Optimizer Statistics Advisor errors

162.7.58 GET_COLUMN_STATS Procedures

These procedures gets all column-related information. In the form of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

Syntax

```
DBMS_STATS.GET_COLUMN_STATS (  
ownname VARCHAR2,  
tabname VARCHAR2,  
colname VARCHAR2,  
partname VARCHAR2 DEFAULT NULL,  
stattab VARCHAR2 DEFAULT NULL,  
statid VARCHAR2 DEFAULT NULL,  
distcnt OUT NUMBER,  
density OUT NUMBER,  
nulcnt OUT NUMBER,  
srec OUT StatRec,  
avgclen OUT NUMBER,  
statown VARCHAR2 DEFAULT NULL);
```

Use the following for user-defined statistics:

```
DBMS_STATS.GET_COLUMN_STATS (  
ownname VARCHAR2,  
tabname VARCHAR2,  
colname VARCHAR2,  
partname VARCHAR2 DEFAULT NULL,  
stattab VARCHAR2 DEFAULT NULL,  
statid VARCHAR2 DEFAULT NULL,  
ext_stats OUT RAW,  
stattypown OUT VARCHAR2 DEFAULT NULL,  
stattypname OUT VARCHAR2 DEFAULT NULL,  
statown VARCHAR2 DEFAULT NULL);
```
Parameters

Table 162-63   GET_COLUMN_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column or extension</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition from which to get the statistics. If the table</td>
</tr>
<tr>
<td></td>
<td>is partitioned and if partname is NULL, statistics are retrieved from the</td>
</tr>
<tr>
<td></td>
<td>global table level.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the</td>
</tr>
<tr>
<td></td>
<td>statistics. If stattab is NULL, statistics are retrieved directly from the</td>
</tr>
<tr>
<td></td>
<td>dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td></td>
<td>(Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>ext_stats</td>
<td>The user-defined statistics</td>
</tr>
<tr>
<td>stattypown</td>
<td>Schema of the statistics type</td>
</tr>
<tr>
<td>stattypname</td>
<td>Name of the statistics type</td>
</tr>
<tr>
<td>distcnt</td>
<td>Number of distinct values</td>
</tr>
<tr>
<td>density</td>
<td>Column density</td>
</tr>
<tr>
<td>nullcnt</td>
<td>Number of NULLs</td>
</tr>
<tr>
<td>srec</td>
<td>Structure holding internal representation of column minimum, maximum, and</td>
</tr>
<tr>
<td></td>
<td>histogram values.</td>
</tr>
<tr>
<td>avgclen</td>
<td>Average length of the column (in bytes)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

Usage Notes

Before invoking this procedure, ensure that the table exists.

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.
162.7.59 GET_INDEX_STATS Procedures

This overloaded procedure gets all index-related statistics. In the form of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

Syntax

```sql
DBMS_STATS.GET_INDEX_STATS ( 
    ownname VARCHAR2, 
    indname VARCHAR2, 
    partname VARCHAR2 DEFAULT NULL, 
    stattab VARCHAR2 DEFAULT NULL, 
    statid VARCHAR2 DEFAULT NULL, 
    numrows OUT NUMBER, 
    numblks OUT NUMBER, 
    numdist OUT NUMBER, 
    avglblk OUT NUMBER, 
    avgdblk OUT NUMBER, 
    clstfct OUT NUMBER, 
    indlevel OUT NUMBER, 
    statown VARCHAR2 DEFAULT NULL);

DBMS_STATS.GET_INDEX_STATS ( 
    ownname VARCHAR2, 
    indname VARCHAR2, 
    partname VARCHAR2 DEFAULT NULL, 
    stattab VARCHAR2 DEFAULT NULL, 
    statid VARCHAR2 DEFAULT NULL, 
    numrows OUT NUMBER, 
    numblks OUT NUMBER, 
    numdist OUT NUMBER, 
    avglblk OUT NUMBER, 
    avgdblk OUT NUMBER, 
    clstfct OUT NUMBER, 
    indlevel OUT NUMBER, 
    statown VARCHAR2 DEFAULT NULL, 
    guessq OUT NUMBER);

Use the following form of the procedure for user-defined statistics:

```dbms_stats.get_index_stats ( 
    ownname VARCHAR2, 
    indname VARCHAR2, 
    partname VARCHAR2 DEFAULT NULL, 
    stattab VARCHAR2 DEFAULT NULL, 
    statid VARCHAR2 DEFAULT NULL, 
    ext_stats OUT RAW, 
    stattypown OUT VARCHAR2 DEFAULT NULL, 
    stattypname OUT VARCHAR2 DEFAULT NULL, 
    statown VARCHAR2 DEFAULT NULL);```
Parameters

Table 162-64  GET_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the index partition for which to get the statistics. If the index is partitioned and if partname is NULL, then the statistics are retrieved for the global index level.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>ext_stats</td>
<td>User-defined statistics</td>
</tr>
<tr>
<td>stattypown</td>
<td>Schema of the statistics type</td>
</tr>
<tr>
<td>stattypname</td>
<td>Name of the statistics type</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the index (partition)</td>
</tr>
<tr>
<td>numlblks</td>
<td>Number of leaf blocks in the index (partition)</td>
</tr>
<tr>
<td>numdist</td>
<td>Number of distinct keys in the index (partition)</td>
</tr>
<tr>
<td>avgblbk</td>
<td>Average integral number of leaf blocks in which each distinct key appears for this index (partition)</td>
</tr>
<tr>
<td>avgdblk</td>
<td>Average integral number of data blocks in the table pointed to by a distinct key for this index (partition)</td>
</tr>
<tr>
<td>clstfct</td>
<td>Clustering factor for the index (partition)</td>
</tr>
<tr>
<td>indlevel</td>
<td>Height of the index (partition)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>guessq</td>
<td>Guess quality for the index (partition)</td>
</tr>
</tbody>
</table>

Security Model

Before invoking this procedure, ensure that the table exists. To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

Usage Notes

- The optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The database calculates the total cost of the operation by combining the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
• The database maintains `cachedblk` and `cachehit` at all times. However, the database uses the corresponding caching statistics for optimization as part of the table and index statistics only when the user calls the `DBMS_STATS.GATHER_[TABLE/INDEX/SHEMA/DATABASE]_STATS` procedure for automatic mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. To prevent the user from utilizing inaccurate and unreliable data, the optimizer computes a “confidence factor” for each `cachehit` and a `cachedblk` for each object. If the confidence factor for the value meets confidence criteria, then the database uses this value; otherwise, the database uses defaults.

• The automatic maintenance algorithm for object caching statistics assumes that only one major database workload exists. The algorithm adjusts statistics to this workload, ignoring other "minor" workloads. If this assumption is false, then you must use manual mode for maintaining object caching statistics.

• The object caching statistics maintenance algorithm for automatic mode prevents you from using statistics in the following situations:
  – When not enough data has been analyzed, such as when an object has been recently created
  – When the system does not have one major workload resulting in averages not corresponding to real values

**See Also:**

*Oracle Database SQL Tuning Guide* to learn how to manage optimizer statistics

### 162.7.60 GET_PARAM Function

This function returns the default value of parameters of `DBMS_STATS` procedures.

**Note:**

This subprogram has been replaced by improved technology and is maintained only for purposes of backward compatibility. In this case, use the `GET_PREFS` Function.

See also [DBMS_STATS Deprecated Subprograms](#).

**Syntax**

```
DBMS_STATS.GET_PARAM (  
  pname   IN   VARCHAR2) 
RETURN VARCHAR2;
```
Parameters

Table 162-65  GET_PARAM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>Parameter name</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20001: Invalid input values

162.7.61 GET_PREFS Function

This function returns the default value of the specified preference.

Syntax

```
DBMS_STATS.GET_PREFS (  
   pname     IN   VARCHAR2,  
   ownname   IN   VARCHAR2 DEFAULT NULL,  
   tabname   IN   VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

Table 162-66  GET_PREFS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| pname     | Preference name. The possible values are:  
  - APPROXIMATE_NDV_ALGORITHM  
  - AUTO_STAT_EXTENSIONS  
  - CASCADE  
  - CONCURRENT  
  - DEGREE  
  - ESTIMATE_PERCENT  
  - GLOBAL_TEMP_TABLE_STATS  
  - GRANULARITY  
  - INCREMENTAL  
  - INCREMENTAL_StALENESS  
  - INCREMENTAL_LEVEL  
  - METHOD_OPT  
  - NO_INVALIDATE  
  - OPTIONS  
  - PREFERENCE_OVERRIDES_PARAMETER  
  - PUBLISH  
  - STALE_PERCENT  
  - STAT_CATEGORY  
  - TABLE_CACHED_BLOCKS  
  - WAIT_TIME_TO_UPDATE_STATUS  
| ownname    | Owner name |
Table 162-66  (Cont.) GET_PREFS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
</tbody>
</table>

Table 162-67  Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPROXIMATE_NDV_ALGORITHM</td>
<td>Specifies the synopsis generation algorithm. A synopsis is special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. Consider a synopsis as an internal management structure that samples distinct values. You can specify the following preferences:</td>
</tr>
<tr>
<td></td>
<td>• REPEAT OR HYPERLOGLOG</td>
</tr>
<tr>
<td></td>
<td>This is the default. If INCREMENTAL is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format. This approach is attractive when existing performance is acceptable, and you do not want to incur the performance cost of reformatting legacy content.</td>
</tr>
<tr>
<td></td>
<td>• ADAPTIVE_SAMPLING</td>
</tr>
<tr>
<td></td>
<td>The database uses the adaptive sampling algorithm for all synopses. This is the most conservative option.</td>
</tr>
<tr>
<td></td>
<td>• HYPERLOGLOG</td>
</tr>
<tr>
<td></td>
<td>The database uses the HyperLogLog algorithm for all new and stale synopses. In contrast to dynamic sampling, the HyperLogLog algorithm uses a randomization technique. The advantages of HyperLogLog over adaptive sampling are:</td>
</tr>
<tr>
<td></td>
<td>- The accuracy of the new algorithm is similar to the original algorithm.</td>
</tr>
<tr>
<td></td>
<td>- The memory required is significantly lower, which typically leads to huge reductions in synopsis size.</td>
</tr>
</tbody>
</table>
### Table 162-67 (Cont.) Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_STAT_EXTENSIONS</td>
<td>Controls the automatic creation of extensions when database statistics are gathered.</td>
</tr>
<tr>
<td></td>
<td>You can set the following values:</td>
</tr>
<tr>
<td></td>
<td>• <strong>ON</strong> — When applicable, a SQL plan directive can trigger the creation of column group</td>
</tr>
<tr>
<td></td>
<td>statistics based on usage of columns in the predicates in the workload.</td>
</tr>
<tr>
<td></td>
<td>• <strong>OFF</strong> — The database does not create column group statistics automatically. The database</td>
</tr>
<tr>
<td></td>
<td>creates them only when the <code>CREATE_EXTENDED_STATS</code> function is executed, or when extended</td>
</tr>
<tr>
<td></td>
<td>statistics are specified explicitly in the <code>METHOD_OPT</code> clause of <code>DBMS_STATS</code>. This is the</td>
</tr>
<tr>
<td></td>
<td>default.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Determines whether index statistics are collected as part of gathering table statistics.</td>
</tr>
<tr>
<td>CONCURRENT</td>
<td>Determines whether statistics are gathered concurrently on multiple objects, or serially, one</td>
</tr>
<tr>
<td></td>
<td>object at a time. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• <strong>MANUAL</strong> — Concurrency is enabled only for manual statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>• <strong>AUTOMATIC</strong> — Concurrency is enabled only for the automatic statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>• <strong>ALL</strong> — Concurrency is enabled for both manual and automatic statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>• <strong>OFF</strong> — Concurrency is disabled for both manual and automatic statistics.</td>
</tr>
<tr>
<td>DEGREE</td>
<td>Determines degree of parallelism used for gathering statistics.</td>
</tr>
<tr>
<td>ESTIMATE_PERCENT</td>
<td>Determines the percentage of rows to sample.</td>
</tr>
<tr>
<td></td>
<td>The valid range is between 0.000001 and 100. Use the constant <code>DBMS_STATS.AUTO_SAMPLE_SIZE</code> to</td>
</tr>
<tr>
<td></td>
<td>enable the database to determine the appropriate sample size for optimal statistics. This is the</td>
</tr>
<tr>
<td></td>
<td>default.</td>
</tr>
<tr>
<td>GLOBAL_TEMP_TABLE_STATS</td>
<td>Controls whether the statistics gathered for a global temporary table should be stored as shared</td>
</tr>
<tr>
<td></td>
<td>statistics or session statistics. This preference takes two values:</td>
</tr>
<tr>
<td></td>
<td>• <strong>SHARED</strong> — All sessions see the same set of statistics</td>
</tr>
<tr>
<td></td>
<td>• <strong>SESSION</strong> — Statistics gathered by the <code>GATHER_TABLE_STATS</code> procedure on a global temporary</td>
</tr>
<tr>
<td></td>
<td>table are session-specific. Thus, the database only uses them for queries issued in the same</td>
</tr>
<tr>
<td></td>
<td>session as the statistics gathering process. The database deletes session-specific statistics</td>
</tr>
<tr>
<td></td>
<td>when a session terminates.</td>
</tr>
</tbody>
</table>
### Table 162-67  (Cont.) Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| GRANULARITY     | Determines the granularity of statistics to collect. This preference is only relevant for partitioned tables.  
The following values are valid:  
  • ALL — Gathers all statistics: subpartition, partition, and global.  
  • AUTO — Determines the granularity based on the partitioning type. This is the default value.  
  • DEFAULT — Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. Use GLOBAL AND PARTITION for this functionality.  
  • GLOBAL — Gathers global statistics.  
  • GLOBAL AND PARTITION — Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.  
  • PARTITION — Gathers partition-level statistics.  
  • SUBPARTITION — Gathers subpartition-level statistics. |
| INCREMENTAL     | Determines whether the global statistics for a partitioned table are maintained without performing a full table scan.  
When a table is partitioned, an application typically loads data into a new partition. As new partitions are added and data is loaded, global table statistics must be kept up to date. If the following conditions are met, then the database updates the global table statistics by scanning only the changed partitions instead of the entire table:  
  • The INCREMENTAL value for the partitioned table is set to TRUE.  
  • The PUBLISH value for the partitioned table is set to TRUE.  
  • The user specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table.  
If the INCREMENTAL value for the partitioned table was set to FALSE (default value), then the database uses a full table scan to maintain the global statistics. This technique is a much more resource-intensive and time-consuming operation for large tables. |
### Table 162-67  (Cont.) Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>Controls which synopses to collect when INCREMENTAL preference is set to TRUE. It takes the following values:</td>
</tr>
<tr>
<td></td>
<td>• PARTITION — Gathers partition-level synopses. This is the default value. If PARTITION is set on a nonpartitioned table, then the database does not gather synopses.</td>
</tr>
<tr>
<td></td>
<td>• TABLE — Gathers table-level synopses. Specify this value when you want to exchange this table with a partition. Before the exchange, you can run GATHER_TABLE_STATS on this table with INCREMENTAL set to TRUE and INCREMENTAL_LEVEL to TABLE. The result is that the database gathers table-level synopses on this table. After the exchange, the partition has synopses that come from the table-level synopses of the table before the exchange. You can only use preference value in the SET_TABLE_PREFS procedure: this value is not allowed in the other SET_*_PREFS procedures.</td>
</tr>
</tbody>
</table>
Table 162-67  (Cont.) Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_STALENESS</td>
<td>Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT,USE_LOCKED_STATS,ALLOW_MIXED_FORMAT'. The parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>•  USE_STALE_PERCENT—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference. For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics. Assume a different case in which STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.</td>
</tr>
<tr>
<td></td>
<td>•  USE_LOCKED_STATS—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes. For example, assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS,USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.</td>
</tr>
<tr>
<td></td>
<td>•  ALLOW_MIXED_FORMAT—Adaptive sampling synopses and HyperLogLog synopses are permitted to coexist.</td>
</tr>
<tr>
<td></td>
<td>•  NULL—A partition or subpartition is considered stale when it has any DML changes. For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics.</td>
</tr>
</tbody>
</table>
## Table 162-67  (Cont.) Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note that the following two executions are different:</td>
<td></td>
</tr>
<tr>
<td>EXEC DBMS_STATS.SET_TABLE_PREFS</td>
<td></td>
</tr>
<tr>
<td>('sh', 'sales', 'INCREMENTAL_STALENESS', 'NULL');</td>
<td></td>
</tr>
<tr>
<td>EXEC DBMS_STATS.SET_TABLE_PREFS</td>
<td></td>
</tr>
<tr>
<td>('sh', 'sales', 'INCREMENTAL_STALENESS', null);</td>
<td></td>
</tr>
<tr>
<td>The first execution uses single quotes to set the preference to the value NULL, whereas the second sets the preference to the default, which is ALLOW_MIXED_FORMAT.</td>
<td></td>
</tr>
<tr>
<td>METHOD_OPT</td>
<td></td>
</tr>
<tr>
<td>Controls column statistics collection and histogram creation. When setting preferences at the global, schema, database, or dictionary level, only FOR ALL syntax is allowed:</td>
<td></td>
</tr>
<tr>
<td>· FOR ALL [INDEXED</td>
<td>HIDDEN] COLUMNS</td>
</tr>
<tr>
<td>[size_clause]</td>
<td></td>
</tr>
<tr>
<td>The size_clause is defined as size_clause := SIZE {integer</td>
<td>REPEAT</td>
</tr>
<tr>
<td>integer — Specifies the number of histogram buckets. The number must be between 1 and 2048.</td>
<td></td>
</tr>
<tr>
<td>REPEAT — Collects histograms only on the columns that already have histograms.</td>
<td></td>
</tr>
<tr>
<td>AUTO — Determines the columns on which to collect histograms based on data distribution and the workload of the columns.</td>
<td></td>
</tr>
<tr>
<td>SKEWONLY — Determines the columns on which to collect histograms based on the data distribution of the columns.</td>
<td></td>
</tr>
<tr>
<td>The default is FOR ALL COLUMNS SIZE AUTO. You can change the value using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
<td></td>
</tr>
</tbody>
</table>
Table 162-67  (Cont.) Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_INVALIDATE</td>
<td>Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. The default is DBMS_STATS.AUTO_INVALIDATE, which means the database decides when to invalidate dependent cursors. If set to TRUE, then the database not invalidate dependent cursors. If set to FALSE, then the procedure invalidates dependent cursors immediately.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Determines the options parameter used in the GATHER_TABLE_STATS procedure. The preference takes the following values:</td>
</tr>
<tr>
<td></td>
<td>• GATHER — Gathers statistics for all objects in the table. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• GATHER AUTO — Gathers all necessary statistics automatically. Oracle recommends setting GATHER AUTO on tables that undergo bulk load operations that gather statistics. This option is only applicable to tables that do not have INCREMENTAL enabled. Running GATHER_TABLE_STATS procedure on these tables with the GATHER AUTO option skips regathering the already fresh statistics.</td>
</tr>
<tr>
<td>PREFERENCE_OVERRIDES_PARAMETER</td>
<td>Determines whether to override the input value of a parameter with the preference value of that parameter for a statistics operation. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• TRUE — Ignores input parameter values, and uses the value of the corresponding preference.</td>
</tr>
<tr>
<td></td>
<td>• FALSE — Obeys input parameter values.</td>
</tr>
<tr>
<td>PUBLISH</td>
<td>Determines whether the database publishes newly gathered statistics after the gathering job completes.</td>
</tr>
<tr>
<td></td>
<td>You can gather statistics without publishing them immediately. This technique enables you to test new statistics before publishing them.</td>
</tr>
<tr>
<td>STALE_PERCENT</td>
<td>Determines the percentage of rows in a table that must change before the statistics on that table are stale and need to be regathered.</td>
</tr>
<tr>
<td></td>
<td>The valid domain for stale_percent is non-negative numbers. The default value is 10, which means that a table having more than 10% of changes is considered stale.</td>
</tr>
</tbody>
</table>
Table 162-67 (Cont.) Preference Descriptions

<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| STAT_CATEGORY   | Specifies which statistics to import or export, accepting multiple values separated by a comma. Values supported:  
  • OBJECT_STATS - table statistics, column statistics and index statistics (default)  
  • SYNOPSES - information to support incremental statistics  
  The value 'OBJECT_STATS, SYNOPSES' specifies table statistics, column statistics, index statistics, and synopses. |
| TABLE_CACHED_BLOCKS | Specifies the average number of blocks assumed to be cached in the buffer cache when calculating the index clustering factor. |
| WAIT_TIME_TO_UPDATE_STATS | Specifies the number of minutes before timing out for locks and pins required for updating statistics. It accepts values in the range 0 to 65535. The default value is 15 minutes. The value 0 gets the locks and pins in no-wait mode. |

Security Model

No special privilege or role is needed to invoke this procedure. To gather statistics concurrently, however, you must either have the DBA role, or have the following privileges in addition to privileges that are required for gathering statistics: CREATE JOB, MANAGE SCHEDULER, and MANAGE ANY QUEUE.

Exceptions

• ORA-20000: Unable to gather statistics concurrently: Resource Manager is not enabled.
• ORA-20001: Invalid input values

Usage Notes

Note the following guidelines:

• The CONCURRENT preference determines whether statistics are gathered concurrently when the user issues GATHER_*_STATS procedures. DBMS_STATS can collect statistics for a single object in parallel based on the value of the DEGREE parameter. However, parallelism is limited to one object. The CONCURRENT preference extends the scope of parallelism to multiple database objects. This approach is primarily intended for multi-CPU systems, and may not be suitable for small databases on single-CPU computers.

To gather statistics concurrently, Resource Manager must be enabled, and the setting for the JOB_QUEUE_PROCESSES initialization parameter must be at least 4.

• If the ownname and tabname are provided, and if a preference has been entered for the table, then the function returns the preference as specified for the table. In all other cases, it returns the global preference if it has been specified, otherwise it returns the default value.
162.7.62 GET_STATS_HISTORY_AVAILABILITY Function

This function returns oldest timestamp where statistics history is available. Users cannot restore statistics to a timestamp older than this one.

Syntax

```sql
DBMS_STATS.GET_STATS_HISTORY_AVAILABILITY
RETURN TIMESTAMP WITH TIMEZONE;
```

Usage Notes

No special privilege or role is needed to invoke this procedure.

162.7.63 GET_STATS_HISTORY_RETENTION Function

This function returns the current statistics history retention value.

Syntax

```sql
DBMS_STATS.GET_STATS_HISTORY_RETENTION
RETURN NUMBER;
```

Usage Notes

No special privilege or role is needed to invoke this procedure.

162.7.64 GET_SYSTEM_STATS Procedure

This procedure gets system statistics from `stattab`, or from the dictionary if `stattab` is `NULL`.

Syntax

```sql
DBMS_STATS.GET_SYSTEM_STATS (  
    status OUT VARCHAR2,  
    dstart OUT DATE,  
    dstop OUT DATE,  
    pname IN VARCHAR2,  
    pvalue OUT NUMBER,  
    stattab IN VARCHAR2 DEFAULT NULL,  
    statid IN VARCHAR2 DEFAULT NULL,  
    statown IN VARCHAR2 DEFAULT NULL);  
```
### Parameters

**Table 162-68  GET_SYSTEM_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Output is one of the following:</td>
</tr>
<tr>
<td></td>
<td>· COMPLETED:</td>
</tr>
<tr>
<td></td>
<td>· AUTOGATHERING:</td>
</tr>
<tr>
<td></td>
<td>· MANUALGATHERING:</td>
</tr>
<tr>
<td></td>
<td>· BADSTATS:</td>
</tr>
<tr>
<td>dstart</td>
<td>Date when statistics gathering started.</td>
</tr>
<tr>
<td></td>
<td>If status = MANUALGATHERING, the start date is returned.</td>
</tr>
<tr>
<td>dstop</td>
<td>Date when statistics gathering stopped.</td>
</tr>
<tr>
<td></td>
<td>· If status = COMPLETE, the finish date is returned.</td>
</tr>
<tr>
<td></td>
<td>· If status = AUTOGATHERING, the future finish date is returned.</td>
</tr>
<tr>
<td></td>
<td>· If status = BADSTATS, the must-finished-by date is returned.</td>
</tr>
<tr>
<td>pname</td>
<td>The parameter name to get, which can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>· iotfrspeed - I/O transfer speed in bytes for each millisecond</td>
</tr>
<tr>
<td></td>
<td>· ioseektim - seek time + latency time + operating system overhead time, in milliseconds</td>
</tr>
<tr>
<td></td>
<td>· sreadtim - average time to read single block (random read), in milliseconds</td>
</tr>
<tr>
<td></td>
<td>· mreadtim - average time to read an mbrc block at once (sequential read), in milliseconds</td>
</tr>
<tr>
<td></td>
<td>· cpuspeed - average number of CPU cycles for each second, in millions, captured for the workload (statistics collected using 'INTERVAL' or 'START' and 'STOP' options)</td>
</tr>
<tr>
<td></td>
<td>· cpuspeednw - average number of CPU cycles for each second, in millions, captured for the no-workload (statistics collected using 'NOWORKLOAD' option.</td>
</tr>
<tr>
<td></td>
<td>· mbrc - average multiblock read count for sequential read, in blocks</td>
</tr>
<tr>
<td></td>
<td>· maxthr - maximum I/O system throughput, in bytes/second</td>
</tr>
<tr>
<td></td>
<td>· slavethr - average slave I/O throughput, in bytes/second</td>
</tr>
<tr>
<td>pvalue</td>
<td>Parameter value to get</td>
</tr>
<tr>
<td>stattab</td>
<td>Identifier of the user statistics table where the statistics will be obtained. If stattab is NULL, the statistics will be obtained from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics saved in the stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
</tbody>
</table>

### Exceptions

- **ORA-20000**: Object does not exist or insufficient privileges
- **ORA-20002**: Bad user statistics table; may need to be upgraded
- **ORA-20003**: Unable to gather system statistics
- **ORA-20004**: Parameter does not exist
Usage Notes

To run this procedure, you need the GATHER_SYSTEM_STATISTICS role.

162.7.65 GET_TABLE_STATS Procedure

This overloaded procedure gets all table-related statistics.

Syntax

DBMS_STATS.GET_TABLE_STATS (  
  ownname         VARCHAR2,  
  tabname         VARCHAR2,  
  partname        VARCHAR2 DEFAULT NULL,  
  stattab         VARCHAR2 DEFAULT NULL,  
  statid          VARCHAR2 DEFAULT NULL,  
  numrows     OUT NUMBER,  
  numblks     OUT NUMBER,  
  avgrlen     OUT NUMBER,  
  statown         VARCHAR2 DEFAULT NULL);

DBMS_STATS.GET_TABLE_STATS (  
  ownname            VARCHAR2,  
  tabname            VARCHAR2,  
  partname           VARCHAR2 DEFAULT NULL,  
  stattab            VARCHAR2 DEFAULT NULL,  
  statid             VARCHAR2 DEFAULT NULL,  
  numrows        OUT NUMBER,  
  numblks        OUT NUMBER,  
  avgrlen        OUT NUMBER,  
  statown            VARCHAR2 DEFAULT NULL,  
  im_imcu_count  OUT NUMBER,  
  im_block_count OUT NUMBER,  
  scanrate       OUT NUMBER);

DBMS_STATS.GET_TABLE_STATS (  
  ownname         VARCHAR2,  
  tabname         VARCHAR2,  
  partname        VARCHAR2 DEFAULT NULL,  
  stattab         VARCHAR2 DEFAULT NULL,  
  statid          VARCHAR2 DEFAULT NULL,  
  numrows     OUT NUMBER,  
  numblks     OUT NUMBER,  
  avgrlen     OUT NUMBER,  
  statown    OUT NUMBER,  
  cachedblk   OUT NUMBER,  
  cachehit    OUT NUMBER);

Chapter 162
Summary of DBMS_STATS Subprograms
Parameters

Table 162-69  GET_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the table or partition.</td>
</tr>
<tr>
<td>numblks</td>
<td>Number of blocks in the table or partition.</td>
</tr>
<tr>
<td>avgrlen</td>
<td>Average row length for the table or partition.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from ownname).</td>
</tr>
<tr>
<td>im_imcu_count</td>
<td>The number of In-Memory Compression Units (IMCUs) in the table or partition.</td>
</tr>
<tr>
<td>im_block_count</td>
<td>The number of In-Memory blocks in the table or partition. An In-Memory block corresponds to a specific data block on disk. If the table is fully populated in the IM column store, then the number of In-Memory blocks equals the number of data blocks.</td>
</tr>
<tr>
<td>scanrate</td>
<td>The rate, in MB/s, at which the database scans external tables. This parameter is relevant only for external tables.</td>
</tr>
<tr>
<td>cachedblk</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>cachehit</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>

Security Model

Before invoking this procedure, ensure that the table exists. To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you must be either the owner of the table, or have the ANALYZE ANY DICTIONARY or SYSDBA privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

ORA-20002: Bad user statistics table; may need to upgrade it

Usage Notes

• The optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The database calculates the total cost of the operation by combining the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
• The database maintains `cachedblk` and `cachehit` at all times. However, the database uses the corresponding caching statistics for optimization as part of the table and index statistics only when the user calls the `DBMS_STATS.GATHER_[TABLE/INDEX/SHEMA/DATABASE]_STATS` procedure for automatic mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. To prevent the user from utilizing inaccurate and unreliable data, the optimizer computes a “confidence factor” for each `cachehit` and a `cachedblk` for each object. If the confidence factor for the value meets confidence criteria, then the database uses this value; otherwise, the database uses defaults.

• The automatic maintenance algorithm for object caching statistics assumes that only one major database workload exists. The algorithm adjusts statistics to this workload, ignoring other “minor” workloads. If this assumption is false, then you must use manual mode for maintaining object caching statistics.

• The object caching statistics maintenance algorithm for automatic mode prevents you from using statistics in the following situations
  – When not enough data has been analyzed, such as when an object has been recently created
  – When the system does not have one major workload resulting in averages not corresponding to real values

• The database does not support export or import of statistics across databases of different character sets.

See Also:

Oracle Database SQL Tuning Guide to learn how to manage optimizer statistics preferences

162.7.66 IMPLEMENT_ADVISOR_TASK Function

This function implements the recommendations made by Optimizer Statistics Advisor.

Syntax

```sql
DBMS_STATS.IMPLEMENT_ADVISOR_TASK (  
  task_name          IN   VARCHAR2,  
  execution_name     IN   VARCHAR2    := NULL,  
  level              IN   VARCHAR2    := 'TYPICAL')  
RETURN CLOB;
```

Parameters

Table 162-70 IMPLEMENT_ADVISOR_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
</tbody>
</table>
Table 162-70  (Cont.) IMPLEMENT_ADVISOR_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execution_name</td>
<td>A name that qualifies and identifies an advisor execution. If not specified, then the advisor automatically generates it. If the specified execution conflicts with the name of an existing execution, then the function returns an error.</td>
</tr>
<tr>
<td>level</td>
<td>The level of the implementation. Possible values are</td>
</tr>
<tr>
<td></td>
<td>• ALL: Ignores the filters and implements all recommendations.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: Implements the recommendations according to the filters in place.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

• To execute this subprogram, you must have the ADVISOR privilege.
• You must be the owner of the task.
• You can execute this subprogram for AUTO_STATS_ADVISOR_TASK, which is predefined.
• This subprogram executes using invoker’s rights.

The results of performing this task depend on the privileges of the executing user:

• SYSTEM level
  Only users with both the ANALYZE ANY and ANALYZE ANY DICTIONARY privileges can perform this task on system-level rules.
• Operation level
  The results depend on the following privileges:
  – Users with both the ANALYZE ANY and ANALYZE ANY DICTIONARY privileges can perform this task for all statistics operations.
  – Users with the ANALYZE ANY privilege but not the ANALYZE ANY DICTIONARY privilege can perform this task for statistics operations related to any schema except SYS.
  – Users with the ANALYZE ANY DICTIONARY privilege but not the ANALYZE ANY privilege can perform this task for statistics operations related to their own schema and the SYS schema.
  – Users with neither the ANALYZE ANY nor the ANALYZE ANY DICTIONARY privilege can only perform this operation for statistics operations relating to their own schema.
• Object level
  Users can perform this task for any object for which they have statistics collection privileges.

Return Values

This function returns an XML CLOB that indicates which recommendations were successfully implemented.
Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
- ORA-20012: Optimizer Statistics Advisor errors

Example 162-10  Implementing Optimizer Statistics Advisor Recommendations

This script illustrates a basic Optimizer Statistics Advisor session. It creates a task, executes it, generates a report, and then implements the recommendations.

```sql
DECLARE
    v_tname      VARCHAR2(128) := 'my_task';
    v_ename      VARCHAR2(128) := NULL;
    v_report     CLOB := null;
    v_script     CLOB := null;
    v_imp_result CLOB;
BEGIN
    -- create a task
    v_tname := DBMS_STATS.CREATE_ADVISOR_TASK(v_tname);

    -- execute the task
    v_ename := DBMS_STATS.EXECUTE_ADVISOR_TASK(v_tname);

    -- view the task report
    v_report := DBMS_STATS.REPORT_ADVISOR_TASK(v_tname);
    DBMS_OUTPUT.PUT_LINE(v_report);

    -- implement all recommendations
    v_imp_result := DBMS_STATS.IMPLEMENT_ADVISOR_TASK(v_tname);
END;
```

162.7.67 IMPORT_COLUMN_STATS Procedure

This procedure retrieves statistics for a particular column from the user statistics table identified by `stattab` and stores them in the dictionary.

Syntax

```sql
DBMS_STATS.IMPORT_COLUMN_STATS ( 
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    colname      VARCHAR2,
    partname     VARCHAR2 DEFAULT NULL,
    stattab      VARCHAR2,
    statid       VARCHAR2 DEFAULT NULL,
    statown      VARCHAR2 DEFAULT NULL,
    no_invalidate BOOLEAN DEFAULT to_no_invalidate_type ( 
        get_param('NO_INVALIDATE')),
    force        BOOLEAN DEFAULT FALSE);
```
Parameters

Table 162-71  IMPORT_COLUMN_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column or extension</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition column statistics are imported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier to associate with these statistics within stattab (optional)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>If set to TRUE, imports statistics even if statistics are locked</td>
</tr>
</tbody>
</table>

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

ORA-20005: Object statistics are locked

Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

162.7.68 IMPORT_DATABASE_PREFS Procedure

This procedure is used to import the statistics preferences of all the tables, excluding the tables owned by Oracle. These tables can by included by passing TRUE for the add_sys parameter.

Syntax

```sql
DBMS_STATS.IMPORT_DATABASE_PREFS (
    stattab    IN  VARCHAR2,
```
Parameters

Table 162-72 IMPORT_DATABASE_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Statistics table name where to import the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>add_sys</td>
<td>Value TRUE will include the Oracle-owned tables</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Insufficient privileges.

Usage Notes

- To run this procedure, you need to have the SYSDBA role, or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.
- Oracle does not support export or import of statistics across databases of different character sets.

Examples

DBMS_STATS.IMPORT_DATABASE_PREFS('STATTAB', statown=>'SH');

162.7.69 IMPORT_DATABASE_STATS Procedure

This procedure imports statistics for all objects in the database from the user statistics table and stores them in the data dictionary.

Syntax

```
DBMS_STATS.IMPORT_DATABASE_STATS (           
  stattab VARCHAR2,                         
  statid VARCHAR2 DEFAULT NULL,            
  statown VARCHAR2 DEFAULT NULL,           
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(  
    get_param('NO_INVALIDATE')),            
  force BOOLEAN DEFAULT FALSE,             
  stat_category VARCHAR2 DEFAULT FALSE)    
```

Parameters

Table 162-73  IMPORT_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Specifies the statistics table that contains the statistics to be imported.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Indicates whether to invalidate dependent cursors.</td>
</tr>
<tr>
<td></td>
<td>- TRUE, then the procedure does not invalidate dependent cursors. If set to</td>
</tr>
<tr>
<td></td>
<td>FALSE, then the procedure invalidates the dependent cursors immediately.</td>
</tr>
<tr>
<td></td>
<td>Use DBMS_STATS.AUTO_INVALIDATE to let Oracle Database decide when</td>
</tr>
<tr>
<td></td>
<td>to invalidate dependent cursors. This is the default.</td>
</tr>
<tr>
<td></td>
<td>The default can be changed using the SET_DATABASE_PREFS Procedure,</td>
</tr>
<tr>
<td></td>
<td>SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and</td>
</tr>
<tr>
<td></td>
<td>SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>Overrides statistics locked at the object (table) level:</td>
</tr>
<tr>
<td></td>
<td>- TRUE - Ignores the statistics lock and imports the statistics</td>
</tr>
<tr>
<td></td>
<td>- FALSE - The statistics will be imported only if they are not locked</td>
</tr>
<tr>
<td>stat_category</td>
<td>Specifies what statistics to import, accepting multiple values separated by</td>
</tr>
<tr>
<td></td>
<td>a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>- OBJECT_STATS - table statistics, column statistics and index statistics</td>
</tr>
<tr>
<td></td>
<td>(Default)</td>
</tr>
<tr>
<td></td>
<td>- SYNOPSES - information to support incremental statistics</td>
</tr>
</tbody>
</table>

Security Model

You must have either the SYSDBA privilege or both the ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

Usage Notes

Oracle Database does not support export or import of statistics across databases of different character sets.

162.7.70 IMPORT_DICTIONARY_STATS Procedure

This procedure imports statistics for all data dictionary schemas (SYS, SYSTEM, and RDBMS component schemas) from the user statistics table and stores them in the dictionary.

Syntax

```sql
DBMS_STATS.IMPORT_DICTIONARY_STATS (     stattab VARCHAR2,     statid VARCHAR2 DEFAULT NULL,
```
statown    VARCHAR2 DEFAULT NULL,
no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
    get_param('NO_INVALIDATE')),
force    BOOLEAN DEFAULT FALSE,
stat_category    VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);

Parameters

Table 162-74  IMPORT_DICTIONARY_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>The (optional) identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Indicates whether to invalidate dependent cursors. If this parameter is set to TRUE, then the procedure does not invalidate dependent cursors. If set to FALSE, then the procedure invalidates the dependent cursors immediately. Use DBMS_STATS.AUTO_INVALIDATE to let Oracle Database decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>Overrides statistics lock at the object (table) level:</td>
</tr>
<tr>
<td></td>
<td>• TRUE - Ignores the statistics lock and imports the statistics.</td>
</tr>
<tr>
<td></td>
<td>• FALSE - The statistics will be imported only if there is no lock.</td>
</tr>
<tr>
<td>stat_category</td>
<td>Specifies what statistics to import, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_STATS - table statistics, column statistics and index statistics (Default)</td>
</tr>
<tr>
<td></td>
<td>• SYNOPSES - information to support incremental statistics</td>
</tr>
</tbody>
</table>

Security Model

You must have either the SYSDBA privilege or both the ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid or inconsistent values in the user statistics table
ORA-20002: Bad user statistics table, may need to upgrade it

Usage Notes

Oracle Database does not support export or import of statistics across databases of different character sets.
162.7.71 IMPORT_FIXED_OBJECTS_STATS Procedure

This procedure imports statistics for fixed tables from the user statistics table and stores them in the data dictionary.

Syntax

DBMS_STATS.IMPORT_FIXED_OBJECTS_STATS (  
  stattab       VARCHAR2,  
  statid        VARCHAR2 DEFAULT NULL,  
  statown       VARCHAR2 DEFAULT NULL,  
  no_invalidate BOOLEAN  DEFAULT to_no_invalidate_type(  
    get_param('NO_INVALIDATE')),  
  force         BOOLEAN  DEFAULT FALSE);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Indicates whether to invalidate dependent cursors. If this parameter is set to TRUE, then the procedure does not invalidate dependent cursors. If set to FALSE, then the procedure invalidates the dependent cursors immediately. Use DBMS_STATS.AUTO_INVALIDATE to let Oracle Database decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>Overrides statistics lock:</td>
</tr>
<tr>
<td></td>
<td>• TRUE - Ignores the statistics lock and imports the statistics</td>
</tr>
<tr>
<td></td>
<td>• FALSE - The statistics will be imported only if there is no lock</td>
</tr>
</tbody>
</table>

Security Model

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid or inconsistent values in the user statistics table
ORA-20002: Bad user statistics table, may need to upgrade it

Usage Notes

Oracle Database does not support export or import of statistics across databases of different character sets.
162.7.72 IMPORT_INDEX_STATS Procedure

This procedure retrieves statistics for a particular index from the user statistics table identified by `stattab` and stores them in the dictionary.

Syntax

```sql
DBMS_STATS.IMPORT_INDEX_STATS (  
  ownname       VARCHAR2,
  indname       VARCHAR2,
  partname      VARCHAR2 DEFAULT NULL,
  stattab       VARCHAR2,
  statid        VARCHAR2 DEFAULT NULL,
  statown       VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(  
    get_param('NO_INVALIDATE')),
  force         BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the index partition. If the index is partitioned and if <code>partname</code> is <code>NULL</code>, then global and partition index statistics are imported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within <code>stattab</code></td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing <code>stattab</code> (if different than <code>ownname</code>)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to <code>TRUE</code>. The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code>. Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <code>SET_DATABASE_PREFS Procedure</code>, <code>SET_GLOBAL_PREFS Procedure</code>, <code>SET_SCHEMA_PREFS Procedure</code> and <code>SET_TABLE_PREFS Procedure</code>.</td>
</tr>
<tr>
<td>force</td>
<td>Imports statistics even if index statistics are locked</td>
</tr>
</tbody>
</table>

Exceptions

- `ORA-20000`: Object does not exist or insufficient privileges
- `ORA-20001`: Invalid or inconsistent values in the user statistics table
- `ORA-20005`: Object statistics are locked
Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Oracle does not support export or import of statistics across databases of different character sets.

162.7.73 IMPORT_SCHEMA_PREFS Procedure

This procedure is used to import the statistics preferences of all the tables owned by the specified owner name.

Syntax

```sql
DBMS_STATS.IMPORT_SCHEMA_PREFS (
    ownname    IN  VARCHAR2,
    stattab    IN  VARCHAR2,
    statid     IN  VARCHAR2 DEFAULT NULL,
    statown    IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>stattab</td>
<td>Statistics table name from where to import the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>(Optional) Identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if other than ownname)</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

- To run this procedure, you need to connect as owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

Examples

```sql
DBMS_STATS.IMPORT_SCHEMA_PREFS('SH', 'STAT');
```
162.7.74 IMPORT_SCHEMA_STATS Procedure

This procedure retrieves statistics for all objects in the schema identified by `ownname` from the user statistics table and stores them in the dictionary.

Syntax

```sql
DBMS_STATS.IMPORT_SCHEMA_STATS (  
    ownname         VARCHAR2,  
    stattab         VARCHAR2,  
    statid          VARCHAR2 DEFAULT NULL,  
    statown         VARCHAR2 DEFAULT NULL,  
    no_invalidate   BOOLEAN DEFAULT to_no_invalidate_type(  
    force           BOOLEAN DEFAULT FALSE,  
    stat_category   VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ownname</code></td>
<td>Name of the schema</td>
</tr>
<tr>
<td><code>stattab</code></td>
<td>User statistics table identifier describing from where to retrieve the statistics</td>
</tr>
<tr>
<td><code>statid</code></td>
<td>Identifier (optional) to associate with these statistics within <code>stattab</code></td>
</tr>
<tr>
<td><code>statown</code></td>
<td>Schema containing <code>stattab</code> (if different than <code>ownname</code>)</td>
</tr>
<tr>
<td><code>no_invalidate</code></td>
<td>Does not invalidate the dependent cursors if set to <code>TRUE</code>. The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code>. Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <code>SET_DATABASE_PREFS Procedure</code>, <code>SET_GLOBAL_PREFS Procedure</code>, <code>SET_SCHEMA_PREFS Procedure</code> and <code>SET_TABLE_PREFS Procedure</code>.</td>
</tr>
<tr>
<td><code>force</code></td>
<td>Overrides statistics locked at the object (table) level:</td>
</tr>
<tr>
<td></td>
<td>- <code>TRUE</code> - Ignores the statistics lock and imports the statistics.</td>
</tr>
<tr>
<td></td>
<td>- <code>FALSE</code> - Statistics will be imported only if there is no lock.</td>
</tr>
<tr>
<td><code>stat_category</code></td>
<td>Specifies what statistics to import, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>- <code>OBJECT_STATS</code> - table statistics, column statistics and index statistics (Default)</td>
</tr>
<tr>
<td></td>
<td>- <code>SYNOPSES</code> - information to support incremental statistics</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table
**Usage Notes**

To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.

Oracle does not support export or import of statistics across databases of different character sets.

**162.7.75 IMPORT_SYSTEM_STATS Procedure**

This procedure retrieves system statistics from the user statistics table, identified by `stattab`, and stores the statistics in the dictionary.

**Syntax**

```sql
DBMS_STATS.IMPORT_SYSTEM_STATS (
    stattab       VARCHAR2,
    statid        VARCHAR2 DEFAULT NULL,
    statown       VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>Identifier of the user statistics table where the statistics will be retrieved</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics retrieved from the <code>stattab</code></td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing <code>stattab</code> (if different from current schema)</td>
</tr>
</tbody>
</table>

**Exceptions**

- **ORA-20000**: Object does not exist or insufficient privileges
- **ORA-20001**: Invalid or inconsistent values in the user statistics table
- **ORA-20002**: Bad user statistics table; may need to be upgraded
- **ORA-20003**: Unable to import system statistics

**Usage Notes**

To run this procedure, you need the `GATHER_SYSTEM_STATISTICS` role.

Oracle does not support export or import of statistics across databases of different character sets.
162.7.76 IMPORT_TABLE_PREFS Procedure

This procedure is used to set the statistics preferences of the specified table in the specified schema.

Syntax

```
DBMS_STATS.IMPORT_TABLE_PREFS (
    ownname    IN  VARCHAR2,
    tabname    IN  VARCHAR2,
    stattab    IN  VARCHAR2,
    statid     IN  VARCHAR2 DEFAULT NULL,
    statown    IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
<tr>
<td>stattab</td>
<td>Statistics table name from where to import the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>(Optional) Identifier to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if other than ownname)</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

Usage Notes

- To run this procedure, you need to connect as owner of the table, or have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

Examples

```
DBMS_STATS.IMPORT_TABLE_PREFS('SH', 'SALES', 'STAT');
```

162.7.77 IMPORT_TABLE_STATS Procedure

This procedure retrieves statistics for a particular table from the user statistics table identified by stattab and stores them in the dictionary.

Cascade results in all index statistics associated with the specified table being imported as well.
Syntax

```sql
DBMS_STATS.IMPORT_TABLE_STATS (
    ownname VARCHAR2,
    tabname VARCHAR2,
    partname VARCHAR2 DEFAULT NULL,
    stattab VARCHAR2,
    statid VARCHAR2 DEFAULT NULL,
    cascade BOOLEAN DEFAULT TRUE,
    statown VARCHAR2 DEFAULT NULL,
    no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
        get_param('NO_INVALIDATE')),
    force BOOLEAN DEFAULT FALSE,
    stat_category VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

Parameters

Table 162-81  IMPORT_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are imported.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing from where to retrieve the statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>cascade</td>
<td>If true, column and index statistics for this table are also imported</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>Imports statistics even if table statistics are locked</td>
</tr>
<tr>
<td>stat_category</td>
<td>Specifies what statistics to import, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>• OBJECT_STATS - table statistics, column statistics and index statistics (Default)</td>
</tr>
<tr>
<td></td>
<td>• SYNOPSES - information to support incremental statistics</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table
Usage Notes

To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.

Oracle does not support export or import of statistics across databases of different character sets.

162.7.78 INTERRUPT_ADVISOR_TASK Procedure

This procedure interrupts a currently executing Optimizer Statistics Advisor task.

The task ends its operations as it does when at a normal exit, at which point you can access intermediate results. You can also resume the task using the "RESUME_ADVISOR_TASK Procedure".

Syntax

```
DBMS_STATS.INTERRUPT_ADVISOR_TASK (
    task_name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the `ADVISOR` privilege.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.

Consider a case in which a task is executed by one user, interrupted, and then resumed by a different user. In this case, Optimizer Statistics Advisor bases its checks of the resumed execution on the privilege of the user who resumed the task.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
- ORA-20012: Optimizer Statistics Advisor errors
Example 162-11  Interrupting an Optimizer Statistics Advisor Task

In this example, you start a SQL*Plus session, and then create and execute an advisor task named `my_task`:

```sql
DECLARE
    v_tname   VARCHAR2(128) := 'my_task';
    v_ename   VARCHAR2(128) := NULL;
BEGIN
    -- create a task
    v_tname := DBMS_STATS.CREATE_ADVISOR_TASK(v_tname);

    -- execute the task
    v_ename := DBMS_STATS.EXECUTE_ADVISOR_TASK(v_tname);
END;
/
```

In a separate terminal, you start a second SQL*Plus session, and then execute the following program:

```sql
XEC DBMS_STATS.INTERRUPT_ADVISOR_TASK('my_task');
```

The first session returns an `ORA-13632` to indicate the cancelation of the task:

```
ORA-13638: The user interrupted the current operation.
```

162.7.79 LOCK_PARTITION_STATS Procedure

This procedure enables the user to lock statistics for a partition.

**Syntax**

```sql
DBMS_STATS.LOCK_PARTITION_STATS ( 
    ownname    VARCHAR2,
    tabname    VARCHAR2,
    partname   VARCHAR2);
```

**Parameters**

**Table 162-83  LOCK_PARTITION_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema to lock</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
<tr>
<td>partname</td>
<td>[Sub]Partition name</td>
</tr>
</tbody>
</table>

**Usage Notes**

To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
162.7.80 LOCK_SCHEMA_STATS Procedure

This procedure locks the statistics of all tables of a schema.

Syntax

```
DBMS_STATS.LOCK_SCHEMA_STATS (
    ownname    VARCHAR2);
```

Parameters

**Table 162-84  LOCK_SCHEMA_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema to lock</td>
</tr>
</tbody>
</table>

Usage Notes

- To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
- When statistics on a table are locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The `SET_*`, `DELETE_*`, `IMPORT_*`, `GATHER_*` procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as `GATHER_SCHEMA_STATS`) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.
- This procedure either freezes the current set of the statistics or keeps the statistics empty (uncollected) to use dynamic statistics.
- The locked or unlocked state is not exported along with the table statistics when using `EXPORT_*_STATS` procedures.
- Neither the `UNLOCK_SCHEMA_STATS Procedure` nor the `UNLOCK_TABLE_STATS Procedure` is designed to unlock statistics of corresponding partitions. When you invoke the `LOCK_TABLE_STATS Procedure`, it sets the statistics lock bit at the table level. In that case, you cannot gather statistics on dependent objects such as partitions and indexes. By the same token, if table statistics are locked, the dependents are locked and you do not need to explicitly invoke the `LOCK_PARTITION_STATS Procedure`. 
162.7.81 LOCK_TABLE_STATS Procedure

This procedure locks the statistics on the table.

Syntax

```sql
DBMS_STATS.LOCK_TABLE_STATS (  
    ownname    VARCHAR2,
    tabname    VARCHAR2);
```

Parameters

Table 162-85  LOCK_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
</tbody>
</table>

Usage Notes

- To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
- When statistics on a table are locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The `SET_*`, `DELETE_*`, `IMPORT_*`, `GATHER_*` procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as `GATHER_SCHEMA_STATS`) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.
- This procedure either freezes the current set of the statistics or keeps the statistics empty (uncollected) to use dynamic statistics.
- The locked or unlocked state is not exported along with the table statistics when using `EXPORT_*_STATS` procedures.
- Neither the `UNLOCK_SCHEMA_STATS Procedure` nor the `UNLOCK_TABLE_STATS Procedure` is designed to unlock statistics of corresponding partitions. When you invoke the `LOCK_TABLE_STATS Procedure`, it sets the statistics lock bit at the table level. In that case, you cannot gather statistics on dependent objects such as partitions and indexes. By the same token, if table statistics are locked, the dependents are locked and you do not need to explicitly invoke the `LOCK_PARTITION_STATS Procedure`. 
162.7.82 MERGE_COL_USAGE Procedure

This procedure merges column usage information from a source database by means of a dblink into the local database.

If column usage information already exists for a given table or column, `MERGE_COL_USAGE` will combine both the local and the remote information.

Syntax

```sql
DBMS_STATS.MERGE_COL_USAGE (  
    dblink IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblink</td>
<td>Name of dblink</td>
</tr>
</tbody>
</table>

Usage Notes

User must be SYS to execute this procedure. Additionally, the user specified during the creation of the dblink is expected to have privileges to select from tables in the SYS schema.

Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Parameter dblink cannot be NULL

ORA-20002: Unable to create a TEMP table

162.7.83 PREPARE_COLUMN_VALUES Procedures

These procedures convert user-specified minimum, maximum, and histogram endpoint actual values into Oracle’s internal representation for future storage using `SET_COLUMN_STATS`.

Syntax

```sql
DBMS_STATS.PREPARE_COLUMN_VALUES (  
    srec IN OUT StatRec,  
    charvals CHARARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (  
    srec IN OUT StatRec,  
    datevals DATEARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (  
    srec IN OUT StatRec,  
    dblvals DBLARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (  
    srec IN OUT StatRec,  
    intvals INTARRAY);
```
srec IN OUT StatRec,
fltvals FLTARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (    
  srec IN OUT StatRec,
  numvals NUMARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (    
  srec IN OUT StatRec,
  rawvals RAWARRAY);

Parameters

Table 162-87   PREPARE_COLUMN_VALUES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srec.epc</td>
<td>Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 2050, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.</td>
</tr>
<tr>
<td>srec.bkvals</td>
<td>If you want a frequency or hybrid histogram, this array contains the number of occurrences of each distinct value specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.</td>
</tr>
<tr>
<td>srec.rpcnts</td>
<td>If you want a hybrid histogram, this array contains the total frequency of values that are less than or equal to each distinct value specified in charvals, datevals, numvals, or rawvals. Otherwise, it is merely an output argument and must be set to NULL when this procedure is called. As an example, for a given array numvals with numvals(i)=4, rpcnts(i)=13 means that there are 13 rows in the column which are less than or equal to 4.</td>
</tr>
</tbody>
</table>

Note:
- Whenever srec.rpcnts is populated, srec.bkvals must be populated as described above.
- Whenever bkvals and/or rpcnts are populated, there should not be any duplicates in charvals, datevals, numvals, or rawvals.

Datatype-specific input parameters (use one) are shown in Table 162-88.
### Table 162-88 Datatype-Specific Input Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charvals</td>
<td>The array of values when the column type is character-based. Up to the first 64 bytes of each string should be provided. Arrays must have between 2 and 2050 entries, inclusive. If the datatype is fixed \texttt{CHAR}, the strings must be space-padded to 15 characters for correct normalization.</td>
</tr>
<tr>
<td>datevals</td>
<td>Array of values when the column type is date-based</td>
</tr>
<tr>
<td>dblvals</td>
<td>Array of values when the column type is double-based</td>
</tr>
<tr>
<td>fltvals</td>
<td>Array of values when the column type is float-based</td>
</tr>
<tr>
<td>numvals</td>
<td>Array of values when the column type is numeric-based</td>
</tr>
<tr>
<td>rawvals</td>
<td>Array of values when the column type is \texttt{RAW}. Up to the first 64 bytes of each value should be provided.</td>
</tr>
<tr>
<td>nvmin, nvmax</td>
<td>Minimum and maximum values when the column type is national character set based. No histogram information can be provided for a column of this type. If the datatype is fixed \texttt{CHAR}, the strings must be space-padded to 15 characters for correct normalization.</td>
</tr>
<tr>
<td>rwmin, rwmax</td>
<td>Minimum and maximum values when the column type is \texttt{rowid}. No histogram information is provided for a column of this type.</td>
</tr>
</tbody>
</table>

### Output Parameters

### Table 162-89 PREPARE_COLUMN_VALUES Procedure Output Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srec.minval</td>
<td>Internal representation of the minimum suitable for use in a call to \texttt{SET_COLUMN_STATS}</td>
</tr>
<tr>
<td>srec.maxval</td>
<td>Internal representation of the maximum suitable for use in a call to \texttt{SET_COLUMN_STATS}</td>
</tr>
<tr>
<td>srec.bkvals</td>
<td>Array suitable for use in a call to \texttt{SET_COLUMN_STATS}</td>
</tr>
<tr>
<td>srec.novals</td>
<td>Array suitable for use in a call to \texttt{SET_COLUMN_STATS}</td>
</tr>
<tr>
<td>srec.eavals</td>
<td>Array suitable for use in a call to \texttt{SET_COLUMN_STATS}</td>
</tr>
<tr>
<td>srec.rpcnts</td>
<td>Array suitable for use in a call to \texttt{SET_COLUMN_STATS}</td>
</tr>
</tbody>
</table>

### Exceptions

ORA-20001: Invalid or inconsistent input values

### Usage Notes

No special privilege or role is needed to invoke this procedure.
162.7.84 PREPARE_COLUMN_VALUES_NVARCHAR Procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint actual values into Oracle’s internal representation for future storage using the SET_COLUMN_STATS Procedures.

Syntax

DBMS_STATS.PREPARE_COLUMN_VALUES_NVARCHAR (  
srec IN OUT StatRec,  
nvmin NVARCHAR2,  
nvmax NVARCHAR2);  

Parameters

Table 162-90   PREPARE_COLUMN_VALUES_NVARCHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srec.epc</td>
<td>Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 2050, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as it is for a call to SET_COLUMN_STATS.</td>
</tr>
<tr>
<td>srec.bkvals</td>
<td>If you want a frequency or hybrid histogram, then this array contains the number of occurrences of each distinct value specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.</td>
</tr>
<tr>
<td>srec.rpcnts</td>
<td>If you want a hybrid histogram, this array contains the total frequency of values that are less than or equal to each distinct value specified in charvals, datevals, numvals, or rawvals. Otherwise, it is merely an output argument and must be set to NULL when this procedure is called. As an example, for a given array numvals with numvals(i)=4, rpcnts(i)=13 means that there are 13 rows in the column which are less than or equal to 4.</td>
</tr>
</tbody>
</table>

Note:
- Whenever srec.rpcnts is populated, srec.bkvals must be populated as described above.
- Whenever bkvals and/or rpcnts are populated, there should not be any duplicates in charvals, datevals, numvals, or rawvals.

Datatype-specific input parameters (use one) are shown in Table 162-88.
Table 162-91  PREPARE_COLUMN_VALUES_NVARCHAR Datatype-Specific
Input Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvmin, nvmax</td>
<td>The minimum and maximum values when the column type is national character set based. No histogram information can be provided for a column of this type. If the datatype is fixed CHAR, the strings must be space-padded to 15 characters for correct normalization.</td>
</tr>
</tbody>
</table>

Output Parameters

Table 162-92  PREPARE_COLUMN_VALUES_NVARCHAR Procedure Output Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srec.minval</td>
<td>Internal representation of the minimum suitable for use in a call to SET_COLUMN_STATS</td>
</tr>
<tr>
<td>srec.maxval</td>
<td>Internal representation of the maximum suitable for use in a call to SET_COLUMN_STATS</td>
</tr>
<tr>
<td>srec.bkvals</td>
<td>Array suitable for use in a call to SET_COLUMN_STATS.</td>
</tr>
<tr>
<td>srec.novals</td>
<td>Array suitable for use in a call to SET_COLUMN_STATS</td>
</tr>
<tr>
<td>srec.eavals</td>
<td>Array suitable for use in a call to SET_COLUMN_STATS</td>
</tr>
<tr>
<td>srec.rpcnts</td>
<td>Array suitable for use in a call to SET_COLUMN_STATS</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20001: Invalid or inconsistent input values

Usage Notes

No special privilege or role is needed to invoke this procedure.

Related Topics

- SET_COLUMN_STATS Procedures
  This procedure sets column-related information.

162.7.85 PREPARE_COLUMN_VALUES_ROWSID Procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle’s internal representation for future storage using SET_COLUMN_STATS.

Syntax

```sql
DBMS_STATS.PREPARE_COLUMN_VALUES_ROWSID (  
  srec  IN OUT StatRec,  
  rwmin  ROWID,  
  rwmax  ROWID);  
```
Pragmas

```plaintext
pragma restrict_references(prepare_column_values_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 162-93  PREPARE_COLUMN_VALUES_ROWID Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srec</td>
<td>Values (IN):</td>
</tr>
<tr>
<td></td>
<td>· epc</td>
</tr>
<tr>
<td></td>
<td>· bkvals</td>
</tr>
<tr>
<td></td>
<td>· rpcnts</td>
</tr>
<tr>
<td></td>
<td>Values (OUT):</td>
</tr>
<tr>
<td></td>
<td>· minval</td>
</tr>
<tr>
<td></td>
<td>· maxval</td>
</tr>
<tr>
<td></td>
<td>· bkvals</td>
</tr>
<tr>
<td></td>
<td>· novals</td>
</tr>
<tr>
<td></td>
<td>· eavals</td>
</tr>
<tr>
<td></td>
<td>· rpcnts</td>
</tr>
<tr>
<td>rwmin</td>
<td>Minimum value when the column type is <code>rowid</code>. No histogram information is provided for a column of this type.</td>
</tr>
<tr>
<td>rwmax</td>
<td>Maximum value when the column type is <code>rowid</code>. No histogram information is provided for a column of this type.</td>
</tr>
</tbody>
</table>

Table 162-94  StatRec Record Type Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>epc (IN)</td>
<td>Number of values specified in <code>charvals</code>, <code>datevals</code>, <code>dblvals</code>, <code>fltvals</code>, <code>numvals</code>, or <code>rawvals</code>. This value must be between 2 and 2050, inclusive, and it should be set to 2 for procedures which do not allow histogram information (<code>nvarchar</code> and <code>rowid</code>). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to <code>SET_COLUMN_STATS</code>.</td>
</tr>
<tr>
<td>bkvals (IN)</td>
<td>If you want a frequency or hybrid histogram, this array contains the number of occurrences of each distinct value specified in <code>charvals</code>, <code>datevals</code>, <code>dblvals</code>, <code>fltvals</code>, <code>numvals</code>, or <code>rawvals</code>. Otherwise, it is merely an output parameter, and it must be set to <code>NULL</code> when this procedure is called.</td>
</tr>
</tbody>
</table>
Table 162-94  (Cont.) StatRec Record Type Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| rpcnts (IN) | If you want a hybrid histogram, this array contains the total frequency of values that are less than or equal to each distinct value specified in charvals, datevals, numvals, or rawvals. Otherwise, it is merely an output argument and must be set to NULL when this procedure is called. As an example, for a given array numvals with numvals(i)=4, rpcnts(i)=13 means that there are 13 rows in the column which are less than or equal to 4. Note:  
  • Whenever srec.rpcnts is populated, srec.bkvals must be populated as described above.  
  • Whenever bkvals and/or rpcnts are populated, there should not be any duplicates in charvals, datevals, numvals, or rawvals. |
| minval (OUT) | Internal representation of the minimum suitable for use in a call to SET_COLUMN_STATS. |
| maxval (OUT) | Internal representation of the maximum suitable for use in a call to SET_COLUMN_STATS. |
| bkvals (OUT) | Array suitable for use in a call to SET_COLUMN_STATS. |
| novals (OUT) | Array suitable for use in a call to SET_COLUMN_STATS. |
| eavals (OUT) | Array suitable for use in a call to SET_COLUMN_STATS. |
| rpcnts (OUT) | Array suitable for use in a call to SET_COLUMN_STATS. |

Usage Notes

No special privilege or role is needed to invoke this procedure.

162.7.86 PUBLISH_PENDING_STATS Procedure

This procedure is used to publish the statistics gathered and stored as pending.

Syntax

```
DBMS_STATS.PUBLISH_PENDING_STATS (  
    ownname IN VARCHAR2 DEFAULT USER,  
    tabname IN VARCHAR2,  
    no_invalidate BOOLEAN DEFAULT TO_NO_INVALIDATE_TYPE(GET_PARAM('NO_INVALIDATE')),  
    force IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 162-95  PUBLISH_PENDING_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
</tbody>
</table>
Table 162-95  (Cont.) PUBLISH_PENDING_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Do not invalidate the dependent cursors if set to TRUE. The procedure</td>
</tr>
<tr>
<td></td>
<td>invalidates the dependent cursors immediately if set to FALSE. Use</td>
</tr>
<tr>
<td></td>
<td>DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate</td>
</tr>
<tr>
<td></td>
<td>dependent cursors. This is the default. The default can be changed using</td>
</tr>
<tr>
<td></td>
<td>the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS</td>
</tr>
<tr>
<td></td>
<td>Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, will override the lock</td>
</tr>
</tbody>
</table>

Security Model

To run this procedure, you must have the same privilege for gathering statistics on the tables that will be touched by this procedure.

Exceptions

ORA-20000: Insufficient privileges

Usage Notes

- If the parameter tabname is NULL then publish applies to all tables of the specified schema.
- The default owner/schema is the user who runs the procedure.

Examples

```
DBMS_STATS.PUBLISH_PENDING_STATS ('SH', null);
```

162.7.87 PURGE_STATS Procedure

This procedure purges old versions of statistics saved in the dictionary.

To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

Syntax

```
DBMS_STATS.PURGE_STATS(  
    before_timestamp TIMESTAMP WITH TIME ZONE);  
```
Parameters

Table 162-96  PURGE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>before_timestamp</td>
<td>Versions of statistics saved before this timestamp are purged. If NULL, it uses the purging policy used by automatic purge. The automatic purge deletes all history older than the older of (current time - statistics history retention) and (time of recent analyze in the system - 1). The statistics history retention value can be changed using ALTER_STATS_HISTORY_RETENTION Procedure. The default is 31 days.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

Usage Notes

To invoke this procedure you need the ANALYZE ANY privilege and the ANALYZE ANY DICTIONARY privilege.

162.7.88 REMAP_STAT_TABLE Procedure

This procedure remaps the names of objects in the user statistics table. It allows you to import the statistics to objects with same definition but with different names.

Syntax

```sql
DBMS_STATS.REMAP_STAT_TABLE ( 
  ownname    IN    VARCHAR2,  
  stattab    IN    VARCHAR2,  
  src_own    IN    VARCHAR2,  
  src_tab    IN    VARCHAR2,  
  tgt_own    IN    VARCHAR2,  
  tgt_tab    IN    VARCHAR2); 
```

Parameters

Table 162-97  REMAP_STAT_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner of the statistics table. NULL means the current schema.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier</td>
</tr>
<tr>
<td>src_own</td>
<td>Owner of the table to be renamed. This argument cannot be NULL.</td>
</tr>
<tr>
<td>src_tab</td>
<td>Name of the table to be renamed. If NULL, all tables are owned by src_own.</td>
</tr>
</tbody>
</table>
Table 162-97  (Cont.) REMAP_STAT_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgt_own</td>
<td>New name of the owner of the table. The owner name is also updated for the dependent objects such as columns and indexes. Note that an index of src_tab not owned by src_own is not renamed. This argument cannot be NULL.</td>
</tr>
<tr>
<td>tgt_tab</td>
<td>New name of the table. This argument is valid only if src_tab is not NULL.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Insufficient privileges
ORA-20001: Invalid input

Examples

The following statement remaps all objects of sh to shsave in user statistics table sh.ustat:

DBMS_STATS.REMAP_STAT_TABLE ('sh', 'ustat', 'sh', NULL, 'shsave', NULL);

The following statement can be used to import statistics into objects of shsave once the preceding remap procedure is completed:

DBMS_STATS.IMPORT_SCHEMA_STATS ('shsave', 'ustat', statown => 'sh');

The following statement remaps sh.customers to shsave.customers_sav:

DBMS_STATS.REMAP_STAT_TABLE ('sh', 'ustat', 'sh', 'customers', 'shsave', 'customers_sav');

162.7.89 REPORT_ADVISOR_TASK Function

This function reports the results of an Optimizer Statistics Advisor task.

Syntax

```
DBMS_STATS.REPORT_ADVISOR_TASK(
    task_name IN VARCHAR2,
    execution_name IN VARCHAR2 := NULL,
    type IN VARCHAR2 := 'TEXT',
    section IN VARCHAR2 := 'ALL',
    level IN VARCHAR2 := 'TYPICAL')
RETURN CLOB;
```

Parameters

Table 162-98  REPORT_ADVISOR_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
</tbody>
</table>
Table 162-98  (Cont.) REPORT_ADVISOR_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execution_name</td>
<td>A name that qualifies and identifies an advisor task execution. If not specified, the function uses the latest execution of the specified task.</td>
</tr>
<tr>
<td>type</td>
<td>The type of the Optimizer Statistics Advisor report. Possible values are TEXT, HTML, and XML.</td>
</tr>
<tr>
<td>section</td>
<td>A section in the report. Possible values are SUMMARY, FINDINGS, ERRORS, and ALL. You can specify combinations of different values can be using the plus (+) and minus (-) operator, as in 'SUMMARY +FINDINGS +ERRORS', and 'ALL -ERRORS'.</td>
</tr>
<tr>
<td>level</td>
<td>The format of the report. Possible values are BASIC, TYPICAL, ALL, and SHOW_HIDDEN. You can specify SHOW_HIDDEN together with the other three input values, as in 'BASIC +SHOW_HIDDEN' and 'TYPICAL +SHOW_HIDDEN'.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- You can execute this subprogram for AUTO_STATS_ADVISOR_TASK, which is predefined.
- This subprogram executes using invoker’s rights.

The results of performing this task depend on the privileges of the executing user:

- **SYSTEM level**
  Only users with both the ANALYZE ANY and ANALYZE ANY DICTIONARY privileges can perform this task on system-level rules.

- **Operation level**
  The results depend on the following privileges:
  - Users with both the ANALYZE ANY and ANALYZE ANY DICTIONARY privileges can perform this task for all statistics operations.
  - Users with the ANALYZE ANY privilege but not the ANALYZE ANY DICTIONARY privilege can perform this task for statistics operations related to any schema except SYS.
  - Users with the ANALYZE ANY DICTIONARY privilege but not the ANALYZE ANY privilege can perform this task for statistics operations related to their own schema and the SYS schema.
  - Users with neither the ANALYZE ANY nor the ANALYZE ANY DICTIONARY privilege can only perform this operation for statistics operations relating to their own schema.

- **Object level**
Users can perform this task for any object for which they have statistics collection privileges.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: User input errors
- ORA-20012: Optimizer Statistics Advisor errors

Returns

This function returns a CLOB that contains the report.

Examples

(Optional) List and briefly describe the examples for using the API or subprogram here.

Example 162-12  Generating an HTML Report

This example creates a procedure named myrep, and then calls this procedure to generate an HTML report.

```sql
SET ECHO ON
SET FEEDBACK ON
SET SERVEROUTPUT ON
SET TRIMS ON
SET LINESIZE 300

EXECUTE DBMS_OUTPUT.ENABLE (buffer_size => 10000000);

CREATE OR REPLACE PROCEDURE myrep(p_tname VARCHAR2, p_ftype VARCHAR2, which VARCHAR2)
IS
    v_report CLOB := null;
    v_script CLOB := null;
    v_ftype VARCHAR2(400) := p_ftype;
    v_tname VARCHAR2(400) := p_tname;
    v_len NUMBER(10);
    v_ps NUMBER(10) := 10000;
    v_pn NUMBER(10) := 1;
    v_ret VARCHAR2(32767);
BEGIN
    IF which = 'REPORT'
    THEN
        -- generate a report
        v_report := DBMS_STATS.REPORT_ADVISOR_TASK(
            task_name => v_tname,
            type => v_ftype,
            section => 'ALL',
            level => 'ALL');
        v_len := DBMS_LOB.getlength(v_report);
        WHILE (v_pn < v_len)
        LOOP
            DBMS_OUTPUT.PUT_LINE(DBMS_LOB.SUBSTR(v_report, v_ps, v_pn));
            v_pn := v_pn + v_ps;
        END LOOP;
    END IF;
END myrep;
```

Chapter 162  Summary of DBMS_STATS Subprograms
Example 162-13  Generating a Textual Report for AUTO_STATS_ADVISOR_TASK

The following example invokes the myrep procedure created in the preceding example for AUTO_STATS_ADVISOR_TASK:

EXEC myrep('AUTO_STATS_ADVISOR_TASK','TEXT','REPORT');

The following sample output shows part of the report:

GENERAL INFORMATION

Task Name : AUTO_STATS_ADVISOR_TASK
Execution Name : EXEC_97
Created : 07-08-16 10:18:10
Last Modified : 07-11-16 03:02:30

For execution EXEC_97 of task AUTO_STATS_ADVISOR_TASK, the Statistics Advisor has 10 finding(s). The findings are related to the following rules:
COMPLETEAUTOJOB, MAINTAINSTATSHISTORY, USEDEFAULTPREFERENCE,
AVOIDSETPROcedures, USEDEFAULTPARAMS, USEGATHERSCHMASTATS,
AVOIDSTALESTATS,
UNLOCKNONVOLATILETABLE, USEINCREMENTAL, AVOIDANALYZETABLE. Please refer to the finding section for detailed information.
FINDINGS

---

Rule Name: MaintainStatsHistory
Rule Description: Maintain Statistics History

Finding: Statistics history tables are too big.

Recommendation: Check the other findings of this rule, as well as the findings for the rules AvoidFrequentStatsCollection, UseDefaultPreference, UseDefaultParams for possible causes and recommendations.

Rationale: The size of the statistics history table could be big because of violations of other rules.

---

Rule Name: UseDefaultPreference
Rule Description: Use Default Preference for Stats Collection

Finding: Global preference SYS_FLAGS is set to a non-default value '1'.

Recommendation: Set the value of preference SYS_FLAGS to '0'.

Example:
-- Setting preference cascade to default value:
dbms_stats.set_global_prefs('CASCADE', NULL);

Rationale: Preference SYS_FLAGS is for Oracle internal use only, setting it to nondefault value '1' could cause unforeseen consequences.

---

162.7.90 REPORT_COL_USAGE Function

This function reports the recorded column (group) usage information.

Syntax

```sql
DBMS_STATS.REPORT_COL_USAGE (  
    ownname  IN  VARCHAR2,  
    tabname  IN  VARCHAR2)  
RETURN CLOB;
```

---

Chapter 162
Summary of DBMS_STATS Subprograms
Parameters

Table 162-99  REPORT_COL_USAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name. If NULL it reports column usage information for tables in all schemas in the database.</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name. If NULL it reports column usage information for all tables of ownname.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, you need to have the SYSDBA administrative privilege or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

162.7.91 REPORT_GATHER_AUTO_STATS Function

This function runs the auto statistics gathering job in reporting mode. That is, statistics are not actually collected, but all the objects that will be affected when auto statistics gathering is invoked are reported.

Syntax

```sql
DBMS_STATS.REPORT_GATHER_AUTO_STATS (  
    detail_level      VARCHAR2  DEFAULT 'TYPICAL',  
    format            VARCHAR2  DEFAULT 'TEXT')  
RETURN CLOB;
```
### Parameters

Table 162-100  REPORT_GATHER_AUTO_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detail_level</td>
<td>Detail level for the content of the report</td>
</tr>
<tr>
<td></td>
<td>• <strong>BASIC</strong>: The report includes</td>
</tr>
<tr>
<td></td>
<td>- operation ID</td>
</tr>
<tr>
<td></td>
<td>- operation name</td>
</tr>
<tr>
<td></td>
<td>- operation target object</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status (such as: succeeded, failed)</td>
</tr>
<tr>
<td></td>
<td>• <strong>TYPICAL</strong>: In addition to the information provided at level <strong>BASIC</strong>, the report includes individual target objects for which statistics are gathered in this operation. Specifically, with regard to operation related details:</td>
</tr>
<tr>
<td></td>
<td>- total number of target objects</td>
</tr>
<tr>
<td></td>
<td>- total number of successfully completed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of failed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of timed-out objects (applies to only auto statistics gathering)</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- owner and name of each target object</td>
</tr>
<tr>
<td></td>
<td>- target object type (such as: table, index)</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status</td>
</tr>
<tr>
<td></td>
<td>• <strong>ALL</strong>: In addition to the information provided at level <strong>TYPICAL</strong>, the report includes further information on each target object. Specifically, with regard to operation-related details:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- session ID</td>
</tr>
<tr>
<td></td>
<td>- parameter values</td>
</tr>
<tr>
<td></td>
<td>- error message if the operation failed</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- batching details</td>
</tr>
<tr>
<td></td>
<td>- estimated cost</td>
</tr>
<tr>
<td></td>
<td>- rank in the target list</td>
</tr>
<tr>
<td></td>
<td>- columns for which histograms were collected</td>
</tr>
<tr>
<td></td>
<td>- list of collected extended statistics (if any)</td>
</tr>
<tr>
<td></td>
<td>- reason for including the object in the target list</td>
</tr>
<tr>
<td></td>
<td>- additional error details if the task has failed.</td>
</tr>
</tbody>
</table>

Note that several fields (such as job name, estimated task cost) in the report are populated only when an operation is executed concurrently (CONCURRENT preference is turned on).
Table 162-100  (Cont.) REPORT_GATHER_AUTO_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Report format:</td>
</tr>
<tr>
<td></td>
<td>· XML</td>
</tr>
<tr>
<td></td>
<td>· HTML</td>
</tr>
<tr>
<td></td>
<td>· TEXT (Default)</td>
</tr>
</tbody>
</table>

Usage Notes

Only user SYS can run the REPORT_GATHER_AUTO_STATS function.

162.7.92 REPORT_GATHER_DATABASE_STATS Functions

This function runs the GATHER_DATABASE_STATS function in reporting mode.

The database does not collect statistics, but reports all objects that would be affected when invoking GATHER_DATABASE_STATS. The input set of parameters is the same as in GATHER_DATABASE_STATS, with two extra parameters.

Syntax

```sql
DBMS_STATS.REPORT_GATHER_DATABASE_STATS (  
estimate_percent     IN    NUMBER     DEFAULT  
to_estimate_percent_type (  
    GET_PARAM('ESTIMATE_PERCENT')),  
    block_sample         IN    BOOLEAN    DEFAULT FALSE,  
    method_opt           IN    VARCHAR2   DEFAULT  
    GET_PARAM('METHOD_OPT'),  
    degree               IN    NUMBER     DEFAULT TO_DEGREE_TYPE(  
        GET_PARAM('DEGREE')),  
    granularity          IN    VARCHAR2   DEFAULT  
    GET_PARAM('GRANULARITY'),  
    cascade              IN    BOOLEAN    DEFAULT to_cascade_type (  
        GET_PARAM('CASCADE'))),  
    stattab              IN    VARCHAR2   DEFAULT NULL,  
    statid               IN    VARCHAR2   DEFAULT NULL,  
    options              IN    VARCHAR2   DEFAULT 'GATHER',  
    statown              IN    VARCHAR2   DEFAULT NULL,  
    gather_sys           IN    BOOLEAN    DEFAULT TRUE,  
    no_invalidate        IN    BOOLEAN    DEFAULT TO_NO_INVALIDATE_TYPE (  
        GET_PARAM('NO_INVALIDATE')),  
    gather_temp          IN    BOOLEAN    DEFAULT FALSE,  
    gather_fixed         IN    BOOLEAN    DEFAULT FALSE,  
    stattype             IN    VARCHAR2   DEFAULT DATA,  
    obj_filter_list      IN    ObjectTab  DEFAULT NULL,  
    detail_level         IN    VARCHAR2   DEFAULT 'TYPICAL',  
    format               IN    VARCHAR2   DEFAULT 'TEXT')  
RETURN CLOB;  
```
Parameters

Table 162-101 REPORT_GATHER_DATABASE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate_percent</td>
<td>The percentage of rows to use for the sample size. The valid range is between 0.000001 and 100. The null value means to compute. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to enable the database to determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. This parameter is only relevant when estimating statistics.</td>
</tr>
</tbody>
</table>
| method_opt      | Method options. This parameter accepts the following values: FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]. size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}  

- integer: Number of histogram buckets. Must be in the range [1,2048].
- REPEAT: Collects histograms only on the columns that already have histograms.
- AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns.
- SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns. 

The default is FOR ALL COLUMNS SIZE AUTO. The value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. |
| degree          | Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=>NULL, DEGREE=>n, or DEGREE=>DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution. |
Table 162-101  (Cont.) REPORT_GATHER_DATABASE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| granularity | Determines the granularity of statistics to collect. This preference is only relevant for partitioned tables. The following values are valid:  
  • ALL — Gathers all statistics: subpartition, partition, and global.  
  • AUTO — Determines the granularity based on the partitioning type. This is the default value.  
  • DEFAULT — Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. Use GLOBAL AND PARTITION for this functionality.  
  • GLOBAL — Gathers global statistics.  
  • GLOBAL AND PARTITION — Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.  
  • PARTITION — Gathers partition-level statistics.  
  • SUBPARTITION — Gathers subpartition-level statistics. |
| cascade | Gather statistics on the indexes as well. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the database in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. |
| stattab | User statistics table identifier describing where to save the current statistics. The database assumes that the statistics table resides in the same schema as the object being analyzed. Thus, to use this option, one such table must exist in each schema. |
| statid | Identifier (optional) to associate with these statistics within stattab. |
### Table 162-101  (Cont.) REPORT_GATHER_DATABASE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Further specification of which objects to gather statistics for: GATHER: Gathers statistics on all objects in the schema.</td>
</tr>
<tr>
<td></td>
<td>GATHER AUTO: Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are stattab, statid, objlist and statown; all other parameter settings are ignored. Returns a list of processed objects.</td>
</tr>
<tr>
<td></td>
<td>GATHER STALE: Gathers statistics on stale objects as determined by looking at the *_tab_modifications views. Also, return a list of objects found to be stale.</td>
</tr>
<tr>
<td></td>
<td>GATHER EMPTY: Gathers statistics on objects which currently have no statistics. Return a list of objects found to have no statistics.</td>
</tr>
<tr>
<td></td>
<td>LIST AUTO: Returns a list of objects to be processed with GATHER AUTO. LIST STALE: Returns a list of stale objects as determined by looking at the *_tab_modifications views</td>
</tr>
<tr>
<td></td>
<td>LIST EMPTY: Returns a list of objects which currently have no statistics</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab, if different from current schema.</td>
</tr>
<tr>
<td>gather_sys</td>
<td>Gathers statistics on the objects owned by the SYS user.</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>gather_temp</td>
<td>Gathers statistics on global temporary tables when TRUE. The default is FALSE. The temporary table must be created with the ON COMMIT PRESERVE ROWS clause. Also, the statistics collected are based on the data in the session in which this procedure is run, but they are shared across all sessions.</td>
</tr>
<tr>
<td>gather_fixed</td>
<td>Gather statistics on fixed tables when TRUE. The default is FALSE. Only user SYS can collect statistics for fixed tables. The ownname must be SYS or null. When gathering statistics for fixed tables, the database ignores specified values for the following arguments: estimate_percent, block_sample, stattab, statid, statown</td>
</tr>
<tr>
<td></td>
<td>The database does not invalidate the dependent cursors on fixed tables on which stats is collected. This option is meant for internal use only.</td>
</tr>
</tbody>
</table>


Table 162-101  (Cont.) REPORT_GATHER_DATABASE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattype</td>
<td>The type of statistics:</td>
</tr>
<tr>
<td></td>
<td>- DATA — Data statistics only</td>
</tr>
<tr>
<td></td>
<td>- CACHE — Cache statistics only</td>
</tr>
<tr>
<td></td>
<td>- ALL — All statistics</td>
</tr>
<tr>
<td>obj_filter_list</td>
<td>A list of object filters. The attribute values specified in the object filter are case-insensitive unless double-quoted. Wildcards are allowed in the attribute values. When specified, GATHER_DATABASE_STATS gathers statistics only on objects that satisfy at least one object filter in the list as needed. In a single-object filter, the database can specify the constraints on the object attributes. For example, non-NULL values s1 and s2 are specified for attributes a1 and a2 in one object filter. An object o is said to satisfy this object filter when (o.a1 LIKE s1) AND (o.a2 LIKE s2) is true.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>detail_level</td>
<td>The level of detail for the content of the report. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>• <strong>BASIC</strong>: The report includes</td>
</tr>
<tr>
<td></td>
<td>- operation ID</td>
</tr>
<tr>
<td></td>
<td>- operation name</td>
</tr>
<tr>
<td></td>
<td>- operation target object</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status (such as: succeeded, failed)</td>
</tr>
<tr>
<td></td>
<td>• <strong>TYPICAL</strong>: In addition to the information provided at level <strong>BASIC</strong>, the report includes individual target objects for which statistics are gathered in this operation. Specifically, with regard to operation related details:</td>
</tr>
<tr>
<td></td>
<td>- total number of target objects</td>
</tr>
<tr>
<td></td>
<td>- total number of successfully completed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of failed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of timed-out objects (applies to only auto statistics gathering)</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- owner and name of each target object</td>
</tr>
<tr>
<td></td>
<td>- target object type (such as: table, index)</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status</td>
</tr>
<tr>
<td></td>
<td>• <strong>ALL</strong>: In addition to the information provided at level <strong>TYPICAL</strong>, the report includes further information on each target object. Specifically, with regard to operation-related details:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- session ID</td>
</tr>
<tr>
<td></td>
<td>- parameter values</td>
</tr>
<tr>
<td></td>
<td>- error message if the operation failed</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- batching details</td>
</tr>
<tr>
<td></td>
<td>- estimated cost</td>
</tr>
<tr>
<td></td>
<td>- rank in the target list</td>
</tr>
<tr>
<td></td>
<td>- columns for which histograms were collected</td>
</tr>
<tr>
<td></td>
<td>- list of collected extended statistics (if any)</td>
</tr>
<tr>
<td></td>
<td>- additional error details if the task has failed.</td>
</tr>
<tr>
<td></td>
<td>Note that several fields (such as job name, estimated task cost) in the report are populated only when an operation is executed concurrently (CONCURRENT preference is turned on).</td>
</tr>
</tbody>
</table>
Table 162-101  (Cont.) REPORT_GATHER_DATABASE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>The format of the report. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>· XML</td>
</tr>
<tr>
<td></td>
<td>· HTML</td>
</tr>
<tr>
<td></td>
<td>· TEXT (Default)</td>
</tr>
</tbody>
</table>

Return Values

A CLOB object that contains the report

Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Bad input value

Usage Notes

To run this procedure, you need to have the SYSDBA role or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

162.7.93 REPORT_GATHER_DICTIONARY_STATS Functions

This function runs the GATHER_DICTIONARY_STATS procedure in reporting mode.

The database does not collect statistics, but reports all objects affected when invoking GATHER_DICTIONARY_STATS. The detail level for the report is defined by the detail_level input parameter. See the descriptions of detail_level and format in REPORT_GATHER_DICTIONARY_STATS Functions. For all other input parameters, see GATHER_DICTIONARY_STATS Procedure.

Syntax

```sql
DBMS_STATS.REPORT_GATHER_DICTIONARY_STATS (
  comp_id               IN    VARCHAR2   DEFAULT NULL,
  estimate_percent      IN    NUMBER     DEFAULT TO_ESTIMATE_PERCENT_TYPE
  block_sample          IN    BOOLEAN    DEFAULT FALSE,
  method_opt            IN    VARCHAR2   DEFAULT GET_PARAM('METHOD_OPT'),
  degree                IN    NUMBER     DEFAULT TO_DEGREE_TYPE
  GET_PARAM('DEGREE'),
  granularity           IN    VARCHAR2   DEFAULT GET_PARAM('GRANULARITY'),
  cascade               IN    BOOLEAN    DEFAULT TO_CASCADE_TYPE
  GET_PARAM('CASCADE'),
  stattab               IN    VARCHAR2   DEFAULT NULL,
  options               IN    VARCHAR2   DEFAULT 'GATHER AUTO',
  no_invalidate         IN    BOOLEAN    DEFAULT TO_NO_INVALIDATE_TYPE
)
```
Parameters

Table 162-102  REPORT_GATHER_DICTIONARY_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp_id</td>
<td>Component ID of the schema to analyze. NULL results in analyzing schemas of all RDBMS components. Refer refer to the COMP_ID column of the DBA_REGISTRY view. The procedure always gather statistics on SYS and SYSTEM schemas regardless of this argument.</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to sample (NULL means compute). The valid range is between 0.000001 and 100. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to let the database determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Determines whether to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk then the sample values may be somewhat correlated. Only pertinent when performing estimate statistics.</td>
</tr>
<tr>
<td>method_opt</td>
<td>The method options. This parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>· FOR ALL [INDEXED</td>
</tr>
<tr>
<td></td>
<td>size_clause is defined as size_clause := SIZE {integer</td>
</tr>
<tr>
<td></td>
<td>- integer: Number of histogram buckets. Must be in the range [1,2048].</td>
</tr>
<tr>
<td></td>
<td>- REPEAT: Collects histograms only on the columns that already have histograms.</td>
</tr>
<tr>
<td></td>
<td>- AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns.</td>
</tr>
<tr>
<td></td>
<td>- SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns.</td>
</tr>
<tr>
<td></td>
<td>The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
</tbody>
</table>
Table 162-102 (Cont.) REPORT_GATHER_DICTIONARY_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE_TABLE or ALTER_TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</td>
</tr>
</tbody>
</table>
| granularity | Determines the granularity of statistics to collect. This preference is only relevant for partitioned tables. The following values are valid:  
  • ALL — Gathers all statistics: subpartition, partition, and global.  
  • AUTO — Determines the granularity based on the partitioning type. This is the default value.  
  • DEFAULT — Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. Use GLOBAL AND PARTITION for this functionality.  
  • GLOBAL — Gathers global statistics.  
  • GLOBAL AND PARTITION — Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.  
  • PARTITION — Gathers partition-level statistics.  
  • SUBPARTITION — Gathers subpartition-level statistics. |
| cascade | Gathers statistics on indexes also. Index statistics gathering will not be parallelized. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. |
| stattab | User statistics table identifier describing where to save the current statistics. |
### Table 162-102  (Cont.) REPORT_GATHER DICTIONARY_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Further specification of objects for which to gather statistics:</td>
</tr>
<tr>
<td></td>
<td>• 'GATHER' - Gathers statistics on all objects in the schema</td>
</tr>
<tr>
<td></td>
<td>• 'GATHER AUTO' - Gathers all necessary statistics automatically. Oracle</td>
</tr>
<tr>
<td></td>
<td>implicitly determines which objects need new statistics and determines</td>
</tr>
<tr>
<td></td>
<td>how to gather those statistics. When 'GATHER AUTO' is specified, the</td>
</tr>
<tr>
<td></td>
<td>only additional valid parameters are comp_id, stattab, statid and</td>
</tr>
<tr>
<td></td>
<td>statown; all other parameter settings will be ignored. Also, returns a</td>
</tr>
<tr>
<td></td>
<td>list of objects processed.</td>
</tr>
<tr>
<td></td>
<td>• 'GATHER STALE' - Gathers statistics on stale objects as determined</td>
</tr>
<tr>
<td></td>
<td>by looking at the *_tab_modifications views. Also, returns a list of</td>
</tr>
<tr>
<td></td>
<td>objects found to be stale.</td>
</tr>
<tr>
<td></td>
<td>• 'GATHER EMPTY' - Gathers statistics on objects which currently have</td>
</tr>
<tr>
<td></td>
<td>no statistics. Also, returns a list of objects found to have no statistics.</td>
</tr>
<tr>
<td></td>
<td>• 'LIST AUTO' - Returns list of objects to be processed with 'GATHER AUTO'</td>
</tr>
<tr>
<td></td>
<td>• 'LIST STALE' - Returns list of stale objects as determined by looking</td>
</tr>
<tr>
<td></td>
<td>at the *_tab_modifications views</td>
</tr>
<tr>
<td></td>
<td>• 'LIST EMPTY' - Returns list of objects which currently have no statistics</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure</td>
</tr>
<tr>
<td></td>
<td>invalidates the dependent cursors immediately if set to FALSE. Use</td>
</tr>
<tr>
<td></td>
<td>DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate</td>
</tr>
<tr>
<td></td>
<td>dependent cursors. This is the default. The default can be changed using</td>
</tr>
<tr>
<td></td>
<td>the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA</td>
</tr>
<tr>
<td></td>
<td>_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>stattype</td>
<td>The type of statistics:</td>
</tr>
<tr>
<td></td>
<td>• DATA — Data statistics only</td>
</tr>
<tr>
<td></td>
<td>• CACHE — Cache statistics only</td>
</tr>
<tr>
<td></td>
<td>• ALL — All statistics</td>
</tr>
<tr>
<td>obj_filter_list</td>
<td>A list of object filters. When provided, this will gather statistics only</td>
</tr>
<tr>
<td></td>
<td>on objects which satisfy at least one object filter in the list as</td>
</tr>
<tr>
<td></td>
<td>needed. In a single object filter, we can specify the constraints on the</td>
</tr>
<tr>
<td></td>
<td>object attributes. The attribute values specified in the object filter</td>
</tr>
<tr>
<td></td>
<td>are case-insensitive unless double-quoted. Wildcard is allowed in the</td>
</tr>
<tr>
<td></td>
<td>attribute values. Suppose non-NULL values s1, s2, ... are specified for</td>
</tr>
<tr>
<td></td>
<td>attributes a1, a2, ... in one object filter. An object o is said to</td>
</tr>
<tr>
<td></td>
<td>satisfy this object filter if (o.a1 like s1) and (o.a2 like s2) and ...</td>
</tr>
<tr>
<td></td>
<td>is true.</td>
</tr>
<tr>
<td>detail_level</td>
<td>See the description in REPORT_GATHER_DICTIONARY_STATS Functions.</td>
</tr>
<tr>
<td>format</td>
<td>See the description in REPORT_GATHER_DICTIONARY_STATS Functions.</td>
</tr>
</tbody>
</table>

**Return Values**

A CLOB object that contains the report
Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

Exceptions

ORA-20000: Index does not exist or insufficient privileges
ORA-20001: Bad input value
ORA-20002: Bad user statistics table, may need to upgrade it

162.7.94 REPORT_GATHER_FIXED_OBJ_STATS Function

This function runs the GATHER_FIXED_OBJECTS_STATS Procedure in reporting mode.

That is, statistics are not actually collected, but all the objects that will be affected when GATHER_FIXED_OBJ_STATS is invoked are reported. The input set of parameters are exactly the same as in GATHER_FIXED_OBJ_STATS with two extra parameters.

Syntax

```sql
DBMS_STATS.REPORT_GATHER_FIXED_OBJ_STATS (  
  stattab          IN  VARCHAR2 DEFAULT NULL,  
  statid           IN  VARCHAR2 DEFAULT NULL,  
  statown          IN  VARCHAR2 DEFAULT NULL,  
  no_invalidate    IN  BOOLEAN  DEFAULT TO_NO_INVALIDATE_TYPE (  
    GET_PARAM('NO_INVALIDATE'))),  
  detail_level     IN  VARCHAR2   DEFAULT 'TYPICAL',  
  format           IN  VARCHAR2   DEFAULT 'TEXT')  
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to save the current statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier to associate with these statistics within stattab (optional)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different from current schema)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
</tbody>
</table>

162-172
### Table 162-103 (Cont.) REPORT_GATHER_FIXED_OBJ_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detail_level</td>
<td>Detail level for the content of the report</td>
</tr>
<tr>
<td></td>
<td>• <strong>BASIC:</strong> The report includes</td>
</tr>
<tr>
<td></td>
<td>- operation ID</td>
</tr>
<tr>
<td></td>
<td>- operation name</td>
</tr>
<tr>
<td></td>
<td>- operation target object</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status (such as: succeeded, failed)</td>
</tr>
<tr>
<td></td>
<td>• <strong>TYPICAL:</strong> In addition to the information provided at level <strong>BASIC,</strong> the report includes individual target objects for which statistics are gathered in this operation. Specifically, with regard to operation-related details:</td>
</tr>
<tr>
<td></td>
<td>- total number of target objects</td>
</tr>
<tr>
<td></td>
<td>- total number of successfully completed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of failed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of timed-out objects (applies to only auto statistics gathering)</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- owner and name of each target object</td>
</tr>
<tr>
<td></td>
<td>- target object type (such as: table, index)</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status</td>
</tr>
<tr>
<td></td>
<td>• <strong>ALL:</strong> In addition to the information provided at level <strong>TYPICAL,</strong> the report includes further information on each target object. Specifically, with regard to operation-related details:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- session ID</td>
</tr>
<tr>
<td></td>
<td>- parameter values</td>
</tr>
<tr>
<td></td>
<td>- error message if the operation failed</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- batching details</td>
</tr>
<tr>
<td></td>
<td>- estimated cost</td>
</tr>
<tr>
<td></td>
<td>- rank in the target list</td>
</tr>
<tr>
<td></td>
<td>- columns for which histograms were collected</td>
</tr>
<tr>
<td></td>
<td>- list of collected extended statistics (if any)</td>
</tr>
<tr>
<td></td>
<td>- additional error details if the task has failed</td>
</tr>
</tbody>
</table>

Note that several fields (such as job name, estimated task cost) in the report are populated only when an operation is executed concurrently (**CONCURRENT** preference is turned on).
Table 162-103 (Cont.) REPORT_GATHER_FIXED_OBJ_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Report format:</td>
</tr>
<tr>
<td></td>
<td>· XML</td>
</tr>
<tr>
<td></td>
<td>· HTML</td>
</tr>
<tr>
<td></td>
<td>· TEXT (Default)</td>
</tr>
</tbody>
</table>

Return Values

A CLOB object that contains the report

Usage Notes

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.

Exceptions

ORA-20000: Insufficient privileges
ORA-20001: Bad input value
ORA-20002: Bad user statistics table, may need to upgrade it

Related Topics

- GATHER_FIXED_OBJECTS_STATS Procedure
  This procedure gathers statistics for all fixed objects (dynamic performance tables).

162.7.95 REPORT_GATHER_SCHEMA_STATS Functions

This function runs the GATHER_SCHEMA_STATS procedure in reporting mode.

The database does not actually gather statistics, but reports all objects that would be affected when invoking GATHER_SCHEMA_STATS. The input set of parameters is exactly the same as in GATHER_SCHEMA_STATS, with two extra parameters.

Syntax

```
DBMS_STATS.REPORT_GATHER_SCHEMA_STATS (              
  ownname           IN   VARCHAR2,                   
  estimate_percent  IN   NUMBER      DEFAULT TO_ESTIMATE_PERCENT_TYPE (   
    GET_PARAM ('ESTIMATE_PERCENT'))),                
  block_sample      IN   BOOLEAN     DEFAULT FALSE,   
  method_opt        IN   VARCHAR2    DEFAULT GET_PARAM ('METHOD_OPT'),     
  degree            IN   NUMBER      DEFAULT TO_DEGREE_TYPE (   
    GET_PARAM('DEGREE')),                            
  granularity       IN   VARCHAR2    DEFAULT GET_PARAM('GRANULARITY'),     
  cascade           IN   BOOLEAN     DEFAULT GET_PARAM('CASCADE'),         
  stattab           IN   VARCHAR2    DEFAULT NULL,                      
  statid            IN   VARCHAR2    DEFAULT NULL,                      
  options           IN   VARCHAR2    DEFAULT 'GATHER',                   
)                                                       
```
Parameters

Table 162-104 REPORT_GATHER_SCHEMA_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema to analyze (NULL means current schema)</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.</td>
</tr>
<tr>
<td>method_opt</td>
<td>Accepts: FOR ALL [INDEXED</td>
</tr>
<tr>
<td></td>
<td>- integer: Number of histogram buckets. Must be in the range [1,2048].</td>
</tr>
<tr>
<td></td>
<td>- REPEAT: Collects histograms only on the columns that already have histograms</td>
</tr>
<tr>
<td></td>
<td>- AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns.</td>
</tr>
<tr>
<td></td>
<td>- SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns.</td>
</tr>
</tbody>
</table>

The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</td>
</tr>
</tbody>
</table>
| granularity | Granularity of statistics to collect (only pertinent if the table is partitioned).  
- 'ALL' - Gathers all (subpartition, partition, and global) statistics  
- 'AUTO' - Determines the granularity based on the partitioning type. This is the default value.  
- 'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.  
- 'GLOBAL' - Gathers global statistics  
- 'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.  
- 'PARTITION' - Gathers partition-level statistics  
- 'SUBPARTITION' - Gathers subpartition-level statistics. |
| cascade | Gather statistics on the indexes as well. Using this option is equivalent to running the GATHER_INDEX_STATS Procedure on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant DBMS_STATS.AUTO_CASCADE to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. |
| stattab | User statistics table identifier describing where to save the current statistics |
| statid | Identifier (optional) to associate with these statistics within stattab |
### Table 162-104 (Cont.) REPORT_GATHER_SCHEMA_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Further specification of which objects to gather statistics for:</td>
</tr>
<tr>
<td></td>
<td>GATHER: Gathers statistics on all objects in the schema.</td>
</tr>
<tr>
<td></td>
<td>GATHER AUTO: Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When GATHER AUTO is specified, the only additional valid parameters are ownname, stattab, statid, objlist and statown; all other parameter settings are ignored. Returns a list of processed objects.</td>
</tr>
<tr>
<td></td>
<td>GATHER STALE: Gathers statistics on stale objects as determined by looking at the *_tab_modifications views. Also, return a list of objects found to be stale.</td>
</tr>
<tr>
<td></td>
<td>GATHER EMPTY: Gathers statistics on objects which currently have no statistics. Also, return a list of objects found to have no statistics.</td>
</tr>
<tr>
<td></td>
<td>LIST AUTO: Returns a list of objects to be processed with GATHER AUTO.</td>
</tr>
<tr>
<td></td>
<td>LIST STALE: Returns list of stale objects as determined by looking at the *_tab_modifications views.</td>
</tr>
<tr>
<td></td>
<td>LIST EMPTY: Returns list of objects which currently have no statistics.</td>
</tr>
<tr>
<td>objlist</td>
<td>List of objects found to be stale or empty</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>Gather statistics on objects even if they are locked</td>
</tr>
<tr>
<td>obj_filter_list</td>
<td>A list of object filters. When provided, GATHER_SCHEMA_STATS will gather statistics only on objects which satisfy at least one object filter in the list as needed. In a single object filter, we can specify the constraints on the object attributes. The attribute values specified in the object filter are case-insensitive unless double-quoted. Wildcard is allowed in the attribute values. Suppose non-NULL values s1, s2, ... are specified for attributes a1, a2, ... in one object filter. An object o is said to satisfy this object filter if (o.a1 like s1) and (o.a2 like s2) and ... is true. See Applying an Object Filter List.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>detail_level</td>
<td>Detail level for the content of the report</td>
</tr>
<tr>
<td></td>
<td>• <strong>BASIC</strong>: The report includes</td>
</tr>
<tr>
<td></td>
<td>- operation ID</td>
</tr>
<tr>
<td></td>
<td>- operation name</td>
</tr>
<tr>
<td></td>
<td>- operation target object</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status (such as: succeeded, failed)</td>
</tr>
<tr>
<td></td>
<td>• <strong>TYPICAL</strong>: In addition to the information provided at level <strong>BASIC</strong>, the report includes individual target objects for which statistics are gathered in this operation. Specifically, with regard to operation related details:</td>
</tr>
<tr>
<td></td>
<td>- total number of target objects</td>
</tr>
<tr>
<td></td>
<td>- total number of successfully completed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of failed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of timed-out objects (applies to only auto statistics gathering)</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- owner and name of each target object</td>
</tr>
<tr>
<td></td>
<td>- target object type (such as: table, index)</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status</td>
</tr>
<tr>
<td></td>
<td>• <strong>ALL</strong>: In addition to the information provided at level <strong>TYPICAL</strong>, the report includes further information on each target object. Specifically, with regard to operation-related details:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- session ID</td>
</tr>
<tr>
<td></td>
<td>- parameter values</td>
</tr>
<tr>
<td></td>
<td>- error message if the operation failed</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- batching details</td>
</tr>
<tr>
<td></td>
<td>- estimated cost</td>
</tr>
<tr>
<td></td>
<td>- rank in the target list</td>
</tr>
<tr>
<td></td>
<td>- columns for which histograms were collected</td>
</tr>
<tr>
<td></td>
<td>- list of collected extended statistics (if any)</td>
</tr>
<tr>
<td></td>
<td>- additional error details if the task has failed.</td>
</tr>
<tr>
<td></td>
<td>Note that several fields (such as job name, estimated task cost) in the report are populated only when an operation is executed concurrently (CONCURRENT preference is turned on).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>format</th>
<th>Report format:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• XML</td>
</tr>
<tr>
<td></td>
<td>• HTML</td>
</tr>
<tr>
<td></td>
<td>• TEXT (Default)</td>
</tr>
</tbody>
</table>
Usage Notes

To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.

Exceptions

ORA-20000: Schema does not exist or insufficient privileges

ORA-20001: Bad input value

Examples

Applying an Object Filter List

The following example specifies that any table with a "T" prefix in the `SAMPLE` schema or any table in the `SYS` schema, if stale, will have statistics gathered upon it.

```sql
DECLARE
    filter_lst  DBMS_STATS.OBJECTTAB := DBMS_STATS.OBJECTTAB();
BEGIN
    filter_lst.extend(2);
    filter_lst(1).ownname := 'SAMPLE';
    filter_lst(1).objname := 'T%';
    filter_lst(2).ownname := 'SYS';
    DBMS_STATS.GATHER_SCHEMA_STATS(NULL, obj_filter_list => filter_lst,
                                    options => 'GATHER STALE');
END;
```

162.7.96 REPORT_GATHER_TABLE_STATS Function

This procedure runs the `GATHER_TABLE_STATS` Procedure in reporting mode. That is, statistics are not actually collected, but all the objects that will be affected when `GATHER_TABLE_STATS` is invoked are reported.

Syntax

```sql
DBMS_STATS.REPORT_GATHER_TABLE_STATS (    ownname          VARCHAR2,    tabname          VARCHAR2,    partname         VARCHAR2 DEFAULT NULL,    estimate_percent NUMBER   DEFAULT to_estimate_percent_type     (get_param('ESTIMATE_PERCENT')),    block_sample     BOOLEAN  DEFAULT FALSE,    method_opt       VARCHAR2 DEFAULT? get_param('METHOD_OPT'),    degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),    granularity      VARCHAR2 DEFAULT? get_param('GRANULARITY'),    cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),    stattab          VARCHAR2 DEFAULT NULL,    statid           VARCHAR2 DEFAULT NULL,    statown          VARCHAR2 DEFAULT NULL,    no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (get_param('NO_INVALIDATE')),    stattype         VARCHAR2 DEFAULT 'DATA',    force            BOOLEAN  DEFAULT FALSE,    detail_level     VARCHAR2 DEFAULT 'TYPICAL',    format           VARCHAR2 DEFAULT NULL);
```

Chapter 162

Summary of DBMS_STATS Subprograms

162-179
Parameters

### Table 162-105 REPORT_GATHER_TABLE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of table to analyze</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of table</td>
</tr>
<tr>
<td>partname</td>
<td>Name of partition</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate (NULL means compute) The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>block_sample</td>
<td>Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.</td>
</tr>
<tr>
<td>method_opt</td>
<td>Accepts either of the following options, or both in combination:</td>
</tr>
<tr>
<td></td>
<td>• FOR ALL [INDEXED</td>
</tr>
<tr>
<td></td>
<td>• FOR COLUMNS [column_clause] [size_clause]</td>
</tr>
<tr>
<td></td>
<td>size_clause is defined as size_clause := SIZE {integer</td>
</tr>
<tr>
<td></td>
<td>column_clause is defined as column_clause := column_name</td>
</tr>
<tr>
<td></td>
<td>- integer: Number of histogram buckets. Must be in the range [1,2048].</td>
</tr>
<tr>
<td></td>
<td>- REPEAT: Collects histograms only on the columns that already have histograms</td>
</tr>
<tr>
<td></td>
<td>- AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns.</td>
</tr>
<tr>
<td></td>
<td>- SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns.</td>
</tr>
<tr>
<td></td>
<td>- column_name: Name of a column</td>
</tr>
<tr>
<td></td>
<td>- extension: can be either a column group in the format of</td>
</tr>
<tr>
<td></td>
<td>(column_name, Column_name [, ...]) or an expression</td>
</tr>
</tbody>
</table>

The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.
Table 162-105  (Cont.) REPORT_GATHER_TABLE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>Degree of parallelism. The default for degree is NULL. The default value can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</td>
</tr>
<tr>
<td>granularity</td>
<td>Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - Gathers all (subpartition, partition, and global) statistics 'APPROX_GLOBAL AND PARTITION' - similar to 'GLOBAL AND PARTITION' but in this case the global statistics are aggregated from partition level statistics. This option will aggregate all statistics except the number of distinct values for columns and number of distinct keys of indexes. The existing histograms of the columns at the table level are also aggregated. The aggregation will use only partitions with statistics, so to get accurate global statistics, users should make sure to have statistics for all partitions. Global statistics are gathered if partname is NULL or if the aggregation cannot be performed (for example, if statistics for one of the partitions is missing). 'AUTO' - Determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - Gathers global statistics 'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION' - Gathers partition-level statistics 'SUBPARTITION' - Gathers subpartition-level statistics.</td>
</tr>
</tbody>
</table>
### Table 162-105  (Cont.) REPORT_GATHER_TABLE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cascade</td>
<td>Gathers statistics on the indexes for this table. Using this option is equivalent to running the <code>GATHER_INDEX_STATS Procedure</code> on each of the table's indexes. Use the constant <code>DBMS_STATS.AUTO_CASCADE</code> to have Oracle determine whether index statistics are to be collected or not. This is the default. The default value can be changed using the <code>SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure</code>.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to save the current statistics</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <code>SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure</code>.</td>
</tr>
<tr>
<td>stattype</td>
<td>Statistics type. The only value allowed is <code>DATA</code>.</td>
</tr>
<tr>
<td>force</td>
<td>Gather statistics of table even if it is locked</td>
</tr>
</tbody>
</table>
Table 162-105  (Cont.) REPORT_GATHER_TABLE_STATS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detail_level</td>
<td>Detail level for the content of the report</td>
</tr>
<tr>
<td></td>
<td>• BASIC: The report includes</td>
</tr>
<tr>
<td></td>
<td>- operation ID</td>
</tr>
<tr>
<td></td>
<td>- operation name</td>
</tr>
<tr>
<td></td>
<td>- operation target object</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status (such as: succeeded, failed)</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: In addition to the information provided at level BASIC, the report includes individual target objects for which statistics are gathered in this operation. Specifically, with regard to operation related details:</td>
</tr>
<tr>
<td></td>
<td>- total number of target objects</td>
</tr>
<tr>
<td></td>
<td>- total number of successfully completed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of failed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of timed-out objects (applies to only auto statistics gathering)</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- owner and name of each target object</td>
</tr>
<tr>
<td></td>
<td>- target object type (such as: table, index)</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status</td>
</tr>
<tr>
<td></td>
<td>• ALL: In addition to the information provided at level TYPICAL, the report includes further information on each target object. Specifically, with regard to operation-related details:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- session ID</td>
</tr>
<tr>
<td></td>
<td>- parameter values</td>
</tr>
<tr>
<td></td>
<td>- error message if the operation failed</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- batching details</td>
</tr>
<tr>
<td></td>
<td>- estimated cost</td>
</tr>
<tr>
<td></td>
<td>- rank in the target list</td>
</tr>
<tr>
<td></td>
<td>- columns for which histograms were collected</td>
</tr>
<tr>
<td></td>
<td>- list of collected extended statistics (if any)</td>
</tr>
<tr>
<td></td>
<td>- additional error details if the task has failed.</td>
</tr>
<tr>
<td></td>
<td>Note that several fields (such as job name, estimated task cost) in the report are populated only when an operation is executed concurrently (CONCURRENT preference is turned on).</td>
</tr>
<tr>
<td>format</td>
<td>Report format:</td>
</tr>
<tr>
<td></td>
<td>• XML</td>
</tr>
<tr>
<td></td>
<td>• HTML</td>
</tr>
<tr>
<td></td>
<td>• TEXT (Default)</td>
</tr>
</tbody>
</table>
Return Values

A CLOB object that contains the report

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Related Topics

- GATHER_TABLE_STATS Procedure
  This procedure gathers table, column, and index statistics. It attempts to parallelize as much work as possible, but there are some restrictions, which are described in the individual parameters.

162.7.97 REPORT_SINGLE_STATS_OPERATION Function

This function generates a report for the provided operation optionally in a particular pluggable database (PDB) in a multitenant environment.

Syntax

```sql
DBMS_STATS.REPORT_SINGLE_STATS_OPERATIONS (opid              NUMBER,
detail_level      VARCHAR2  DEFAULT 'TYPICAL',
format            VARCHAR2  DEFAULT 'TEXT'
container_id      NUMBER    DEFAULT NULL)
RETURN CLOB;
```

Parameters

Table 162-106  REPORT_SINGLE_STATS_OPERATION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opid</td>
<td>Operation ID</td>
</tr>
</tbody>
</table>
Table 162-106 (Cont.) REPORT_SINGLE_STATS_OPERATION Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detail_level</td>
<td>Detail level for the content of the report</td>
</tr>
<tr>
<td></td>
<td>• <strong>BASIC:</strong> The report includes</td>
</tr>
<tr>
<td></td>
<td>- operation ID</td>
</tr>
<tr>
<td></td>
<td>- operation name</td>
</tr>
<tr>
<td></td>
<td>- operation target object</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status (such as: succeeded, failed)</td>
</tr>
<tr>
<td></td>
<td>• <strong>TYPICAL:</strong> In addition to the information provided at level <strong>BASIC,</strong> the report includes individual target objects for which statistics are gathered in this operation. Specifically, with regard to operation related details:</td>
</tr>
<tr>
<td></td>
<td>- total number of target objects</td>
</tr>
<tr>
<td></td>
<td>- total number of successfully completed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of failed objects</td>
</tr>
<tr>
<td></td>
<td>- total number of timed-out objects (applies to only auto statistics gathering)</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- owner and name of each target object</td>
</tr>
<tr>
<td></td>
<td>- target object type (such as: table, index)</td>
</tr>
<tr>
<td></td>
<td>- start time</td>
</tr>
<tr>
<td></td>
<td>- end time</td>
</tr>
<tr>
<td></td>
<td>- completion status</td>
</tr>
<tr>
<td></td>
<td>• <strong>ALL:</strong> In addition to the information provided at level <strong>TYPICAL,</strong> the report includes further information on each target object. Specifically, with regard to operation-related details:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- session ID</td>
</tr>
<tr>
<td></td>
<td>- parameter values</td>
</tr>
<tr>
<td></td>
<td>- error message if the operation failed</td>
</tr>
<tr>
<td></td>
<td>With regard to target objects:</td>
</tr>
<tr>
<td></td>
<td>- job name</td>
</tr>
<tr>
<td></td>
<td>- batching details</td>
</tr>
<tr>
<td></td>
<td>- estimated cost</td>
</tr>
<tr>
<td></td>
<td>- rank in the target list</td>
</tr>
<tr>
<td></td>
<td>- columns for which histograms were collected</td>
</tr>
<tr>
<td></td>
<td>- list of collected extended statistics (if any)</td>
</tr>
<tr>
<td></td>
<td>- reason for including the object in the target list (applies to only automatic statistics gathering operation tasks)</td>
</tr>
<tr>
<td></td>
<td>- additional error details if the task has failed</td>
</tr>
</tbody>
</table>

Note that several fields (such as job name, estimated task cost) in the report are populated only when an operation is executed concurrently (**CONCURRENT** preference is turned on).
### Table 162-106 (Cont.) REPORT_SINGLE_STATS_OPERATION Function

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Report format:</td>
</tr>
<tr>
<td></td>
<td>• XML</td>
</tr>
<tr>
<td></td>
<td>• HTML</td>
</tr>
<tr>
<td></td>
<td>• TEXT (Default)</td>
</tr>
<tr>
<td>container_id</td>
<td>ID of the pluggable database (PDB) on which this operation was performed. Note that in a multitenant environment, operation ID does not uniquely identify an operation. That is, different operations from distinct PDBs may have the same operation ID. Hence, in a multitenant environment, if a PDB ID is not provided, then the report may contain multiple operations. In a typical (non-CDB) database environment, operation ID is unique to each operation.</td>
</tr>
</tbody>
</table>

### Usage Notes

To invoke this procedure you need the `ANALYZE ANY` privilege and the `ANALYZE ANY DICTIONARY` privilege.

### 162.7.98 REPORT_STATS_OPERATIONS Function

This function generates a report of all statistics operations that take place between two timestamps which may or may not have been provided.

It allows the scope of the report to be narrowed down so that report will include only auto statistics gathering runs. Furthermore, in a multitenant environment, users may optionally provide a set of pluggable database (PDB) IDs so that only statistics operations from the specified pluggable databases will be reported.

### Syntax

```sql
DBMS_STATS.REPORT_STATS_OPERATIONS ( 
    detail_level       VARCHAR2 DEFAULT 'TYPICAL',
    format             VARCHAR2 DEFAULT 'TEXT',
    latestN            NUMBER DEFAULT NULL,
    since              TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    until              TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    auto_only          BOOLEAN DEFAULT FALSE,
    container_ids      DBMS_UTILITY.NUMBER_ARRAY DEFAULT DBMS_STATS.NULL_NUMTAB) 
RETURN CLOB;
```
### Parameters

**Table 162-107  REPORT_STATS_OPERATIONS Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **detail_level** | Detail level for the content of the report  
|             | - **BASIC**: The report includes  
|             |  - operation ID  
|             |  - operation name  
|             |  - operation target object  
|             |  - start time  
|             |  - end time  
|             |  - completion status (such as: succeeded, failed)  
|             | - **TYPICAL**: In addition to the information provided at level **BASIC**, the report includes individual target objects for which statistics are gathered in this operation. Specifically, with regard to operation related details:  
|             |  - total number of target objects  
|             |  - total number of successfully completed objects  
|             |  - total number of failed objects  
|             |  - total number of timed-out objects (applies to only auto statistics gathering)  
|             | - **ALL**: In addition to the information provided at level **TYPICAL**, the report includes further information on each target object. Specifically, with regard to operation-related details:  
|             |  - job name (if the operation was run in a job)  
|             |  - session ID  
|             |  - parameter values  
|             |  - additional error details if the operation has failed  
| **format** | Report format:  
|             | - XML  
|             | - HTML  
|             | - TEXT (Default)  
| **latestN** | Restricts the report to contain only the latest N operations that took place between the provided time points (since and until). The default value is **NULL**, meaning that all qualifying operations will be reported.  
| **since** | The report will include only statistics operations that started after this timestamp.  
| **until** | The report will include only statistics operations that before after this timestamp.  
| **auto_only** | When **TRUE**, the report will contain only auto statistics gathering job runs.  
| **container_ids** | A multitenant environment contains one or more pluggable databases (PDBs). **container_ids** represents a set of PDB IDs so that only statistics operations from the specified PDBs are reported (applies to only multitenant environments).
Usage Notes

To invoke this procedure you need the ANALYZE ANY privilege and the ANALYZE ANY DICTIONARY privilege.

Examples

Note that the type for container_ids input parameter is DBMS_UTILITY.NUMBER_ARRAY which is an associative PL/SQL array collection. Although associative array type allows for more flexible values table-like organization of entries, this function treats container_ids as a regular table collection with the first ID located at index 1 and the last ID located at index container_ids.count without any empty array slot left between any two IDs. An example for 3 container ids is provided.

```plsql
DECLARE
    conid_tab DBMS_UTILITY.NUMBER_ARRAY;
    report clob;
BEGIN
    conid_tab(1) := 124;
    conid_tab(2) := 63;
    conid_tab(3) := 98;
    report := DBMS_STATS.REPORT_STATS_OPERATIONS (container_ids => conid_tab);
END;
```

162.7.99 RESET_ADVISOR_TASK Procedure

This procedure resets an Optimizer Statistics Advisor task execution to its initial state. Only reset a task that is not currently executing.

Syntax

```plsql
DBMS_STATS.RESET_ADVISOR_TASK (  
    task_name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.

Exceptions

- ORA-20000: Insufficient privileges
• ORA-20001: Invalid input values
• ORA-20012: Optimizer Statistics Advisor errors

162.7.100 RESET_COL_USAGE Procedure

This procedure deletes the recorded column (group) usage information.

This procedure should only be used in very rare cases when the seed column usage needs to be initialized.

Syntax

```sql
DBMS_STATS.RESET_COL_USAGE (  
    ownname    IN    VARCHAR2,  
    tabname    IN    VARCHAR2);
```

Parameters

Table 162-109  RESET_COL_USAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name. If NULL it deletes column usage information for tables in all schemas in the database.</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name. If NULL it deletes column usage information for all tables of ownname. If both the owner and tabname is NULL, the seed column usage is stopped if applicable. See :SEED_COL_USAGE Procedure for more information.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, you need to have the SYSDBA administrative privilege, or both the ANALYZE ANY DICTIONARY and the ANALYZE ANY system privileges.

162.7.101 RESET_GLOBAL_PREF_DEFAULTS Procedure

This procedure sets global preference, such as CASCADE, ESTIMATE_PERCENT and GRANULARITY, to default values.

This reverses the global preferences set by the SET_GLOBAL_PREFS Procedure.

Syntax

```sql
DBMS_STATS.RESET_GLOBAL_PREF_DEFAULTS;
```

Usage Notes

To invoke this procedure you need the ANALYZE ANY privilege and the ANALYZE ANY DICTIONARY privilege.
162.7.102 RESET_PARAM_DEFAULTS Procedure

This deprecated procedure resets the default values of all parameters to Oracle recommended values.

**Note:**
This subprogram has been replaced by improved technology and is maintained only for purposes of backward compatibility. Instead of this procedure, use the `RESET_GLOBAL_PREF_DEFAULTS` Procedure.

See also: [DBMS_STATS Deprecated Subprograms](#)

**Syntax**

```
DBMS_STATS.RESET_PARAM_DEFAULTS;
```

162.7.103 RESTORE_DATABASE_STATS Procedure

This procedure restores statistics of all tables of the database as of a specified timestamp (`as_of_timestamp`).

**Syntax**

```
DBMS_STATS.RESTORE_DATABASE_STATS(
    as_of_timestamp        TIMESTAMP WITH TIME ZONE,
    force                  BOOLEAN DEFAULT FALSE,
    no_invalidate          BOOLEAN DEFAULT to_no_invalidate_type
    (GET_PARAM('NO_INVALIDATE')));
```

**Parameters**

**Table 162-110**  
RESTORE_DATABASE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_of_timestamp</td>
<td>The timestamp to which to restore statistics</td>
</tr>
<tr>
<td>force</td>
<td>Restores statistics even if their statistics are locked</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <code>SET_DATABASE_PREFS Procedure</code>, <code>SET_GLOBAL_PREFS Procedure</code>, <code>SET_SCHEMA_PREFS Procedure</code> and <code>SET_TABLE_PREFS Procedure</code>.</td>
</tr>
</tbody>
</table>

**Exceptions**

ORA-20000: **Object does not exist or insufficient privileges**
162.7.104 RESTORE_DICTIONARY_STATS Procedure

This procedure restores statistics of all dictionary tables (tables of 'SYS', 'SYSTEM' and RDBMS component schemas) as of a specified timestamp (as_of_timestamp).

Syntax

```
DBMS_STATS.RESTORE_DICTIONARY_STATS(
    as_of_timestamp        TIMESTAMP WITH TIME ZONE,
    force                  BOOLEAN DEFAULT FALSE,
    no_invalidate          BOOLEAN DEFAULT to_no_invalidate_type
                   (GET_PARAM('NO_INVALIDATE')));
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_of_timestamp</td>
<td>Timestamp to which to restore statistics</td>
</tr>
<tr>
<td>force</td>
<td>Restores statistics even if their statistics are locked</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

Exceptions

- ORA-20000: Object does not exist or insufficient privileges
- ORA-20001: Invalid or inconsistent values
- ORA-20006: Unable to restore statistics, statistics history not available

162.7.105 RESTORE_FIXED_OBJECTS_STATS Procedure

This procedure restores statistics of all fixed tables as of a specified timestamp (as_of_timestamp).

Syntax

```
DBMS_STATS.RESTORE_FIXED_OBJECTS_STATS(
    as_of_timestamp        TIMESTAMP WITH TIME ZONE,
    force                  BOOLEAN DEFAULT FALSE,
```


Parameters

Table 162-112 RESTORE_FIXED_OBJECTS_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_of_timestamp</td>
<td>The timestamp to which to restore statistics</td>
</tr>
<tr>
<td>force</td>
<td>Restores statistics even if their statistics are locked</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, you must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid or inconsistent values
ORA-20006: Unable to restore statistics, statistics history not available

162.7.106 RESTORE_SCHEMA_STATS Procedure

This procedure restores statistics of all tables of a schema as of a specified timestamp (as_of_timestamp).

Syntax

```sql
DBMS_STATS.RESTORE_SCHEMA_STATS(
    ownname VARCHAR2,
    as_of_timestamp TIMESTAMP WITH TIME ZONE,
    force BOOLEAN DEFAULT FALSE,
    no_invalidate BOOLEAN DEFAULT to_no_invalidate_type
);```

Parameters

Table 162-113 RESTORE_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Schema of the tables for which the statistics are to be restored</td>
</tr>
</tbody>
</table>
Table 162-113  (Cont.) RESTORE_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_of_timestamp</td>
<td>The timestamp to which to restore statistics</td>
</tr>
<tr>
<td>force</td>
<td>Restores statistics even if their statistics are locked</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid or inconsistent values
ORA-20006: Unable to restore statistics, statistics history not available

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

162.7.107 RESTORE_SYSTEM_STATS Procedure

This procedure restores system statistics as of a specified timestamp (as_of_timestamp).

Syntax

```
DBMS_STATS.RESTORE_SCHEMA_STATS(
   as_of_timestamp TIMEZONE);
```

Parameters

Table 162-114  RESTORE_SYSTEM_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_of_timestamp</td>
<td>The timestamp to which to restore statistics</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid or inconsistent values
ORA-20006: Unable to restore statistics, statistics history not available
Usage Notes

To run this procedure, you need the GATHER_SYSTEM_STATISTICS role.

162.7.108 RESTORE_TABLE_STATS Procedure

This procedure restores statistics of a table as of a specified timestamp (as_of_timestamp). It also restores statistics of associated indexes and columns.

If the table statistics were locked at the specified timestamp the procedure will lock the statistics. The procedure will not restore user defined statistics.

Syntax

```sql
DBMS_STATS.RESTORE_TABLE_STATS (    ownname                   VARCHAR2,    tabname                   VARCHAR2,    as_of_timestamp           TIMESTAMP WITH TIME ZONE,    restore_cluster_index     BOOLEAN DEFAULT FALSE,    force                     BOOLEAN DEFAULT FALSE,    no_invalidate             BOOLEAN DEFAULT to_no_invalidate_type

(GET_PARAM('NO_INVALIDATE')));
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>The schema of the table for which the statistics are to be restored</td>
</tr>
<tr>
<td>tabname</td>
<td>The table name</td>
</tr>
<tr>
<td>as_of_timestamp</td>
<td>The timestamp to which to restore statistics</td>
</tr>
<tr>
<td>restore_cluster_index</td>
<td>If the table is part of a cluster, restore statistics of the cluster index if set to TRUE</td>
</tr>
<tr>
<td>force</td>
<td>Restores statistics even if the table statistics are locked. If the table statistics were not locked at the specified timestamp, it unlocks the statistics.</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

ORA-20006: Unable to restore statistics, statistics history not available
Usage Notes

To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY`
privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need
the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.

162.7.109 RESUME_ADVISOR_TASK Procedure

This procedure resumes an interrupted task. It only resumes the execution that was most
recently interrupted.

Syntax

```sql
DBMS_STATS.RESUME_ADVISOR_TASK (  
    task_name IN VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the `ADVISOR` privilege.
- You must be the owner of the task.
- This subprogram executes using invoker’s rights.

Consider a case in which a task is executed by one user, interrupted, and then resumed by a
different user. In this case, Optimizer Statistics Advisor bases its checks of the resumed
execution on the privilege of the user who resumed the task.

Exceptions

- `ORA-20000`: Insufficient privileges
- `ORA-20001`: Invalid input values
- `ORA-20012`: Optimizer Statistics Advisor errors

Example 162-14  Resuming an Interrupted Task

In this example, you start a SQL*Plus session, and then create and execute an advisor task
named `my_task`:

```sql
DECLARE
    v_tname   VARCHAR2(128) := 'my_task';
    v_ename   VARCHAR2(128) := NULL;
BEGIN
    -- create a task
    v_tname := DBMS_STATS.CREATE_ADVISOR_TASK(v_tname);
```

162-195
In a separate terminal, you start a second SQL*Plus session, and then execute the following program:

```
XEC DBMS_STATS.INTERRUPT_ADVISOR_TASK('my_task');
```

The first session returns an **ORA-13632** to indicate the cancelation of the task:

```
ORA-13638: The user interrupted the current operation.
```

In the second SQL*Plus session, you resume the task execution as follows:

```
XEC DBMS_STATS.RESUME_ADVISOR_TASK('my_task');
```

### 162.7.110 SCRIPT_ADVISOR_TASK Function

Retrieves the script that implements the recommended actions for the problems found by Optimizer Statistics Advisor.

The generated script contains PL/SQL statements that you can choose to execute. Preceding the commands for each action are comments that list the potential side effects. You can review the comments, and choose to execute only the desired sections.

**Syntax**

```
DBMS_STATS.SCRIPT_ADVISOR_TASK (  
  task_name          IN   VARCHAR2,  
  execution_name     IN   VARCHAR2    := NULL,  
  dir_name           IN   VARCHAR2    := NULL,  
  level              IN   VARCHAR2    := 'TYPICAL')  
RETURN CLOB;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td>execution_name</td>
<td>A name that qualifies and identifies an advisor execution. If not specified, if the specified execution conflicts with the name of an existing execution, then the function returns an error.</td>
</tr>
</tbody>
</table>

```
Table 162-117  (Cont.) SCRIPT_ADVISOR_TASK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir_name</td>
<td>Directory name to which to write the generated script. If the name is not specified (NULL), then the function includes the script in the returned CLOB. If the name is specified, then the function returns the script as a CLOB and as a new file in the specified directory.</td>
</tr>
<tr>
<td>level</td>
<td>The level of the script to generate. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• ALL: Ignores the filter and generates a script for all findings</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: Generates a script according to the filters in place</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

• To execute this subprogram, you must have the ADVISOR privilege.
• You must be the owner of the task.
• You can execute this subprogram for AUTO_STATS_ADVISOR_TASK, which is predefined.
• This subprogram executes using invoker’s rights.

The results of performing this task depend on the privileges of the executing user:

• SYSTEM level
  • Only users with both the ANALYZE ANY and ANALYZE ANY DICTIONARY privileges can perform this task on system-level rules.
• Operation level
  The results depend on the following privileges:
  – Users with both the ANALYZE ANY and ANALYZE ANY DICTIONARY privileges can perform this task for all statistics operations.
  – Users with the ANALYZE ANY privilege but not the ANALYZE ANY DICTIONARY privilege can perform this task for statistics operations related to any schema except SYS.
  – Users with the ANALYZE ANY DICTIONARY privilege but not the ANALYZE ANY privilege can perform this task for statistics operations related to their own schema and the SYS schema.
  – Users with neither the ANALYZE ANY nor the ANALYZE ANY DICTIONARY privilege can only perform this operation for statistics operations relating to their own schema.
• Object level
  Users can perform this task for any object for which they have statistics collection privileges.

Return Values

This function returns a CLOB that contains the script.

Exceptions

• ORA-20000: Insufficient privileges
Example 162-15  Creating an Optimizer Statistics Advisor Script

This example creates a procedure named `myrep`, and then calls this procedure to print the script the implements the recommendations.

SET ECHO ON
SET FEEDBACK ON
SET SERVEROUTPUT ON
SET TRIMS ON
SET LINESIZE 300

EXECUTE DBMS_OUTPUT.ENABLE (buffer_size => 10000000);

CREATE OR REPLACE PROCEDURE myrep(p_tname VARCHAR2, p_ftype VARCHAR2,
which VARCHAR2)
IS
  v_report CLOB          := null;
  v_script CLOB          := null;
  v_ftype  VARCHAR2(400) := p_ftype;
  v_tname  VARCHAR2(400) := p_tname;
  v_len    NUMBER(10);
  v_ps     NUMBER(10)    := 10000;
  v_pn     NUMBER(10)    := 1;
  v_ret    VARCHAR2(32767);
BEGIN
  IF which = 'REPORT'
    THEN
      -- generate a report
      v_report := DBMS_STATS.REPORT_ADVISOR_TASK(
        task_name => v_tname,
        type      => v_ftype,
        section   => 'ALL',
        level     => 'ALL');
      v_len := DBMS_LOB.getlength(v_report);
      WHILE (v_pn < v_len)
        LOOP
          DBMS_OUTPUT.PUT_LINE(DBMS_LOB.SUBSTR(v_report, v_ps, v_pn));
          v_pn := v_pn + v_ps;
        END LOOP;
    ELSE
      -- generate a script
      v_script := DBMS_STATS.SCRIPT_ADVISOR_TASK(v_tname);
      v_len := DBMS_LOB.getlength(v_script);
      WHILE (v_pn < v_len)
        LOOP
          DBMS_OUTPUT.PUT_LINE(DBMS_LOB.SUBSTR(v_script, v_ps, v_pn));
          v_pn := v_pn + v_ps;
        END LOOP;
    END IF;
END;
/
162.7.111 SEED_COL_USAGE Procedure

This procedure seeds column usage information from a statements in the specified SQL tuning set, or in the database.

The procedure iterates over the SQL statements, compiles them, and then seeds column usage information for the columns that appear in these statements. You can monitor the workload on the system for given amount of time and seed the column usage information based on the columns that appear in statements executed during the monitoring window.

Syntax

```sql
DBMS_STATS.SEED_COL_USAGE (
    sqlset_name    IN    VARCHAR2,
    owner_name     IN    VARCHAR2,
    time_limit     IN    POSITIVE DEFAULT NULL);
```

Parameters

**Table 162-118 SEED_COL_USAGE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Name of the SQL tuning set that contains the statements to be monitored. If this parameter and owner_name are both null, then the procedure monitors all statements in the database for the specified time limit.</td>
</tr>
<tr>
<td>owner_name</td>
<td>Owner of the SQL tuning set that contains the statements to be monitored. If this parameter and sqlset_name are both null, then the procedure monitors all statements in the database for the specified time limit.</td>
</tr>
<tr>
<td>time_limit</td>
<td>Time limit (in seconds).</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must have the **ANALYZE ANY privilege** and the **ANALYZE ANY DICTIONARY privilege**.

Exceptions

ORA-20000: **Insufficient privileges**
Usage Notes

This procedure also records group of columns. You can create extensions for the recorded group of columns using the CREATE_EXTENDED_STATS Function procedure. If sqlset_name and owner_name are NULL, then the procedure records the column (group) usage information for the statements executed in the system in next time_limit seconds.

This monitoring procedure records different information from the traditional column usage information that is visible in SYS.COL_USAGES. The procedure stores information in SYS.COL_GROUP_USAGES.

Examples

The following example turns on monitoring for 5 minutes or 300 seconds.

```
BEGIN
    DBMS_STATS.SEED_COL_USAGE (null,null,300);
END;
```

162.7.112 SET_ADVISOR_TASK_PARAMETER Procedure

This procedure updates the value of an Optimizer Statistics Advisor task parameter.

Syntax

```
DBMS_STATS.SET_ADVISOR_TASK_PARAMETER (
    task_name     IN   VARCHAR2,
    parameter     IN   VARCHAR2,
    value         IN   VARCHAR2);
```

Parameters

Table 162-119  SET_ADVISOR_TASK_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>The name of the Optimizer Statistics Advisor task.</td>
</tr>
<tr>
<td>parameter</td>
<td>The name of the parameter to set. The function returns an error if the specified parameter does not exist.</td>
</tr>
<tr>
<td>value</td>
<td>The new value of the parameter.</td>
</tr>
</tbody>
</table>

Security Model

Note the following:

- To execute this subprogram, you must have the ADVISOR privilege.
- You must be the owner of the task.
- This subprogram executes using invoker's rights.

Exceptions

- ORA-20000: Insufficient privileges
- ORA-20001: Invalid input values
162.7.113 SET_COLUMN_STATS Procedures

This procedure sets column-related information.

In the version of this procedure that deals with user-defined statistics, the statistics type specified is the type to store in the dictionary, in addition to the actual user-defined statistics. If this statistics type is `NULL`, the statistics type associated with the index or column is stored.

Syntax

```
DBMS_STATS.SET_COLUMN_STATS (  
  ownname VARCHAR2,  
  tabname VARCHAR2,  
  colname VARCHAR2,  
  partname VARCHAR2 DEFAULT NULL,  
  stattab VARCHAR2 DEFAULT NULL,  
  statid VARCHAR2 DEFAULT NULL,  
  distcnt NUMBER DEFAULT NULL,  
  density NUMBER DEFAULT NULL,  
  nullcnt NUMBER DEFAULT NULL,  
  srec StatRec DEFAULT NULL,  
  avgclen NUMBER DEFAULT NULL,  
  flags NUMBER DEFAULT NULL,  
  statown VARCHAR2 DEFAULT NULL,  
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(    
    get_param('NO_INVALIDATE')),  
  force BOOLEAN DEFAULT FALSE);
```

Use the following for user-defined statistics:

```
DBMS_STATS.SET_COLUMN_STATS (  
  ownname VARCHAR2,  
  tabname VARCHAR2,  
  colname VARCHAR2,  
  partname VARCHAR2 DEFAULT NULL,  
  stattab VARCHAR2 DEFAULT NULL,  
  statid VARCHAR2 DEFAULT NULL,  
  ext_stats RAW,  
  stattypown VARCHAR2 DEFAULT NULL,  
  stattypname VARCHAR2 DEFAULT NULL,  
  statown VARCHAR2 DEFAULT NULL,  
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(    
    get_param('NO_INVALIDATE')),  
  force BOOLEAN DEFAULT FALSE);
```
### Parameters

**Table 162-120  SET_COLUMN_STATS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table to which this column belongs.</td>
</tr>
<tr>
<td>colname</td>
<td>Name of the column or extension</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>ext_stats</td>
<td>User-defined statistics</td>
</tr>
<tr>
<td>stattypown</td>
<td>Schema of the statistics type</td>
</tr>
<tr>
<td>stattypname</td>
<td>Name of the statistics type</td>
</tr>
<tr>
<td>distcnt</td>
<td>Number of distinct values</td>
</tr>
<tr>
<td>density</td>
<td>Column density. If this value is NULL and if distcnt is not NULL, then density is derived from distcnt.</td>
</tr>
<tr>
<td>nullcnt</td>
<td>Number of NULLs</td>
</tr>
<tr>
<td>srec</td>
<td>StatRec structure filled in by a call to PREPARE_COLUMN_VALUES or GET_COLUMN_STATS</td>
</tr>
<tr>
<td>avgclen</td>
<td>Average length for the column (in bytes)</td>
</tr>
<tr>
<td>flags</td>
<td>For internal Oracle use (should be left as NULL)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>force</td>
<td>Sets the values even if statistics of the column are locked</td>
</tr>
</tbody>
</table>

### Exceptions

- **ORA-20000**: Object does not exist or insufficient privileges
- **ORA-20001**: Invalid or inconsistent input values
- **ORA-20005**: Object statistics are locked
Usage Notes

To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.

162.7.114 SET_DATABASE_PREFS Procedure

This procedure sets the statistics preferences of all the tables, excluding the tables owned by the database. These tables can by included by passing `TRUE` for the `add_sys` parameter.

Syntax

```sql
DBMS_STATS.SET_DATABASE_PREFS (  
  pname            IN   VARCHAR2,  
  pvalue           IN   VARCHAR2,  
  add_sys          IN   BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| pname     | Preference name. The existing value for following preferences can be set and default preference values will be used:  
  - APPROXIMATE_NDV_ALGORITHM  
  - AUTO_STAT_EXTENSIONS  
  - CASCADE  
  - DEGREE  
  - ESTIMATE_PERCENT  
  - GLOBAL_TEMP_TABLE_STATS  
  - GRANULARITY  
  - INCREMENTAL  
  - INCREMENTAL_STALENESS  
  - INCREMENTAL_LEVEL  
  - METHOD_OPT  
  - NO_INVALIDATE  
  - OPTIONS  
  - PREFERENCE_OVERRIDES_PARAMETER  
  - PUBLISH  
  - STALE_PERCENT  
  - STATCATEGORY  
  - TABLE_CACHED_BLOCKS |
| pvalue    | Preference value. If `NULL` is specified, it will set the Oracle default values |
| add_sys   | Value `TRUE` will include the Oracle-owned tables |
Table 162-122  Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
</table>
| APPROXIMATE_NDV_ALGORITHM   | Specifies the synopsis generation algorithm. A synopsis is special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. Consider a synopsis as an internal management structure that samples distinct values. You can specify the following preferences:  
  · REPEAT OR HYPERLOGLOG  
    This is the default. If INCREMENTAL is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format. This approach is attractive when existing performance is acceptable, and you do not want to incur the performance cost of reformatting legacy content.  
  · ADAPTIVE SAMPLING  
    The database uses the adaptive sampling algorithm for all synopses. This is the most conservative option.  
  · HYPERLOGLOG  
    The database uses the HyperLogLog algorithm for all new and stale synopses. In contrast to dynamic sampling, the HyperLogLog algorithm uses a randomization technique. The advantages of HyperLogLog over adaptive sampling are:  
    – The accuracy of the new algorithm is similar to the original algorithm.  
    – The memory required is significantly lower, which typically leads to huge reductions in synopsis size. |
| AUTO_STAT_EXTENSIONS        | Controls the automatic creation of extensions when database statistics are gathered. You can set the following values:  
  · ON — When applicable, a SQL plan directive can trigger the creation of column group statistics based on usage of columns in the predicates in the workload.  
  · OFF — The database does not create column group statistics automatically. The database creates them only when the CREATE_EXTENDED_STATS function is executed, or when extended statistics are specified explicitly in the METHOD_OPT clause of DBMS_STATS. This is the default. |
| CASCADE                     | Determines whether to collect index statistics as part of gathering table statistics. |
| DEGREE                      | Determines the degree of parallelism used for gathering statistics. |
Table 162-122  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESTIMATE_PERCENT</td>
<td>Determines the percentage of rows to sample. The valid range is between 0.000001 and 100. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to enable the database to determine the appropriate sample size for optimal statistics. This is the default.</td>
</tr>
</tbody>
</table>
| GLOBAL_TEMP_TABLE_STATS | Controls whether the statistics gathered for a global temporary table should be stored as shared statistics or session statistics. This preference takes two values:  
                                              · SHARED — All sessions see the same set of statistics  
                                              · SESSION — Statistics gathered by the GATHER_TABLE_STATS procedure on a global temporary table are session-specific. Thus, the database only uses them for queries issued in the same session as the statistics gathering process. The database deletes session-specific statistics when a session terminates. |
| GRANULARITY        | Determines the granularity of statistics to collect. This preference is only relevant for partitioned tables. The following values are valid:  
                                              · ALL — Gathers all statistics: subpartition, partition, and global.  
                                              · AUTO — Determines the granularity based on the partitioning type. This is the default value.  
                                              · DEFAULT — Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. Use GLOBAL AND PARTITION for this functionality.  
                                              · GLOBAL — Gathers global statistics.  
                                              · GLOBAL AND PARTITION — Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.  
                                              · PARTITION — Gathers partition-level statistics.  
                                              · SUBPARTITION — Gathers subpartition-level statistics. |
Table 162-122  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL</td>
<td>Determines whether the global statistics for a partitioned table are maintained without performing a full table scan. When a table is partitioned, an application typically loads data into a new partition. As new partitions are added and data is loaded, global table statistics must be kept up to date. If the following conditions are met, then the database updates the global table statistics by scanning only the changed partitions instead of the entire table:</td>
</tr>
<tr>
<td></td>
<td>1. The <code>INCREMENTAL</code> value for the partitioned table is set to <code>TRUE</code>.</td>
</tr>
<tr>
<td></td>
<td>2. The <code>PUBLISH</code> value for the partitioned table is set to <code>TRUE</code>.</td>
</tr>
<tr>
<td></td>
<td>3. The user specifies <code>AUTO_SAMPLE_SIZE</code> for <code>ESTIMATE_PERCENT</code> and <code>AUTO</code> for <code>GRANULARITY</code> when gathering statistics on the table.</td>
</tr>
<tr>
<td></td>
<td>If the <code>INCREMENTAL</code> value for the partitioned table was set to <code>FALSE</code> (default value), then the database uses a full table scan to maintain the global statistics. This technique is a much more resource-intensive and time-consuming operation for large tables.</td>
</tr>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>Controls which synopses to collect when <code>INCREMENTAL</code> preference is set to <code>TRUE</code>. It takes the following values:</td>
</tr>
<tr>
<td></td>
<td>1. <code>PARTITION</code> — Gathers partition-level synopses. This is the default value. If <code>PARTITION</code> is set on a nonpartitioned table, then the database does not gather synopses.</td>
</tr>
<tr>
<td></td>
<td>2. <code>TABLE</code> — Gathers table-level synopses. Specify this value when you want to exchange this table with a partition. Before the exchange, you can run <code>GATHER_TABLE_STATS</code> on this table with <code>INCREMENTAL</code> set to <code>TRUE</code> and <code>INCREMENTAL_LEVEL</code> to <code>TABLE</code>. The result is that the database gathers table-level synopses on this table. After the exchange, the partition has synopses that come from the table-level synopses of the table before the exchange. You can only use preference value in the <code>SET_TABLE_PREFS</code> procedure; this value is not allowed in the other <code>SET_*_PREFS</code> procedures.</td>
</tr>
</tbody>
</table>
### Table 162-122  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_STALENESS</td>
<td>Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT,USE_LOCKED_STATS,ALLOW_MIXED_FORMAT'. The parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>· USE_STALE_PERCENT—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference. For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics. Assume a different case in which STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.</td>
</tr>
<tr>
<td></td>
<td>· USE_LOCKED_STATS—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes. For example, assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS, USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.</td>
</tr>
<tr>
<td></td>
<td>· ALLOW_MIXED_FORMAT—Adaptive sampling synopses and HyperLogLog synopses are permitted to coexist.</td>
</tr>
<tr>
<td></td>
<td>· NULL—A partition or subpartition is considered stale when it has any DML changes. For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics. Note that the following two executions are different:</td>
</tr>
</tbody>
</table>

```sql
EXEC DBMS_STATS.SET_TABLE_PREFS
  ('sh', 'sales','INCREMENTAL_STALENESS', 'NULL');

EXEC DBMS_STATS.SET_TABLE_PREFS
  ('sh', 'sales','INCREMENTAL_STALENESS', null);
```
### Table 162-122 (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preference</strong></td>
<td>The first execution uses single quotes to set the preference to the value <strong>NULL</strong>, whereas the second sets the preference to the default, which is <strong>ALLOW_MIXED_FORMAT.</strong></td>
</tr>
<tr>
<td><strong>METHOD_OPT</strong></td>
<td>Controls column statistics collection and histogram creation. When setting preferences at the global, schema, database, or dictionary level, only <strong>FOR ALL</strong> syntax is allowed:</td>
</tr>
<tr>
<td></td>
<td>- <strong>FOR ALL</strong> [INDEXED</td>
</tr>
<tr>
<td></td>
<td>The <strong>size_clause</strong> is defined as **size_clause := SIZE (integer</td>
</tr>
<tr>
<td></td>
<td>integer — Specifies the number of histogram buckets. The number must be between 1 and 2048.</td>
</tr>
<tr>
<td></td>
<td>REPEAT — Collects histograms only on the columns that already have histograms.</td>
</tr>
<tr>
<td></td>
<td>AUTO — Determines the columns on which to collect histograms based on data distribution and the workload of the columns.</td>
</tr>
<tr>
<td></td>
<td>SKEWONLY — Determines the columns on which to collect histograms based on the data distribution of the columns.</td>
</tr>
<tr>
<td></td>
<td>The default is <strong>FOR ALL COLUMNS SIZE AUTO.</strong> You can change the value using the <strong>SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</strong></td>
</tr>
<tr>
<td><strong>NO_INVALIDATE</strong></td>
<td>Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. The default is <strong>DBMS_STATS.AUTO_INVALIDATE</strong>, which means the database decides when to invalidate dependent cursors.</td>
</tr>
<tr>
<td></td>
<td>If set to <strong>TRUE</strong>, then the database not invalidate dependent cursors. If set to <strong>FALSE</strong>, then the procedure invalidates dependent cursors immediately.</td>
</tr>
<tr>
<td><strong>OPTIONS</strong></td>
<td>Specifies which objects require statistics to be gathered. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>- <strong>GATHER</strong> — Gathers statistics on all objects in the database. This is the default.</td>
</tr>
<tr>
<td></td>
<td>- <strong>GATHER AUTO</strong> — Gathers all necessary statistics automatically. This is the default.</td>
</tr>
<tr>
<td></td>
<td>The database implicitly determines which objects need new statistics and determines how to gather those statistics. When <strong>GATHER AUTO</strong> is specified, the only additional valid parameters are <strong>comp_id, no_invalidate, stattab, statid, and statown</strong>; all other parameter settings will be ignored. Also, the database returns a list of objects processed.</td>
</tr>
</tbody>
</table>
Table 162-122  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFERENCEOverridesParameter</td>
<td>Determines whether to override the input value of a parameter with the preference value of that parameter for a statistics operation. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- TRUE — Ignores input parameter values, and uses the value of the corresponding preference.</td>
</tr>
<tr>
<td></td>
<td>- FALSE — Obeys input parameter values.</td>
</tr>
<tr>
<td></td>
<td>Specifying this preference does not change the order of precedence of table, global, and default.</td>
</tr>
<tr>
<td>Publish</td>
<td>Determines whether the database publishes newly gathered statistics after the gathering job completes. You can gather statistics without publishing them immediately. This technique enables you to test new statistics before publishing them.</td>
</tr>
<tr>
<td>Stale_percent</td>
<td>Determines the percentage of rows in a table that must change before the statistics on that table are stale and need to be regathered.</td>
</tr>
<tr>
<td></td>
<td>The valid domain for stale_percent is non-negative numbers. The default value is 10, which means that a table having more than 10% of changes is considered stale.</td>
</tr>
<tr>
<td>Stat_category</td>
<td>Specifies which statistics to import or export, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>- OBJECT_STATS — table statistics, column statistics and index statistics (default)</td>
</tr>
<tr>
<td></td>
<td>- SYNOPSES — information to support incremental statistics</td>
</tr>
<tr>
<td></td>
<td>- REALTIME_STATS — specifies only real-time statistics.</td>
</tr>
<tr>
<td></td>
<td>The value ‘OBJECT_STATS, SYNOPSES’ specifies table statistics, column statistics, index statistics, and synopses.</td>
</tr>
<tr>
<td>Table Cached Blocks</td>
<td>Specifies the average number of blocks assumed to be cached in the buffer cache when calculating the index clustering factor.</td>
</tr>
</tbody>
</table>

Security Model

To run this procedure, you must have the SYSDBA role or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Invalid or illegal input values

Usage Notes

Both arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.
Examples

DBMS_STATS.SET_DATABASE_PREFS('CASCADE', 'DBMS_STATS.AUTO_CASCADE');
DBMS_STATS.SET_DATABASE_PREFS('ESTIMATE_PERCENT', '9');
DBMS_STATS.SET_DATABASE_PREFS('DEGREE', '99');

See Also:

Oracle Database SQL Tuning Guide to learn how to set optimizer statistics preferences

162.7.115 SET_GLOBAL_PREFS Procedure

This procedure sets statistics preferences at the global level.

Syntax

DBMS_STATS.SET_GLOBAL_PREFS (  
    pname   IN   VARCHAR2,  
    pvalue  IN   VARCHAR2);
### Parameters

#### Table 162-123  SET_GLOBAL_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **pname** | Preference name. The default value for the following preferences can be set:  
- APPROXIMATE_NDV_ALGORITHM  
- AUTO_STAT_EXTENSIONS  
- AUTOSTATS_TARGET  
- CASCADE  
- CONCURRENT  
- DEGREE  
- ESTIMATE_PERCENT  
- GLOBAL_TEMP_TABLE_STATS  
- GRANULARITY  
- INCREMENTAL  
- INCREMENTAL_LEVEL  
- INCREMENTAL_STALENESS  
- METHOD_OPT  
- NO_INVALIDATE  
- OPTIONS  
- PREFERENCE_OVERRIDES_PARAMETER  
- PUBLISH  
- STALE_PERCENT  
- STAT_CATEGORY  
- TABLE_CACHED_BLOCKS  
- WAIT_TIME_TO_UPDATE_STATS |
<p>| <strong>pvalue</strong> | Preference value. If NULL is specified, it will set the Oracle default values |</p>
<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPROXIMATE_NDV_ALGORITHM</td>
<td>Specifies the synopsis generation algorithm. A synopsis is special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. Consider a synopsis as an internal management structure that samples distinct values. You can specify the following preferences:</td>
</tr>
<tr>
<td></td>
<td>• REPEAT OR HYPERLOGLOG</td>
</tr>
<tr>
<td></td>
<td>This is the default. If INCREMENTAL is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format. This approach is attractive when existing performance is acceptable, and you do not want to incur the performance cost of reformatting legacy content.</td>
</tr>
<tr>
<td></td>
<td>• ADAPTIVE SAMPLING</td>
</tr>
<tr>
<td></td>
<td>The database uses the adaptive sampling algorithm for all synopses. This is the most conservative option.</td>
</tr>
<tr>
<td></td>
<td>• HYPERLOGLOG</td>
</tr>
<tr>
<td></td>
<td>The database uses the HyperLogLog algorithm for all new and stale synopses. In contrast to dynamic sampling, the HyperLogLog algorithm uses a randomization technique. The advantages of HyperLogLog over adaptive sampling are:</td>
</tr>
<tr>
<td></td>
<td>– The accuracy of the new algorithm is similar to the original algorithm.</td>
</tr>
<tr>
<td></td>
<td>– The memory required is significantly lower, which typically leads to huge reductions in synopsis size.</td>
</tr>
<tr>
<td>AUTO_STAT_EXTENSIONS</td>
<td>Controls the automatic creation of extensions when database statistics are gathered.</td>
</tr>
<tr>
<td></td>
<td>You can set the following values:</td>
</tr>
<tr>
<td></td>
<td>• ON — When applicable, a SQL plan directive can trigger the creation of column group statistics based on usage of columns in the predicates in the workload.</td>
</tr>
<tr>
<td></td>
<td>• OFF — The database does not create column group statistics automatically. The database creates them only when the CREATE_EXTENDED_STATS function is executed, or when extended statistics are specified explicitly in the METHOD_OPT clause of DBMS_STATS. This is the default.</td>
</tr>
</tbody>
</table>
Table 162-124 (Cont.) Global Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSTATS_TARGET</td>
<td>Controls the objects considered for statistics collection. It takes the following values:</td>
</tr>
<tr>
<td></td>
<td>• 'ALL' - Statistics collected for all objects in system</td>
</tr>
<tr>
<td></td>
<td>• 'ORACLE' - Statistics collected for all Oracle owned objects</td>
</tr>
<tr>
<td></td>
<td>• 'AUTO' - Oracle decides on which objects to collect statistics</td>
</tr>
<tr>
<td></td>
<td>This preference is applicable only for automatic statistics collection.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Determines whether to collect index statistics as part of gathering table statistics.</td>
</tr>
<tr>
<td>CONCURRENT</td>
<td>Determines whether statistics are gathered concurrently on multiple objects, or serially, one object at a time. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• MANUAL — Concurrency is enabled only for manual statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>• AUTOMATIC — Concurrency is enabled only for the automatic statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>• ALL — Concurrency is enabled for both manual and automatic statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>• OFF — Concurrency is disabled for both manual and automatic statistics.</td>
</tr>
<tr>
<td>DEGREE</td>
<td>Determines the degree of parallelism used for gathering statistics.</td>
</tr>
<tr>
<td>ESTIMATE_PERCENT</td>
<td>Determines the percentage of rows to sample. The valid range is between 0.000001 and 100. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to enable the database to determine the appropriate sample size for optimal statistics. This is the default.</td>
</tr>
<tr>
<td>GLOBAL_TEMP_TABLE_STATS</td>
<td>Controls whether the statistics gathered for a global temporary table should be stored as shared statistics or session statistics. This preference takes two values:</td>
</tr>
<tr>
<td></td>
<td>• SHARED — All sessions see the same set of statistics</td>
</tr>
<tr>
<td></td>
<td>• SESSION — Statistics gathered by the GATHER_TABLE_STATS procedure on a global temporary table are session-specific. Thus, the database only uses them for queries issued in the same session as the statistics gathering process. The database deletes session-specific statistics when a session terminates.</td>
</tr>
</tbody>
</table>
Table 162-124  (Cont.) Global Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GRANULARITY</strong></td>
<td>Determines the granularity of statistics to collect. This preference is only relevant for partitioned tables.</td>
</tr>
<tr>
<td></td>
<td>The following values are valid:</td>
</tr>
<tr>
<td></td>
<td>• <strong>ALL</strong> — Gathers all statistics: subpartition, partition, and global.</td>
</tr>
<tr>
<td></td>
<td>• <strong>AUTO</strong> — Determines the granularity based on the partitioning type. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>• <strong>DEFAULT</strong> — Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. Use <strong>GLOBAL AND PARTITION</strong> for this functionality.</td>
</tr>
<tr>
<td></td>
<td>• <strong>GLOBAL</strong> — Gathers global statistics.</td>
</tr>
<tr>
<td></td>
<td>• <strong>GLOBAL AND PARTITION</strong> — Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</td>
</tr>
<tr>
<td></td>
<td>• <strong>PARTITION</strong> — Gathers partition-level statistics.</td>
</tr>
<tr>
<td></td>
<td>• <strong>SUBPARTITION</strong> — Gathers subpartition-level statistics.</td>
</tr>
<tr>
<td><strong>INCREMENTAL</strong></td>
<td>Determines whether the global statistics for a partitioned table are maintained without performing a full table scan.</td>
</tr>
<tr>
<td></td>
<td>When a table is partitioned, an application typically loads data into a new partition. As new partitions are added and data is loaded, global table statistics must be kept up to date. If the following conditions are met, then the database updates the global table statistics by scanning only the changed partitions instead of the entire table:</td>
</tr>
<tr>
<td></td>
<td>• The <strong>INCREMENTAL</strong> value for the partitioned table is set to <strong>TRUE</strong>.</td>
</tr>
<tr>
<td></td>
<td>• The <strong>PUBLISH</strong> value for the partitioned table is set to <strong>TRUE</strong>.</td>
</tr>
<tr>
<td></td>
<td>• The user specifies <strong>AUTO_SAMPLE_SIZE</strong> for <strong>ESTIMATE_PERCENT</strong> and <strong>AUTO</strong> for <strong>GRANULARITY</strong> when gathering statistics on the table.</td>
</tr>
<tr>
<td></td>
<td>If the <strong>INCREMENTAL</strong> value for the partitioned table was set to <strong>FALSE</strong> (default value), then the database uses a full table scan to maintain the global statistics. This technique is a much more resource-intensive and time-consuming operation for large tables.</td>
</tr>
</tbody>
</table>
Table 162-124  (Cont.) Global Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>Controls which synopses to collect when INCREMENTAL preference is set to TRUE. It takes the following values:</td>
</tr>
<tr>
<td></td>
<td>• PARTITION — Gathers partition-level synopses.</td>
</tr>
<tr>
<td></td>
<td>This is the default value. If PARTITION is set on a nonpartitioned table, then the database does not gather synopses.</td>
</tr>
<tr>
<td></td>
<td>• TABLE — Gathers table-level synopses.</td>
</tr>
<tr>
<td></td>
<td>Specify this value when you want to exchange this table with a partition. Before the exchange, you can run GATHER_TABLE_STATS on this table with</td>
</tr>
<tr>
<td></td>
<td>INCREMENTAL set to TRUE and INCREMENTAL_LEVEL to TABLE. The result is that the database gathers table-level synopses on this table. After the</td>
</tr>
<tr>
<td></td>
<td>exchange, the partition has synopses that come from the table-level synopses of the table before the exchange. You can only use preference</td>
</tr>
<tr>
<td></td>
<td>value in the SET_TABLE_PREFS procedure: this value is not allowed in the other SET_ *_PREFS procedures.</td>
</tr>
</tbody>
</table>
Table 162-124   (Cont.) Global Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_STALENESS</td>
<td>Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT,USE_LOCKED_STATS,ALLOW_MIXED_FORMAT'. The parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>• USE_STALE_PERCENT—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference. For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics. Assume a different case in which STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.</td>
</tr>
<tr>
<td></td>
<td>• USE_LOCKED_STATS—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes. For example, assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS,USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.</td>
</tr>
<tr>
<td></td>
<td>• ALLOW_MIXED_FORMAT—Adaptive sampling synopses and HyperLogLog synopses are permitted to coexist.</td>
</tr>
<tr>
<td></td>
<td>• NULL—A partition or subpartition is considered stale when it has any DML changes. For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics.</td>
</tr>
</tbody>
</table>
Table 162-124 (Cont.) Global Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Note that the following two executions are different:</td>
</tr>
<tr>
<td></td>
<td>EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'sales', 'INCREMENTAL_STALENESS', 'NULL');</td>
</tr>
<tr>
<td></td>
<td>EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'sales', 'INCREMENTAL_STALENESS', null);</td>
</tr>
<tr>
<td></td>
<td>The first execution uses single quotes to set the preference to the value NULL, whereas the second sets the preference to the default, which is ALLOW_MIXED_FORMAT.</td>
</tr>
</tbody>
</table>

**METHOD_OPT**

Controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:

- FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]
  - size_clause is defined as size_clause ::= SIZE {integer | REPEAT | AUTO | SKEWONLY}
  - integer: Number of histogram buckets. Must be in the range [1,2048].
  - REPEAT: Collects histograms only on the columns that already have histograms
  - AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns
  - SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns
  - column_name: name of a column
  - extension: can be either a column group in the format of (column_name, column_name [, ...]) or an expression

The default is FOR ALL COLUMNS SIZE AUTO.

**NO_INVALIDATE**

Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. The default is DBMS_STATS.AUTO_INVALIDATE, which means the database decides when to invalidate dependent cursors.

If set to TRUE, then the database not invalidate dependent cursors. If set to FALSE, then the procedure invalidates dependent cursors immediately.
<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Determines the options parameter used in the GATHER_TABLE_STATS procedure. The preference takes the following values:</td>
</tr>
<tr>
<td></td>
<td>· <strong>GATHER</strong> — Gathers statistics for all objects in the table. This is the default.</td>
</tr>
<tr>
<td></td>
<td>· <strong>GATHER AUTO</strong> — Gathers all necessary statistics automatically. Oracle recommends setting GATHER AUTO on tables that undergo bulk load operations that gather statistics. This option is only applicable to tables that do not have INCREMENTAL enabled. Running GATHER_TABLE_STATS procedure on these tables with the GATHER AUTO option skips regathering the already fresh statistics.</td>
</tr>
<tr>
<td>PREFERENCE_OVERRIDES_PARAMETER</td>
<td>Determines whether to override the input value of a parameter with the preference value of that parameter for a statistics operation. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>· <strong>TRUE</strong> — Ignores input parameter values, and uses the value of the corresponding preference.</td>
</tr>
<tr>
<td></td>
<td>· <strong>FALSE</strong> — Obeys input parameter values.</td>
</tr>
<tr>
<td></td>
<td>Specifying this preference does not change the order of precedence of table, global, and default.</td>
</tr>
<tr>
<td>PUBLISH</td>
<td>Determines whether the database publishes newly gathered statistics after the gathering job completes.</td>
</tr>
<tr>
<td></td>
<td>You can gather statistics without publishing them immediately. This technique enables you to test new statistics before publishing them.</td>
</tr>
<tr>
<td>STALE_PERCENT</td>
<td>Determines the percentage of rows in a table that must change before the statistics on that table are stale and need to be regathered.</td>
</tr>
<tr>
<td></td>
<td>The valid domain for stale_percent is non-negative numbers. The default value is 10, which means that a table having more than 10% of changes is considered stale.</td>
</tr>
<tr>
<td>STAT_CATEGORY</td>
<td>Specifies which statistics to import or export, accepting multiple values separated by a comma. Values supported:</td>
</tr>
<tr>
<td></td>
<td>· <strong>OBJECT_STATS</strong> - table statistics, column statistics and index statistics (default)</td>
</tr>
<tr>
<td></td>
<td>· <strong>SYNOPSES</strong> - information to support incremental statistics</td>
</tr>
<tr>
<td></td>
<td>The value 'OBJECT_STATS, SYNOPSIS' specifies table statistics, column statistics, index statistics, and synopses.</td>
</tr>
<tr>
<td>TABLE_CACHED_BLOCKS</td>
<td>Specifies the average number of blocks assumed to be cached in the buffer cache when calculating the index clustering factor.</td>
</tr>
</tbody>
</table>
### Table 162-124  (Cont.) Global Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAIT_TIME_TO_UPDATE_STATS</td>
<td>Specifies the number of minutes before timing out for locks and pins required for updating statistics. It accepts values in the range 0 to 65535. The default value is 15 minutes. The value 0 gets the locks and pins in no-wait mode.</td>
</tr>
</tbody>
</table>

#### Security Model

To run this procedure, you must have the **SYSDBA** or both **ANALYZE ANY DICTIONARY** and **ANALYZE ANY system privilege**.

#### Exceptions

- **ORA-20000**: Insufficient privileges
- **ORA-20001**: Invalid or illegal input values

#### Usage Notes

- This setting is honored only if there is no preference specified for the table to be analyzed.
- Both arguments are of type **VARCHAR2** and values are enclosed in quotes, even when they represent numbers.

#### Example 162-16  Overriding Statistics Preferences at the Global Level

You use the **SET_GLOBAL_PREFS** procedure to set the **ESTIMATE_PERCENT** preference to 5 for every table in the database that does not have a table preference set. Because **sh.costs** does not have a preference set, the global setting applies to this table.

```sql
SQL> EXEC DBMS_STATS.SET_GLOBAL_PREFS ('ESTIMATE_PERCENT', '5');
PL/SQL procedure successfully completed.
```

You use **SET_TABLE_PREFS** to set the **PREFERENCE_OVERRIDES_PARAMETER** preference to true for the **sh.sales** table only.

```sql
SQL> EXEC DBMS_STATS.SET_TABLE_PREFS('sh','sales','PREFERENCE_OVERRIDES_PARAMETER','TRUE');
PL/SQL procedure successfully completed.
```

A script attempts to set **ESTIMATE_PERCENT** to 10 when gathering statistics for **sh.sales**. However, because **PREFERENCE_OVERRIDES_PARAMETER** is **true** for this table, and because a
global preference is defined, Oracle Database gathers statistics using the global setting of 5 rather than the specified setting of 10:

SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS ('sh', 'costs', ESTIMATE_PERCENT=>10);

PL/SQL procedure successfully completed.

See Also:
Oracle Database SQL Tuning Guide to learn how to set optimizer statistics preferences

162.7.116 SET_INDEX_STATS Procedures

These procedures set index-related statistics.

The version of this procedure that accepts ext_stats sets statistics for use with domain indexes. The statistics type specified is the type to store in the dictionary, in addition to the actual user-defined statistics. If this statistics type is null, then the database stores the statistics type associated with the index or column.

Syntax

```
DBMS_STATS.SET_INDEX_STATS (  
  ownname       VARCHAR2,  
  indname       VARCHAR2,  
  partname      VARCHAR2 DEFAULT NULL,  
  stattab       VARCHAR2 DEFAULT NULL,  
  statid        VARCHAR2 DEFAULT NULL,  
  numrows       NUMBER DEFAULT NULL,  
  numblks       NUMBER DEFAULT NULL,  
  numdist       NUMBER DEFAULT NULL,  
  avglblk       NUMBER DEFAULT NULL,  
  avgdblk       NUMBER DEFAULT NULL,  
  clstfct       NUMBER DEFAULT NULL,  
  indlevel      NUMBER DEFAULT NULL,  
  flags         NUMBER DEFAULT NULL,  
  statown       VARCHAR2 DEFAULT NULL,  
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(get_param('NO_INVALIDATE')),  
  guessq        NUMBER DEFAULT NULL,  
  cachedblk     NUMBER DEFAULT NULL,  
  cachehit      NUMBER DEFAULT NULL,  
  force         BOOLEAN DEFAULT FALSE);
```

Use the following syntax for user-defined domain index statistics:

```
DBMS_STATS.SET_INDEX_STATS (  
  ownname       VARCHAR2,  
  indname       VARCHAR2,
```
Parameters

Table 162-125  SET_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>indname</td>
<td>Name of the index</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the index partition in which to store the statistics. If the index is partitioned and if partname is NULL, then the statistics are stored at the global index level.</td>
</tr>
<tr>
<td>stattab</td>
<td>User statistics table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)</td>
</tr>
<tr>
<td>ext_stats</td>
<td>User-defined statistics</td>
</tr>
<tr>
<td>stattypown</td>
<td>Schema of the statistics type</td>
</tr>
<tr>
<td>stattypname</td>
<td>Name of the statistics type</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the index (partition)</td>
</tr>
<tr>
<td>numlblks</td>
<td>Number of leaf blocks in the index (partition)</td>
</tr>
<tr>
<td>numdist</td>
<td>Number of distinct keys in the index (partition)</td>
</tr>
<tr>
<td>avglblk</td>
<td>Average integral number of leaf blocks in which each distinct key appears for this index (partition). If not provided, then this value is derived from numlblks and numdist.</td>
</tr>
<tr>
<td>avgdblk</td>
<td>Average integral number of data blocks in the table pointed to by a distinct key for this index (partition). If not provided, then this value is derived from clstfct and numdist.</td>
</tr>
<tr>
<td>clstfct</td>
<td>See clustering_factor column of the all_indexes view for a description</td>
</tr>
<tr>
<td>indlevel</td>
<td>Height of the index (partition)</td>
</tr>
<tr>
<td>flags</td>
<td>For internal Oracle use (should be left as NULL)</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname)</td>
</tr>
</tbody>
</table>
Table 162-125  (Cont.) SET_INDEX_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_invalidate</td>
<td>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>guessq</td>
<td>Guess quality. See the pct_direct_access column of the all_indexes view for a description.</td>
</tr>
<tr>
<td>cachedblk</td>
<td>Internal use only. Do not set.</td>
</tr>
<tr>
<td>cachehit</td>
<td>Internal use only. Do not set.</td>
</tr>
<tr>
<td>force</td>
<td>Sets the values even if statistics of the index are locked</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid input value

ORA-20005: Object statistics are locked

Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.

- Oracle maintains cachedblk and cachehit at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS procedure for auto mode or DBMS_STATS.GATHER_SYSTEM_STATS for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a ‘confidence factor’ for each cachehit and a cachedblk for each object. If the ‘confidence factor’ for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.

- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other “minor” workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.

- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations...
– When not enough data has been analyzed, such as when an object has been recently created.
– When the system does not have one major workload resulting in averages not corresponding to real values.

**See Also:**
Oracle Database SQL Tuning Guide to learn how to set artificial statistics

### 162.7.117 SET_PARAM Procedure

This deprecated procedure sets default values for parameters of `DBMS_STATS` procedures.

**Note:**
This subprogram has been replaced by improved technology and is maintained only for purposes of backward compatibility. In this case, use the `SET_GLOBAL_PREFS` Procedure.

See also [DBMS_STATS Deprecated Subprograms](#).

You can use the `GET_PARAM` Function to get the current default value of a parameter.

**Syntax**

```sql
DBMS_STATS.SET_PARAM (
    pname IN VARCHAR2,
    pval IN VARCHAR2);
```

**Parameters**

**Table 162-126  SET_PARAM Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pname</code></td>
<td>The parameter name. The default value for following parameters can be set.</td>
</tr>
<tr>
<td></td>
<td>- <code>CASCADE</code> - The default value for <code>CASCADE</code> set by <code>SET_PARAM</code> is not used by export/import procedures. It is used only by gather procedures.</td>
</tr>
<tr>
<td></td>
<td>- <code>DEGREE</code></td>
</tr>
<tr>
<td></td>
<td>- <code>ESTIMATE_PERCENT</code></td>
</tr>
<tr>
<td></td>
<td>- <code>METHOD_OPT</code></td>
</tr>
<tr>
<td></td>
<td>- <code>NO_INVALIDATE</code></td>
</tr>
<tr>
<td></td>
<td>- <code>GRANULARITY</code></td>
</tr>
<tr>
<td></td>
<td>- <code>AUTOSTATS_TARGET</code> - This parameter is applicable only for auto statistics collection. The value of this parameter controls the objects considered for statistics collection (see <code>pval</code>).</td>
</tr>
</tbody>
</table>

---

**ORACLE**

162-223
Table 162-126  (Cont.) SET_PARAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pval</td>
<td>The parameter value. If NULL is specified, it will set the default value determined by Oracle. When pname is AUTOSTATS_TARGET, the following are valid values:</td>
</tr>
<tr>
<td></td>
<td>· 'ALL' - Statistics are collected for all objects in the system</td>
</tr>
<tr>
<td></td>
<td>· 'ORACLE' - Statistics are collected for all Oracle owned objects</td>
</tr>
<tr>
<td></td>
<td>· 'AUTO' - Oracle decides for which objects to collect statistics</td>
</tr>
</tbody>
</table>

Usage Notes

- To run this procedure, you must have the SYSDBA or both the ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.

- Note that both arguments are of type VARCHAR2 and the values need to be enclosed in quotes even when they represent numbers.

- Note also the difference between NULL and 'NULL':
  - When NULL is unquoted, this sets the parameter to the value Oracle recommends.
  - In the case of the quoted 'NULL', this sets the value of the parameter to NULL.

Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or illegal input value

Examples

DBMS_STATS.SET_PARAM('CASCADE','DBMS_STATS.AUTO_CASCADE');
DBMS_STATS.SET_PARAM('ESTIMATE_PERCENT','5');
DBMS_STATS.SET_PARAM('DEGREE','NULL');

162.7.118 SET_PROCESSING_RATE Procedure

This procedure sets the value of rate of processing for a given operation.

Syntax

```sql
DBMS_STATS.SET_PROCESSING_RATE (    opname      IN    VARCHAR2,    procrate    IN    NUMBER);
```

Parameters

Table 162-127  SET_PROCESSING_RATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opname</td>
<td>Name of the operation.</td>
</tr>
</tbody>
</table>
Table 162-127  (Cont.) SET_PROCESSING_RATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| procrate  | Processing rate. Valid values are as follows:  
|           | - AGGR       |
|           | - ALL        |
|           | - CPU        |
|           | - CPU_ACCESS |
|           | - CPU_AGGR   |
|           | - CPU_BYTES_PER_SEC |
|           | - CPU_FILTER |
|           | - CPU_GBY    |
|           | - CPU_HASH_JOIN |
|           | - CPU_JOIN   |
|           | - CPU_NL_JOIN|
|           | - CPU_RANDOM_ACCESS |
|           | - CPU_SEQUENTIAL_ACCESS |
|           | - CPU_SM_JOIN|
|           | - CPU_SORT   |
|           | - HASH       |
|           | - IO         |
|           | - IO_ACCESS  |
|           | - IO_BYTES_PER_SEC |
|           | - IO_RANDOM_ACCESS |
|           | - IO_SEQUENTIAL_ACCESS |
|           | - MEMCMP     |
|           | - MEMCPY     |

Security Model
You must have the OPTIMIZER_PROCESSING_RATE role to run this procedure.

Usage Notes
AUTO DOP uses processing rates to determine the optimal degree of parallelism for a SQL statement.

Exceptions
ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid or illegal input value

162.7.119 SET_SCHEMA_PREFS Procedure
This procedure sets the statistics preferences of all tables owned by the specified user.

Syntax
DBMS_STATS.SET_SCHEMA_PREFS (  
  ownername IN VARCHAR2,
Parameters

Table 162-128  SET_SCHEMA_PREFS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
</tbody>
</table>
| pname     | Preference name. You can set the default value for the following preferences:  
  - AUTO_STAT_EXTENSIONS  
  - CASCADE  
  - DEGREE  
  - ESTIMATE_PERCENT  
  - GLOBAL_TEMP_TABLE_STATS  
  - GRANULARITY  
  - INCREMENTAL  
  - INCREMENTAL_LEVEL  
  - INCREMENTAL_STALENESS  
  - METHOD_OPT  
  - NO_VALIDATE  
  - OPTIONS  
  - PREFERENCE_OVERRIDES_PARAMETER  
  - PUBLISH  
  - STALE_PERCENT  
  - TABLE_CACHED_BLOCKS  |
| pvalue    | Preference value. If NULL is specified, it sets the database default value. |
## Table 162-129  Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
</table>
| AUTO_STAT_EXTENSIONS             | Specifies the synopsis generation algorithm. A synopsis is special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. You can consider a synopsis as an internal management structure that samples distinct values.  
You can set the following preferences:  
* **REPEAT OR HYPERLOGLOG**  
  This is the default. If **INCREMENTAL** is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format. This approach is attractive when existing performance is acceptable, and you do not want to incur the performance cost of reformatting legacy content.  
* **ADAPTIVE_SAMPLING**  
  The database uses the adaptive sampling algorithm for all synopses. This is the most conservative option.  
* **HYPERLOGLOG**  
  The database uses the HyperLogLog algorithm for all new and stale synopses. In contrast to dynamic sampling, the HyperLogLog algorithm uses a randomization technique. The advantages of the HyperLogLog algorithm over adaptive sampling are:  
  - The accuracy of the new algorithm is similar to the original algorithm.  
  - The memory required is significantly lower, which typically leads to huge reductions in synopsis size. |
| CASCADE                          | Determines whether index statistics are collected as part of gathering table statistics.                                                                                                                  |
| DEGREE                           | Determines degree of parallelism used for gathering statistics.                                                                                                                                           |
| ESTIMATE_PERCENT                 | Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default. |
| GLOBAL_TEMP_TABLE_STATS          | This preference takes the following values:  
* **SHARED** - All sessions see the same set of statistics  
* **SESSION** - Statistics gathered by the GATHER_TABLE_STATS Procedure on a global temporary table are session specific, and hence are only going to be used by the queries issued in the same session as the statistics gathering process. Session-specific statistics are deleted when a session is ended. |
Table 162-129  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANULARITY</td>
<td>Determines the granularity of statistics to collect (only pertinent if the table is partitioned). Possible values are:</td>
</tr>
<tr>
<td></td>
<td>'ALL' - Gathers all (subpartition, partition, and global) statistics</td>
</tr>
<tr>
<td></td>
<td>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</td>
</tr>
<tr>
<td></td>
<td>'GLOBAL' - Gathers global statistics</td>
</tr>
<tr>
<td></td>
<td>'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</td>
</tr>
<tr>
<td></td>
<td>'PARTITION' - gathers partition-level statistics</td>
</tr>
<tr>
<td></td>
<td>'SUBPARTITION' - gathers subpartition-level statistics</td>
</tr>
<tr>
<td>INCREMENTAL</td>
<td>Determines whether the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</td>
</tr>
<tr>
<td></td>
<td>- INCREMENTAL value for the partitioned table is set to TRUE;</td>
</tr>
<tr>
<td></td>
<td>- PUBLISH value for the partitioned table is set to TRUE;</td>
</tr>
<tr>
<td></td>
<td>- User specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table</td>
</tr>
<tr>
<td></td>
<td>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</td>
</tr>
</tbody>
</table>
### Table 162-129  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>Controls which synopses to collect when INCREMENTAL preference is set to TRUE. It takes two values:</td>
</tr>
<tr>
<td></td>
<td>• TABLE - table level synopses are gathered. This is used when you want to exchange this table with a partition. You can run GATHER_TABLE_STATS on this table with INCREMENTAL to TRUE and INCREMENTAL_LEVEL to TABLE before the exchange. The result is that table level synopses are gathered on this table (currently Oracle supports only table level synopses on non-predestined tables). Once the exchange occurs, the partition will have synopses which come from the table level synopses of the table before exchange. This preference value can be only used in the SET_TABLE_PREFS Procedure. It is not allowed in the SET_GLOBAL/DATABASE/SCHEMA_PREFS procedures.</td>
</tr>
<tr>
<td></td>
<td>• PARTITION - partition level synopses are gathered. This is the default value. If PARTITION is set on a non partitioned table, no synopses are gathered.</td>
</tr>
</tbody>
</table>
Table 162-129  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_STALENESS</td>
<td>Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT, USE_LOCKED_STATS, ALLOW_MIXED_FORMAT'. The parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>• USE_STALE_PERCENT—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference. For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics. Assume a different case in which STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.</td>
</tr>
<tr>
<td></td>
<td>• USE_LOCKED_STATS—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes. Assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS, USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.</td>
</tr>
<tr>
<td></td>
<td>• ALLOW_MIXED_FORMAT—Partitions with synopses in adaptive sampling format are not considered stale, even when the APPROXIMATE_NDV_ALGORITHM preference is set to HYPERLOGLOG. The database uses existing synopses to derive global NDV. This is the default setting.</td>
</tr>
<tr>
<td></td>
<td>• NULL—A partition or subpartition is considered stale when it has any DML changes. For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics. Note that the following two executions are different:</td>
</tr>
</tbody>
</table>

EXEC DBMS_STATS.SET_TABLE_PREFS
('sh', 'sales',  
 'INCREMENTAL_STALENESS', 'NULL');
EXEC DBMS_STATS.SET_TABLE_PREFS
The first execution uses single quotes to set the preference to the value NULL, whereas the second sets the preference to the default, which is ALLOW_MIXED_FORMAT.

**METHOD_OPT**

Controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:

- **FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]**

  `size_clause` is defined as `size_clause := SIZE (integer | REPEAT | AUTO | SKEWONLY)

  - integer: Number of histogram buckets. Must be in the range [1,2048].
  - REPEAT: Collects histograms only on the columns that already have histograms
  - AUTO: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns
  - SKEWONLY: Oracle determines the columns on which to collect histograms based on the data distribution of the columns

- **column_name**: name of a column
- **extension**: can be either a column group in the format of (column_name, column_name [, ...]) or an expression

The default is FOR ALL COLUMNS SIZE AUTO.

**NO_INVALIDATE**

Controls the invalidation of dependent cursors of the tables for which statistics are gathered. If set to TRUE, then this parameter does not invalidate the dependent cursors. If set to FALSE, then this procedure invalidates the dependent cursors immediately. Use DBMS_STATS.AUTO_INVALIDATE to let the database decide when to invalidate dependent cursors. This is the default.
Table 162-129  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
</table>
| OPTIONS    | Determines the options parameter used in the GATHER_TABLE_STATS procedure. The preference takes the following values:  
  - GATHER — Gathers statistics for all objects in the table. This is the default.  
  - GATHER AUTO — Gathers all necessary statistics automatically. Oracle recommends setting GATHER AUTO on tables that undergo bulk load operations that gather statistics. This option is only applicable to tables that do not have INCREMENTAL enabled. Running GATHER_TABLE_STATS procedure on these tables with the GATHER AUTO option skips regathering the already fresh statistics. |
| PREFERENCE_OVERRIDES_PARAMETER | Determines whether to override the input value of a parameter with the preference value of that parameter for a statistics operation. Possible values are:  
  - TRUE — Ignores input parameter values, and uses the value of the corresponding preference.  
  - FALSE — Obeys input parameter values.  
  Specifying this preference does not change the order of precedence of table, global, and default. |
| PUBLISH    | Determines whether newly gathered statistics will be published after the statistics gathering job completes. In releases before Oracle Database 11g Release 1 (11.1), when a statistic gathering job completed, the new statistics were automatically published in the dictionary tables. In subsequent releases, you can gather statistics without publishing them immediately. Thus, you can test new statistics before publishing them. |
| STALE_PERCENT | Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%. Note that if you set stale_percent to zero the AUTO STATS gathering job will gather statistics for this table every time a row in the table is modified. |
| TABLE_CACHED_BLOCKS | Specifies the average number of blocks cached in the buffer cache for any table when calculating the index clustering factor. |

Security Model

To run this procedure, you must be the schema owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges  
ORA-20001: Invalid or illegal input value
Usage Notes

Both arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

Examples

DBMS_STATS.SET_SCHEMA_PREFS('SH','CASCADE','DBMS_STATS.AUTO.Cascade');
DBMS_STATS.SET_SCHEMA_PREFS('SH','ESTIMATE_PERCENT','9');
DBMS_STATS.SET_SCHEMA_PREFS('SH','DEGREE','99');

See Also:

Oracle Database SQL Tuning Guide to learn how to set optimizer statistics preferences

162.7.120 SET_SYSTEM_STATS Procedure

This procedure sets systems statistics.

Syntax

DBMS_STATS.SET_SYSTEM_STATS (  pname VARCHAR2,  pvalue NUMBER,  stattab IN VARCHAR2 DEFAULT NULL,  statid IN VARCHAR2 DEFAULT NULL,  statown IN VARCHAR2 DEFAULT NULL);
Parameters

Table 162-130  SET_SYSTEM_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>The parameter name to get, which can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>· <code>iotfrspeed</code>—I/O transfer speed in bytes for each millisecond</td>
</tr>
<tr>
<td></td>
<td>· <code>ioseektim</code>—Seek time + latency time + operating system overhead time, in milliseconds</td>
</tr>
<tr>
<td></td>
<td>· <code>sreadtim</code>—Average time to read single block (random read), in milliseconds</td>
</tr>
<tr>
<td></td>
<td>· <code>mreadtim</code>—Average time to read an mbrc block at once (sequential read), in milliseconds</td>
</tr>
<tr>
<td></td>
<td>· <code>cpuspeed</code>—Average number of CPU cycles for each second, in millions, captured for the workload (statistics collected using 'INTERVAL' or 'START' and 'STOP' options)</td>
</tr>
<tr>
<td></td>
<td>· <code>cpuspeednw</code>—Average number of CPU cycles for each second, in millions, captured for the no-workload (statistics collected using 'NOWORKLOAD' option).</td>
</tr>
<tr>
<td></td>
<td>· <code>mbrc</code>—Average multiblock read count for sequential read, in blocks</td>
</tr>
<tr>
<td></td>
<td>· <code>maxthr</code>—Maximum I/O system throughput, in bytes/second</td>
</tr>
<tr>
<td></td>
<td>· <code>slavethr</code>—Average slave I/O throughput, in bytes/second</td>
</tr>
<tr>
<td>pvalue</td>
<td>Parameter value to get</td>
</tr>
<tr>
<td>stattab</td>
<td>Identifier of the user statistics table where the statistics will be obtained. If <code>stattab</code> is null, the statistics will be obtained from the dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Optional identifier associated with the statistics saved in the <code>stattab</code></td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing <code>stattab</code> (if different from current schema)</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid input value
ORA-20002: Bad user statistics table; may need to be upgraded
ORA-20003: Unable to set system statistics
ORA-20004: Parameter does not exist

Usage Notes

To run this procedure, you need the `GATHER_SYSTEM_STATISTICS` role.
162.7.121 SET_TABLE_PREFS Procedure

This procedure sets the statistics preferences of the specified table in the specified schema.

Syntax

```sql
DBMS_STATS.SET_TABLE_PREFS ( 
    ownname    IN  VARCHAR2,
    tabname    IN  VARCHAR2,
    pname      IN  VARCHAR2,
    pvalue     IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name</td>
</tr>
<tr>
<td>tabname</td>
<td>Table name</td>
</tr>
<tr>
<td>pname</td>
<td>Preference name. You can set the default value for following preferences:</td>
</tr>
<tr>
<td></td>
<td>· APPROXIMATE_NDV_ALGORITHM</td>
</tr>
<tr>
<td></td>
<td>· AUTO_STAT_EXTENSIONS</td>
</tr>
<tr>
<td></td>
<td>· CASCADE</td>
</tr>
<tr>
<td></td>
<td>· DEGREE</td>
</tr>
<tr>
<td></td>
<td>· ESTIMATE_PERCENT</td>
</tr>
<tr>
<td></td>
<td>· GRANULARITY</td>
</tr>
<tr>
<td></td>
<td>· INCREMENTAL</td>
</tr>
<tr>
<td></td>
<td>· INCREMENTAL_LEVEL</td>
</tr>
<tr>
<td></td>
<td>· INCREMENTAL_STALENESS</td>
</tr>
<tr>
<td></td>
<td>· METHOD_OPT</td>
</tr>
<tr>
<td></td>
<td>· NO_INVALIDATE</td>
</tr>
<tr>
<td></td>
<td>· OPTIONS</td>
</tr>
<tr>
<td></td>
<td>· PREFERENCE_OVERRIDES_PARAMETER</td>
</tr>
<tr>
<td></td>
<td>· PUBLISH</td>
</tr>
<tr>
<td></td>
<td>· STALE_PERCENT</td>
</tr>
<tr>
<td></td>
<td>· TABLE_CACHED_BLOCKS</td>
</tr>
<tr>
<td>pvalue</td>
<td>Preference value. If NULL is specified, it will set the Oracle default value.</td>
</tr>
</tbody>
</table>
## Table 162-132  Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPROXIMATE_NDV_ALGORITHM</td>
<td>Specifies the synopsis generation algorithm. A synopsis is a special type of statistic that tracks the number of distinct values (NDV) for each column in a partition. Consider a synopsis as an internal management structure that samples distinct values. You can specify the following preferences:</td>
</tr>
</tbody>
</table>
|                           | - **REPEAT OR HYPERLOGLOG**  
  This is the default. If INCREMENTAL is enabled on the table, then the database preserves the format of any existing synopses that use the adaptive sampling algorithm. However, the database creates any new synopses in HyperLogLog format. This approach is attractive when existing performance is acceptable, and you do not want to incur the performance cost of reformatting legacy content.  |
|                           | - **ADAPTIVE_SAMPLING**  
  The database uses the adaptive sampling algorithm for all synopses. This is the most conservative option.  |
|                           | - **HYPERLOGLOG**  
  The database uses the HyperLogLog algorithm for all new and stale synopses. In contrast to dynamic sampling, the HyperLogLog algorithm uses a randomization technique. The advantages of HyperLogLog over adaptive sampling are:  |
<p>|                           |   - The accuracy of the new algorithm is similar to the original algorithm.  |
|                           |   - The memory required is significantly lower, which typically leads to huge reductions in synopsis size.  |
| AUTO_STAT_EXTENSIONS      | Controls the automatic creation of extensions when database statistics are gathered. You can set the following values:  |
|                           | - <strong>ON</strong> — When applicable, a SQL plan directive can trigger the creation of column group statistics based on usage of columns in the predicates in the workload.  |
|                           | - <strong>OFF</strong> — The database does not create column group statistics automatically. The database creates them only when the CREATE_EXTENDED_STATS function is executed, or when extended statistics are specified explicitly in the METHOD_OPT clause of DBMS_STATS. This is the default.  |
| CASCADE                   | Determines whether to collect index statistics as part of gathering table statistics. |</p>
<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCURRENT</td>
<td>This preference determines whether statistics will be gathered concurrently on multiple objects, or serially, one object at a time:</td>
</tr>
<tr>
<td></td>
<td>· 'MANUAL': Concurrency will be turned on only for manual statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>· 'AUTOMATIC': Concurrency will be turned on only for the automatic statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>· 'ALL': Concurrency will be turned on for both manual and automatic statistics gathering.</td>
</tr>
<tr>
<td></td>
<td>· 'OFF': Concurrency will be turned off for both manual and automatic statistics</td>
</tr>
<tr>
<td>DEGREE</td>
<td>Determines degree of parallelism used for gathering statistics.</td>
</tr>
<tr>
<td>ESTIMATE_PERCENT</td>
<td>Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.</td>
</tr>
<tr>
<td>GLOBAL_TEMP_TABLE_STATS</td>
<td>This controls whether the statistics gathered for a global temporary table should be stored as shared statistics or session statistics. It takes two values:</td>
</tr>
<tr>
<td></td>
<td>· SHARED: All sessions see the same set of statistics</td>
</tr>
<tr>
<td></td>
<td>· SESSION: Statistics gathered by the GATHER_TABLE_STATS Procedure on a global temporary table are session specific, and hence are only going to be used by the queries issued in the same session as the statistics gathering process. Session-specific statistics are deleted when a session is ended.</td>
</tr>
</tbody>
</table>
Table 162-132  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANULARITY</td>
<td>Determines granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - Gathers all (subpartition, partition, and global) statistics. 'AUTO' - Determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - Gathers global statistics. 'GLOBAL AND PARTITION' - Gathers the global and partition-level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION' - Gathers partition-level statistics. 'SUBPARTITION' - Gathers subpartition-level statistics.</td>
</tr>
</tbody>
</table>
| INCREMENTAL    | Determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:  
  • INCREMENTAL value for the partitioned table is set to TRUE;  
  • PUBLISH value for the partitioned table is set to TRUE;  
  • User specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table.  
If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables. |
### Table 162-132  (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_LEVEL</td>
<td>This value controls what synopses to collect when INCREMENTAL preference is set to TRUE. It takes two values:</td>
</tr>
<tr>
<td></td>
<td>• TABLE - table level synopses are gathered. This is used when you want to exchange this table with a partition. You can run <code>GATHER_TABLE_STATS</code> on this table with INCREMENTAL to TRUE and INCREMENTAL_LEVEL to TABLE before the exchange. The result is that table level synopses are gathered on this table (currently Oracle supports only table level synopses on non-predestined tables). Once the exchange occurs, the partition will have synopses which come from the table level synopses of the table before exchange. This preference value can be only used in the <code>SET_TABLE_PREFS</code> Procedure. It is not allowed in the <code>SET_GLOBAL/DATABASE/SCHEMA_PREFS</code> procedures.</td>
</tr>
<tr>
<td></td>
<td>• PARTITION - partition level synopses are gathered. This is the default value. If PARTITION is set on a non partitioned table, no synopses are gathered.</td>
</tr>
</tbody>
</table>
Table 162-132 (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL_STALENESS</td>
<td>Specifies when a partition or subpartition is considered stale. This parameter takes an enumeration of values, such as 'USE_STALE_PERCENT' and 'USE_LOCKED_STATS'. You can also specify multiple values, such as 'USE_STALE_PERCENT,USE_LOCKED_STATS,ALOW_MIXED_FORMAT'. The parameter accepts the following values:</td>
</tr>
<tr>
<td>· USE_STALE_PERCENT—A partition or subpartition is not considered stale when DML changes are below the threshold set by the STALE_PERCENT preference. For example, assume that STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. The partition has 5% DML changes. The database does not regather statistics. Assume a different case in which STALE_PERCENT is 10. You specify USE_STALE_PERCENT for INCREMENTAL_STALENESS. However, in this case the partition is locked and has 20% of DML changes. Because the partition is locked, the database does not regather statistics.</td>
<td></td>
</tr>
<tr>
<td>· USE_LOCKED_STATS—Locked partitions or subpartitions statistics are never considered stale, regardless of DML changes. For example, assume that STALE_PERCENT is 10. You specify 'USE_LOCKED_STATS, USE_STALE_PERCENT'. The partition, which is locked, has 20% DML changes. The partition is not considered stale. The database uses existing statistics to derive global statistics.</td>
<td></td>
</tr>
<tr>
<td>· ALLOW_MIXED_FORMAT—Adaptive sampling synopses and HyperLogLog synopses are permitted to coexist.</td>
<td></td>
</tr>
<tr>
<td>· NULL—A partition or subpartition is considered stale when it has any DML changes. For example, assume that STALE_PERCENT is 10. You specify the value 'NULL' for INCREMENTAL_STALENESS. The partition has 5% of DML changes. The database regathers statistics.</td>
<td></td>
</tr>
</tbody>
</table>
The following two executions are different:

```
EXEC DBMS_STATS.SET_TABLE_PREFS
  ('sh', 'sales', 'INCREMENTAL_STALENESS', 'NULL');

EXEC DBMS_STATS.SET_TABLE_PREFS
  ('sh', 'sales', 'INCREMENTAL_STALENESS', null);
```

The first execution uses single quotes to set the preference to the value `NULL`, whereas the second sets the preference to the default, which is `ALLOW_MIXED_FORMAT`.

**METHOD_OPT**

The value controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:

- `FOR ALL INDEXED | HIDDEN` [size_clause]

  - `size_clause` is defined as:

    ```
    size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}
    ```

    - `integer`: Number of histogram buckets. Must be in the range [1,2048].
    - `REPEAT`: Collects histograms only on the columns that already have histograms
    - `AUTO`: Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns
    - `SKEWONLY`: Oracle determines the columns on which to collect histograms based on the data distribution of the columns
    - `column_name`: name of a column
    - `extension`: can be either a column group in the format of `{column_name, column_name [,...]}` or an expression

  The default is `FOR ALL COLUMNS SIZE AUTO`.

**NO_INVALIDATE**

The value controls the invalidation of dependent cursors of the tables for which statistics are being gathered. Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE` to have Oracle decide when to invalidate dependent cursors. This is the default.
### Table 162-132 (Cont.) Statistics Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Determines the options parameter used in the GATHER_TABLE_STATS procedure. The preference takes the following values:</td>
</tr>
<tr>
<td></td>
<td>- GATHER — Gathers statistics for all objects in the table. This is the default.</td>
</tr>
<tr>
<td></td>
<td>- GATHER AUTO — Gathers all necessary statistics automatically. Oracle recommends setting GATHER AUTO on tables that undergo bulk load operations that gather statistics. This option is only applicable to tables that do not have INCREMENTAL enabled. Running GATHER_TABLE_STATS procedure on these tables with the GATHER AUTO option skips regathering the already fresh statistics.</td>
</tr>
<tr>
<td>PREFERENCE_OVERRIDES_PARAMETER</td>
<td>Determines whether to override the input value of a parameter with the preference value of that parameter for a statistics operation. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>- TRUE — Ignores input parameter values, and uses the value of the corresponding preference.</td>
</tr>
<tr>
<td></td>
<td>- FALSE — Obeys input parameter values. Specifying this preference does not change the order of precedence of table, global, and default.</td>
</tr>
<tr>
<td>PUBLISH</td>
<td>Determines whether the database publishes newly gathered statistics after the gathering job completes.</td>
</tr>
<tr>
<td></td>
<td>You can gather statistics without publishing them immediately. This technique enables you to test new statistics before publishing them.</td>
</tr>
<tr>
<td>STALE_PERCENT</td>
<td>Determines the percentage of rows in a table that must change before the statistics on that table are stale and need to be regathered.</td>
</tr>
<tr>
<td></td>
<td>The valid domain for stale_percent is non-negative numbers. The default value is 10, which means that a table having more than 10% of changes is considered stale.</td>
</tr>
</tbody>
</table>

#### Exceptions

- **ORA-20000**: Object does not exist or insufficient privileges
- **ORA-20001**: Invalid or illegal input values

#### Usage Notes

- To run this procedure, you must connect as owner of the table or have the `ANALYZE ANY` system privilege.
- All arguments are of type `VARCHAR2` and values are enclosed in quotes, even when they represent numbers.
Examples

DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'CASCADE', 'DBMS_STATS.AUTO_CASCADE');
DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'ESTIMATE_PERCENT', '9');
DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'DEGREE', '99');

Example 162-17  Overriding Statistics Preferences

In this example, legacy scripts set ESTIMATE_PERCENT explicitly rather than using the recommended AUTO_SAMPLE_SIZE. Your goal is to prevent users from using these scripts to set preferences on the sh.costs table.

No preference for ESTIMATE_PERCENT is set for sh.costs or at the global level, so the preference defaults to AUTO_SAMPLE_SIZE:

SELECT DBMS_STATS.GET_PREFS ('ESTIMATE_PERCENT', 'sh','costs') AS "STAT_PREFS" FROM DUAL;

STAT_PREFS
----------
DBMS_STATS.AUTO_SAMPLE_SIZE

By default, Oracle Database accepts preferences that are passed to the statistics gathering procedures. To override these parameters, use SET_TABLE_PREFS to set the PREFERENCE_OVERRIDES_PARAMETER preference to TRUE for the costs table only:

EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'costs',
'PREFERENCE_OVERRIDES_PARAMETER', 'TRUE');

A user-created script attempts to set estimate_percent to 100 when gathering statistics for sh.costs.

EXEC DBMS_STATS.GATHER_TABLE_STATS('sh', 'costs', ESTIMATE_PERCENT=>100);

However, because PREFERENCE_OVERRIDES_PARAMETER is TRUE for this table, Oracle Database gathers statistics using AUTO_SAMPLE_SIZE, which is the default, rather than the specified value of 100.

See Also:

Oracle Database SQL Tuning Guide to learn how to set optimizer statistics preferences
162.7.122 SET_TABLE_STATS Procedure

This procedure creates artificial table statistics for testing purposes.

Syntax

```sql
DBMS_STATS.SET_TABLE_STATS (
    ownname VARCHAR2,
    tabname VARCHAR2,
    partname VARCHAR2 DEFAULT NULL,
    stattab VARCHAR2 DEFAULT NULL,
    statid VARCHAR2 DEFAULT NULL,
    numrows NUMBER DEFAULT NULL,
    numblks NUMBER DEFAULT NULL,
    avgrlen NUMBER DEFAULT NULL,
    flags NUMBER DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL,
    no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (
        get_param('NO_INVALIDATE')),
    cachedblk NUMBER DEFAULT NULL,
    cachehit NUMBER DEFAULT NULL,
    force BOOLEAN DEFAULT FALSE,
    im_imcu_count NUMBER DEFAULT NULL,
    im_block_count NUMBER DEFAULT NULL,
    scanrate NUMBER DEFAULT NULL);
```

Parameters

Table 162-133 SET_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>partname</td>
<td>Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.</td>
</tr>
<tr>
<td>stattab</td>
<td>Table in which to store the statistics. If stattab is NULL, then the database stores the statistics in the data dictionary.</td>
</tr>
<tr>
<td>statid</td>
<td>Identifier (optional) to associate with these statistics within stattab. This identifier is only relevant if stattab is not NULL.</td>
</tr>
<tr>
<td>numrows</td>
<td>Number of rows in the table or partition.</td>
</tr>
<tr>
<td>numblks</td>
<td>Number of blocks that the table or partition occupies.</td>
</tr>
<tr>
<td>avgrlen</td>
<td>Average row length for the table or partition.</td>
</tr>
<tr>
<td>flags</td>
<td>For internal use only. Do not set.</td>
</tr>
<tr>
<td>statown</td>
<td>Schema containing stattab (if different than ownname).</td>
</tr>
</tbody>
</table>
Table 162-133  (Cont.) SET_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_invalidate</td>
<td>The validation setting for dependent cursors. If set to TRUE, then the procedure does not invalidate the dependent cursors. If set to FALSE, then the procedure invalidates the dependent cursors immediately. Use DBMS_STATS.AUTO_INVALIDATE to let the database decide when to invalidate dependent cursors. This is the default. You can change the default using the SET_DATABASE_PREFS Procedure, SET_GLOBAL_PREFS Procedure, SET_SCHEMA_PREFS Procedure and SET_TABLE_PREFS Procedure.</td>
</tr>
<tr>
<td>cachedblk</td>
<td>For internal use only. Do not set.</td>
</tr>
<tr>
<td>cachehit</td>
<td>For internal use only. Do not set.</td>
</tr>
<tr>
<td>force</td>
<td>A flag that determines the behavior when statistics are locked. If TRUE, then the procedure sets the values even if the table statistics are locked. By default, the setting is FALSE.</td>
</tr>
<tr>
<td>im_imcu_count</td>
<td>The number of In-Memory Compression Units (IMCUs) in the table or partition.</td>
</tr>
<tr>
<td>im_block_count</td>
<td>The number of In-Memory blocks in the table or partition.</td>
</tr>
<tr>
<td>scanrate</td>
<td>The rate, in MB/s, at which the database scans external tables. This parameter is relevant only for external tables.</td>
</tr>
</tbody>
</table>

Security Model

To invoke this procedure you must be owner of the table, or have the ANAKYZE ANY privilege. For objects owned by SYS, you must be either the owner of the table, or have the ANAKYZE ANY DICTIONARY privilege or the SYSDBA privilege.

Exceptions

ORA-20000: Object does not exist or insufficient privileges
ORA-20001: Invalid input value
ORA-20002: Bad user statistics table; may need to upgrade it
ORA-20005: Object statistics are locked

Usage Notes

For testing purposes, you can manually create artificial statistics for a table, index, or the system using the DBMS_STATS.SET_*_STATS procedures. These procedures insert the artificial statistics into the data dictionary directly (when stattab is null) or into a user-created table.
The **DBMS_STATS.SET_*_STATS** procedures are intended for development testing only. Do not use them in a production database. If you set statistics in the data dictionary, then Oracle Database considers the set statistics as the “real” statistics, which means that statistics gathering jobs may not re-gather artificial statistics when they do not meet the criteria for staleness.

The most typical use cases for the **DBMS_STATS.SET_*_STATS** procedures are showing how execution plans change as the numbers of rows or blocks in a table change, or creating realistic statistics for temporary tables.

- The optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The database calculates the total cost of the operation by combining the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.

- The database maintains `cachedblk` and `cachehit` at all times. However, the database uses the corresponding caching statistics for optimization as part of the table and index statistics only when the user calls the `DBMS_STATS.GATHER_[TABLE/INDEX/SHEMA/DATABASE]_STATS` procedure for automatic mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. To prevent the user from utilizing inaccurate and unreliable data, the optimizer computes a “confidence factor” for each `cachehit` and a `cachedblk` for each object. If the confidence factor for the value meets confidence criteria, then the database uses this value; otherwise, the database uses defaults.

- The automatic maintenance algorithm for object caching statistics assumes that only one major database workload exists. The algorithm adjusts statistics to this workload, ignoring other “minor” workloads. If this assumption is false, then you must use manual mode for maintaining object caching statistics.

- The object caching statistics maintenance algorithm for automatic mode prevents you from using statistics in the following situations:
  - When not enough data has been analyzed, such as when an object has been recently created
  - When the database does not have one major workload, resulting in averages that do not correspond to real values

**See Also:**

*Oracle Database SQL Tuning Guide* to learn how to set artificial statistics
162.7.123 SHOW_EXTENDED_STATS_NAME Function

This function returns the name of the statistics entry that is created for the user-specified extension. It raises an error if no extension has been created.

Syntax

```
DBMS_STATS.SHOW_EXTENDED_STATS_NAME (  
    ownname    VARCHAR2,       
    tabname    VARCHAR2,       
    extension  VARCHAR2)       
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name of a table</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
<tr>
<td>extension</td>
<td>Can be either a column group or an expression. Suppose the specified table has two column c1, c2. An example column group can be &quot;(c1, c2)&quot; and an example expression can be &quot;(c1 + c2)&quot;.</td>
</tr>
</tbody>
</table>

Exceptions

- ORA-20000: Object does not exist or insufficient privileges
- ORA-20001: Error when processing extension

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

162.7.124 TRANSFER_STATS Procedure

This procedure transfers statistics for specified table(s) from a remote database specified by dblink to the local database.

The statistics at the source database are retained. It likewise transfers statistics-related structures such as synopses and DML monitoring information.

Syntax

```
DBMS_STATS.TRANSFER_STATS (  
    ownname     IN     VARCHAR2,       
    tabname     IN     VARCHAR2,       
    dblink      IN     VARCHAR2,       
    options     IN     NUMBER DEFAULT NULL)
```

Parameters

Table 162-135  TRANSFER_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Owner name of a table. If NULL all schemas in the database. If NULL, the procedure will transfer global preferences as well.</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table. If NULL, all tables in OWNNAME.</td>
</tr>
<tr>
<td>dblink</td>
<td>Database link name</td>
</tr>
<tr>
<td>options</td>
<td>By default the procedure does not transfer the global preferences. Specifying ADD_GLOBAL_PREFS copies global preferences.</td>
</tr>
</tbody>
</table>

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.

162.7.125 UNLOCK_PARTITION_STATS Procedure

This procedure enables the user to unlock statistics for a partition.

Syntax

```sql
DBMS_STATS.UNLOCK_PARTITION_STATS (
    ownname    VARCHAR2,
    tabname    VARCHAR2,
    partname   VARCHAR2);
```

Parameters

Table 162-136  UNLOCK_PARTITION_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema to unlock</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
<tr>
<td>partname</td>
<td>[Sub]Partition name</td>
</tr>
</tbody>
</table>

Usage Notes

To invoke this procedure you must be owner of the table, or you need the ANALYZE ANY privilege. For objects owned by SYS, you need to be either the owner of the table, or you need the ANALYZE ANY DICTIONARY privilege or the SYSDBA privilege.
162.7.126 UNLOCK_SCHEMA_STATS Procedure

This procedure unlocks the statistics on all the tables in schema.

Syntax

```
DBMS_STATS.UNLOCK_SCHEMA_STATS (  
    ownname    VARCHAR2);
```

Parameters

Table 162-137  UNLOCK_SCHEMA_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
</tbody>
</table>

Usage Notes

- To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
- When statistics on a table is locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The `SET_*`, `DELETE_*`, `IMPORT_*`, `GATHER_*` procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as `GATHER_SCHEMA_STATS`) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.
- Neither the `UNLOCK_SCHEMA_STATS` Procedure nor the `UNLOCK_TABLE_STATS` Procedure is designed to unlock statistics of corresponding partitions. When you invoke the `LOCK_TABLE_STATS` Procedure, it sets the statistics lock bit at the table level. In that case, you cannot gather statistics on dependent objects such as partitions and indexes. By the same token, if table statistics are locked, the dependents are locked and you do not need to explicitly invoke the `LOCK_PARTITION_STATS` Procedure.

162.7.127 UNLOCK_TABLE_STATS Procedure

This procedure unlocks the statistics on the table.

Syntax

```
DBMS_STATS.UNLOCK_TABLE_STATS (  
    ownname    VARCHAR2,  
    tabname    VARCHAR2);
```
Parameters

### Table 162-138  UNLOCK_TABLE_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>tabname</td>
<td>Name of the table</td>
</tr>
</tbody>
</table>

Usage Notes

- To invoke this procedure you must be owner of the table, or you need the `ANALYZE ANY` privilege. For objects owned by `SYS`, you need to be either the owner of the table, or you need the `ANALYZE ANY DICTIONARY` privilege or the `SYSDBA` privilege.
- When statistics on a table is locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The `SET_`, `DELETE_`, `IMPORT_`, `GATHER_*` procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as `GATHER_SCHEMA_STATS`) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.
- Neither the `UNLOCK_SCHEMA_STATS Procedure` nor the `UNLOCK_TABLE_STATS Procedure` is designed to unlock statistics of corresponding partitions. When you invoke the `LOCK_TABLE_STATS Procedure`, it sets the statistics lock bit at the table level. In that case, you cannot gather statistics on dependent objects such as partitions and indexes. By the same token, if table statistics are locked, the dependents are locked and you do not need to explicitly invoke the `LOCK_PARTITION_STATS Procedure`.

162.7.128 UPGRADE_STAT_TABLE Procedure

This procedure upgrades a user statistics table from an older version.

Syntax

```sql
DBMS_STATS.UPGRADE_STAT_TABLE (  
  ownname VARCHAR2,      
  stattab VARCHAR2);  
```

Parameters

### Table 162-139  UPGRADE_STAT_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownname</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>stattab</td>
<td>Name of the table</td>
</tr>
</tbody>
</table>
Exceptions

ORA-20000: Unable to upgrade table

Usage Notes

To invoke this procedure you need the privileges to drop and create a table.
With the `DBMS_STORAGE_MAP` package, you can communicate with the Oracle background process FMON to invoke mapping operations that populate mapping views. FMON communicates with operating and storage system vendor-supplied mapping libraries.

This chapter contains the following topics:

- **Overview**
- **Operational Notes**
- **Summary of DBMS_STORAGE_MAP Subprograms**

# 163.1 DBMS_STORAGE_MAP Overview

This terminology and descriptions will help you understand the `DBMS_STORAGE_MAP` API.

- **Mapping libraries**
  Mapping libraries help you map the components of I/O processing stack elements. Examples of I/O processing components include files, logical volumes, and storage array I/O targets. The mapping libraries are identified in `filemap.ora`.

- **Mapping files**
  A mapping file is a mapping structure that describes a file. It provides a set of attributes, including file size, number of extents that the file is composed of, and file type.

- **Mapping elements and sub-elements**
  A mapping element is the abstract mapping structure that describes a storage component within the I/O stack. Examples of elements include mirrors, stripes, partitions, raid5, concatenated elements, and disks—structures that are the mapping building blocks. A mapping sub-element describes the link between an element and the next elements in the I/O mapping stack.

- **Mapping file extents**
  A mapping file extent describes a contiguous chunk of blocks residing on one element. This includes the device offset, the extent size, the file offset, the type (data or parity), and the name of the element where the extent resides. In the case of a raw device or volume, the file is composed of only one file extent component. A mapping file extent is different from Oracle extents.

![See Also:](#)

- *Oracle Database Administrator's Guide* for more information
- *Oracle Database Reference* for `V$MAP` views, including `V$MAP_FILE`, `V$MAP_ELEMENT`, `V$MAP_SUBELEMENT`, `V$MAP_FILE_EXTENT`
163.2 DBMS_STORAGE_MAP Operational Notes

Invoking the MAP_ELEMENT, MAP_FILE, and MAP_ALL functions when mapping information already exists will refresh the mapping, if configuration IDs are supported. If configuration IDs are not supported, invoking these functions again will rebuild the mapping.

See Also:
Oracle Database Administrator’s Guide for a discussion of the configuration ID, an attribute of the element or file that is changed.

163.3 Summary of DBMS_STORAGE_MAP Subprograms

This table lists the DBMS_STORAGE_MAP subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP_ALL Function</td>
<td>Drops all mapping information in the shared memory of the instance</td>
</tr>
<tr>
<td>DROP_ELEMENT Function</td>
<td>Drops the mapping information for the element defined by elemname</td>
</tr>
<tr>
<td>DROP_FILE Function</td>
<td>Drops the file mapping information defined by filename</td>
</tr>
<tr>
<td>LOCK_MAP Procedure</td>
<td>Locks the mapping information in the shared memory of the instance</td>
</tr>
<tr>
<td>MAP_ALL Function</td>
<td>Builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements</td>
</tr>
<tr>
<td>MAP_ELEMENT Function</td>
<td>Builds mapping information for the element identified by elemname</td>
</tr>
<tr>
<td>MAP_FILE Function</td>
<td>Builds mapping information for the file identified by filename</td>
</tr>
<tr>
<td>MAP_OBJECT Function</td>
<td>Builds the mapping information for the Oracle object identified by the object name, owner, and type</td>
</tr>
<tr>
<td>RESTORE Function</td>
<td>Loads the entire mapping information from the data dictionary into the shared memory of the instance</td>
</tr>
<tr>
<td>SAVE Function</td>
<td>Saves information needed to regenerate the entire mapping into the data dictionary</td>
</tr>
<tr>
<td>UNLOCK_MAP Procedure</td>
<td>Unlocks the mapping information in the shared memory of the instance.</td>
</tr>
</tbody>
</table>
163.3.1 DROP_ALL Function

This function drops all mapping information in the shared memory of the instance.

Syntax

```sql
DBMS_STORAGE_MAP.DROP_ALL(
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 163-2 DROP_ALL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary_update</td>
<td>If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.</td>
</tr>
</tbody>
</table>

163.3.2 DROP_ELEMENT Function

This function drops the mapping information for the element defined by `elemname`.

Syntax

```sql
DBMS_STORAGE_MAP.DROP_ELEMENT(
    elemname          IN VARCHAR2,
    cascade           IN BOOLEAN,
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 163-3 DROP_ELEMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elemname</td>
<td>The element for which mapping information is dropped.</td>
</tr>
<tr>
<td>cascade</td>
<td>If TRUE, then DROP_ELEMENT is invoked recursively on all elements of the DAG defined by <code>elemname</code>, if possible.</td>
</tr>
<tr>
<td>dictionary_update</td>
<td>If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.</td>
</tr>
</tbody>
</table>

163.3.3 DROP_FILE Function

This function drops the file mapping information defined by `filename`.

Syntax

```sql
DBMS_STORAGE_MAP.DROP_FILE(
    filename          IN VARCHAR2,
    cascade           IN BOOLEAN,
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```
Parameters

Table 163-4  DROP_FILE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The file for which file mapping information is dropped.</td>
</tr>
<tr>
<td>cascade</td>
<td>If TRUE, then the mapping DAGs for the elements where the file resides are also dropped, if possible.</td>
</tr>
<tr>
<td>dictionary_update</td>
<td>If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.</td>
</tr>
</tbody>
</table>

163.3.4 LOCK_MAP Procedure

This procedure locks the mapping information in the shared memory of the instance.

This is useful when you need a consistent snapshot of the V$MAP tables. Without locking the mapping information, V$MAP_ELEMENT and V$MAP_SUBELEMENT, for example, may be inconsistent.

Syntax

```
DBMS_STORAGE_MAP.LOCK_MAP;
```

163.3.5 MAP_ALL Function

This function builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements. It obtains the latest mapping information because it explicitly synchronizes all mapping libraries.

Syntax

```
DBMS_STORAGE_MAP.MAP_ALL(
    max_num_fileext IN NUMBER DEFAULT 100,
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 163-5  MAP_ALL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_num_fileext</td>
<td>Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is 100; max_num_fileextent is an overloaded argument.</td>
</tr>
<tr>
<td>dictionary_update</td>
<td>If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.</td>
</tr>
</tbody>
</table>
Usage Notes
You must explicitly call MAP_ALL in a cold startup scenario.

163.3.6 MAP_ELEMENT Function

This function builds mapping information for the element identified by elemname. It may not obtain the latest mapping information if the element being mapped, or any one of the elements within its I/O stack (if cascade is TRUE), is owned by a library that must be explicitly synchronized.

Syntax

```sql
DBMS_STORAGE_MAP.MAP_ELEMENT(
    elemname          IN VARCHAR2,
    cascade           IN BOOLEAN,
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elemname</td>
<td>The element for which mapping information is built.</td>
</tr>
<tr>
<td>cascade</td>
<td>If TRUE, all elements within the elemname I/O stack DAG are mapped.</td>
</tr>
<tr>
<td>dictionary_update</td>
<td>If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.</td>
</tr>
</tbody>
</table>

163.3.7 MAP_FILE Function

This function builds mapping information for the file identified by filename. Use this function if the mapping of one particular file has changed. The Oracle database server does not have to rebuild the entire mapping.

Syntax

```sql
DBMS_STORAGE_MAP.MAP_FILE(
    filename           IN VARCHAR2,
    filetype           IN VARCHAR2,
    cascade            IN BOOLEAN,
    max_num_fileextent IN NUMBER DEFAULT 100,
    dictionary_update  IN BOOLEAN DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The file for which mapping information is built.</td>
</tr>
</tbody>
</table>
Table 163-7  (Cont.) MAP_FILE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filetype</td>
<td>Defines the type of the file to be mapped. It can be &quot;DATAFILE&quot;, &quot;SPFILE&quot;, &quot;TEMPFILE&quot;, &quot;CONTROLFILE&quot;, &quot;LOGFILE&quot;, or &quot;ARCHIVEFILE&quot;.</td>
</tr>
<tr>
<td>cascade</td>
<td>Should be <strong>TRUE</strong> only if a storage reconfiguration occurred. For all other instances, such as file resizing (either through an ALTER SYSTEM command or DML operations on extended files), cascade can be set to <strong>FALSE</strong> because the mapping changes are limited to the file extents only. If <strong>TRUE</strong>, mapping DAGs are also built for the elements where the file resides.</td>
</tr>
<tr>
<td>max_num_fileextent</td>
<td>Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is 100; max_num_fileextent is an overloaded argument.</td>
</tr>
<tr>
<td>dictionary_update</td>
<td><strong>If TRUE</strong>, mapping information in the data dictionary is updated to reflect the changes. The default value is <strong>TRUE</strong>; dictionary_update is an overloaded argument.</td>
</tr>
</tbody>
</table>

Usage Notes

This function may not obtain the latest mapping information if the file being mapped, or any one of the elements within its I/O stack (if cascade is **TRUE**), is owned by a library that must be explicitly synchronized.

163.3.8 MAP_OBJECT Function

This function builds the mapping information for the Oracle object identified by the object name, owner, and type.

Syntax

```
DBMS_STORAGE_MAP.MAP_OBJECT(
    objname IN VARCHAR2,
    owner  IN VARCHAR2,
    objtype IN VARCHAR2);
```

Parameters

Table 163-8  MAP_OBJECT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objname</td>
<td>The name of the object.</td>
</tr>
<tr>
<td>owner</td>
<td>The owner of the object.</td>
</tr>
<tr>
<td>objtype</td>
<td>The type of the object.</td>
</tr>
</tbody>
</table>
163.3.9 RESTORE Function

This function loads the entire mapping information from the data dictionary into the shared memory of the instance.

You can invoke **RESTORE** only after a **SAVE** operation. You must explicitly call **RESTORE** in a warm startup scenario.

**Syntax**

```
DBMS_STORAGE_MAP.RESTORE;
```

163.3.10 SAVE Function

This function saves information needed to regenerate the entire mapping into the data dictionary.

**Syntax**

```
DBMS_STORAGE_MAP.SAVE;
```

163.3.11 UNLOCK_MAP Procedure

This procedure unlocks the mapping information in the shared memory of the instance.

**Syntax**

```
DBMS_STORAGE_MAP.UNLOCK_MAP;
```
The **DBMS_STREAMS** package, one of a set of Oracle Streams packages, provides subprograms to convert **ANYDATA** objects into logical change record (LCR) objects, to return information about Oracle Streams attributes and Oracle Streams clients, and to annotate redo entries generated by a session with a binary tag.

This tag affects the behavior of a capture process, a propagation, or an apply process whose rules include specifications for these binary tags in redo entries or LCRs.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Summary of DBMS_STREAMS Subprograms**

### 164.1 DBMS_STREAMS Overview

This package provides subprograms to convert **ANYDATA** objects into logical change record (LCR) objects, to return information about Oracle Streams attributes and Oracle Streams clients, and to annotate redo entries generated by a session with a binary tag.

This tag affects the behavior of a capture process, a propagation, or an apply process whose rules include specifications for these binary tags in redo entries or LCRs.

**See Also:**

*Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Oracle Streams

### 164.2 DBMS_STREAMS Security Model

**PUBLIC** is granted **EXECUTE** privilege on this package.

**See Also:**

*Oracle Database Security Guide* for more information about user group **PUBLIC**

### 164.3 Summary of DBMS_STREAMS Subprograms

This table lists the **DBMS_STREAMS** subprograms and briefly describes them.
### Table 164-1  DBMS_STREAMS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPATIBLE_12_2 Function</td>
<td>Returns the DBMS_STREAMS.COMPATIBLE_12_2 constant</td>
</tr>
<tr>
<td>COMPATIBLE_12_1 Function</td>
<td>Returns the DBMS_STREAMS.COMPATIBLE_12_1 constant</td>
</tr>
<tr>
<td>COMPATIBLE_11_2 Function</td>
<td>Returns the DBMS_STREAMS.COMPATIBLE_11_2 constant</td>
</tr>
<tr>
<td>COMPATIBLE_11_1 Function</td>
<td>Returns the DBMS_STREAMS.COMPATIBLE_11_1 constant</td>
</tr>
<tr>
<td>COMPATIBLE_10_2 Function</td>
<td>Returns the DBMS_STREAMS.COMPATIBLE_10_2 constant</td>
</tr>
<tr>
<td>COMPATIBLE_10_1 Function</td>
<td>Returns the DBMS_STREAMS.COMPATIBLE_10_1 constant</td>
</tr>
<tr>
<td>COMPATIBLE_9_2 Function</td>
<td>Returns the DBMS_STREAMS.COMPATIBLE_9_2 constant</td>
</tr>
<tr>
<td>CONVERT_ANYDATA_TO_LCR_DDL Function</td>
<td>Converts a ANYDATA object to a SYS.LCR$$_DDL$$ RECORD object</td>
</tr>
<tr>
<td>CONVERT_ANYDATA_TO_LCR_ROW Function</td>
<td>Converts a ANYDATA object to a SYS.LCR$$_ROW$$ RECORD object</td>
</tr>
<tr>
<td>CONVERT_LCR_TO_XML Function</td>
<td>Converts a logical change record (LCR) encapsulated in a ANYDATA object into an XML object that conforms to the XML schema for LCRs</td>
</tr>
<tr>
<td>CONVERT_XML_TO_LCR Function</td>
<td>Converts an XML object that conforms to the XML schema for LCRs into a logical change record (LCR) encapsulated in a ANYDATA object</td>
</tr>
<tr>
<td>GET_INFORMATION Function</td>
<td>Returns information about various Oracle Streams attributes</td>
</tr>
<tr>
<td>GET_STREAMS_NAME Function</td>
<td>Returns the name of the invoker</td>
</tr>
<tr>
<td>GET_STREAMS_TYPE Function</td>
<td>Returns the type of the invoker</td>
</tr>
<tr>
<td>GET_TAG Function</td>
<td>Gets the binary tag for all redo entries generated by the current session</td>
</tr>
<tr>
<td>MAX_COMPATIBLE Function</td>
<td>Returns an integer that is greater than the highest possible compatibility constant for the current release of Oracle Database</td>
</tr>
<tr>
<td>SET_TAG Procedure</td>
<td>Sets the binary tag for all redo entries subsequently generated by the current session</td>
</tr>
</tbody>
</table>

**Note:**
The subprograms in this package do not commit.
164.3.1 COMPATIBLE_12_2 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_12_2` constant.

Syntax

```
DBMS_STREAMS.COMPATIBLE_12_2
RETURN INTEGER;
```

Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 12.2.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

See Also:

- `GET_COMPATIBLE` Member Function
- Oracle Streams Concepts and Administration for information about creating rules that discard changes that are not supported by Oracle Streams
- Oracle Database Reference and Oracle Database Upgrade Guide for more information about the `COMPATIBLE` initialization parameter

164.3.2 COMPATIBLE_12_1 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_12_1` constant.

Syntax

```
DBMS_STREAMS.COMPATIBLE_12_1
RETURN INTEGER;
```

Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 12.1.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.
See Also:

- GET_COMPATIBLE Member Function
- Oracle Streams Concepts and Administration for information about creating rules that discard changes that are not supported by Oracle Streams
- Oracle Database Reference and Oracle Database Upgrade Guide for more information about the COMPATIBLE initialization parameter

164.3.3 COMPATIBLE_11_2 Function

This function returns the DBMS_STREAMS.COMPATIBLE_11_2 constant.

Syntax

```sql
DBMS_STREAMS.COMPATIBLE_11_2
RETURN INTEGER;
```

Usage Notes

You can use this function with the GET_COMPATIBLE member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 11.2.0 compatibility in a database. You control the compatibility of an Oracle database using the COMPATIBLE initialization parameter.

See Also:

- GET_COMPATIBLE Member Function
- Oracle Streams Concepts and Administration for information about creating rules that discard changes that are not supported by Oracle Streams
- Oracle Database Reference and Oracle Database Upgrade Guide for more information about the COMPATIBLE initialization parameter

164.3.4 COMPATIBLE_11_1 Function

This function returns the DBMS_STREAMS.COMPATIBLE_11_1 constant.

Syntax

```sql
DBMS_STREAMS.COMPATIBLE_11_1
RETURN INTEGER;
```
Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 11.1.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

**See Also:**
- `GET_COMPATIBLE` Member Function
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

### 164.3.5 COMPATIBLE_10_2 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_10_2` constant.

**Syntax**

```sql
DBMS_STREAMS.COMPATIBLE_10_2
RETURN INTEGER;
```

**Usage Notes**

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 10.2.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

**See Also:**
- `GET_COMPATIBLE` Member Function
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter
164.3.6 COMPATIBLE_10_1 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_10_1` constant.

**Syntax**

```sql
DBMS_STREAMS.COMPATIBLE_10_1
RETURN INTEGER;
```

**Usage Notes**

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 10.1.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

---

164.3.7 COMPATIBLE_9_2 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_9_2` constant.

**Syntax**

```sql
DBMS_STREAMS.COMPATIBLE_9_2
RETURN INTEGER;
```

**Usage Notes**

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 9.2.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.
164.3.8 CONVERT_ANYDATA_TO_LCR_DDL Function

This function converts a ANYDATA object into a SYS.LCR$$_DDL_RECORD object.

Syntax

DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_DDL(
   source  IN  ANYDATA)
RETURN SYS.LCR$$_DDL_RECORD;

Parameters

Table 164-2 CONVERT_ANYDATA_TO_LCR_DDL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The ANYDATA object to be converted. If this object is not a DDL logical change record (DDL LCR), then the function raises an exception.</td>
</tr>
</tbody>
</table>

Usage Notes

You can use this function in a transformation created by the CREATE_TRANSFORMATION procedure in the DBMS_TRANSFORM package. Use the transformation you create when you add a subscriber for propagation of DDL LCRs from a ANYDATA queue to a SYS.LCR$$_DDL_RECORD typed queue.

See Also:
Oracle Database Advanced Queuing User's Guide

164.3.9 CONVERT_ANYDATA_TO_LCR_ROW Function

This function converts a ANYDATA object into a SYS.LCR$$_ROW_RECORD object.

Syntax

DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_ROW(
   source  IN  ANYDATA)
RETURN SYS.LCR$$_ROW_RECORD;
Parameters

Table 164-3  CONVERT_ANYDATA_TO_LCR_ROW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The ANYDATA object to be converted. If this object is not a row logical change record (row LCR), then the function raises an exception.</td>
</tr>
</tbody>
</table>

Usage Notes

You can use this function in a transformation created by the CREATE_TRANSFORMATION procedure in the DBMS_TRANSFORM package. Use the transformation you create when you add a subscriber for propagation of row LCRs from a ANYDATA queue to a SYS.LCR$_ROW_RECORD typed queue.

See Also:
Oracle Database Advanced Queuing User's Guide

164.3.10 CONVERT_LCR_TO_XML Function

This function converts a logical change record (LCR) encapsulated in a ANYDATA object into an XML object that conforms to the XML schema for LCRs. The LCR can be a row LCR or a DDL LCR.

See Also:
Oracle Streams Concepts and Administration for more information about the XML schema for LCRs

Syntax

DBMS_STREAMS.CONVERT_LCR_TO_XML(
    anylcr  IN  ANYDATA) RETURN SYS.XMLTYPE;

Parameters

Table 164-4  CONVERT_LCR_TO_XML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anylcr</td>
<td>The ANYDATA encapsulated LCR to be converted. If this object is not a ANYDATA encapsulated LCR, then the function raises an exception.</td>
</tr>
</tbody>
</table>
164.3.11 CONVERT_XML_TO_LCR Function

This function converts an XML object that conforms to the XML schema for logical change records (LCRs) into an LCR encapsulated in an ANYDATA object. The LCR can be a row or DDL LCR.

See Also:
Oracle Streams Concepts and Administration for more information about the XML schema for LCRs

Syntax

```
DBMS_STREAMS.CONVERT_XML_TO_LCR(
    xmldat  IN  SYS.XMLTYPE)
RETURN ANYDATA;
```

Parameters

Table 164-5  CONVERT_XML_TO_LCR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmldat</td>
<td>The XML LCR object to be converted. If this object does not conform to XML schema for LCRs, then the function raises an exception.</td>
</tr>
</tbody>
</table>

164.3.12 GET_INFORMATION Function

This function returns information about various Oracle Streams attributes.

Syntax

```
DBMS_STREAMS.GET_INFORMATION(
    name  IN  VARCHAR2)
RETURN ANYDATA;
```
### Parameters

#### Table 164-6  GET_INFORMATION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| name      | The type of information you want to retrieve. Currently, the following names are available:  
  - **SENDER**: Returns the name of the sender for the current logical change record (LCR) from its AQ message properties. This function is called inside a procedure DML handler, a DDL handler, an error handler, or a message handler. Returns NULL if called outside of an apply handler. The return value is to be interpreted as a VARCHAR2.  
  - **CONSTRAINT_NAME**: Returns the name of the constraint that was violated for an LCR that raised an error. This function is called inside a procedure DML handler or error handler for an apply process. Returns NULL if called outside of a procedure DML handler or error handler. The return value is to be interpreted as a VARCHAR2. |

#### 164.3.13 GET_STREAMS_NAME Function

This function gets the Oracle Streams name of the invoker if the invoker is one of the following Oracle Streams types: **CAPTURE**, **APPLY**, or **ERROR_EXECUTION**.

If the invoker is not one of these types, then this function returns a NULL.

**Syntax**

```
DBMS_STREAMS.GET_STREAMS_NAME
RETURN VARCHAR2;
```

**Usage Notes**

You can use this function in rule conditions, rule-based transformations, apply handlers, and error handlers. For example, if you use one error handler for multiple apply processes, then you can use the **GET_STREAMS_NAME** function to determine the name of the apply process that raised the error.

#### 164.3.14 GET_STREAMS_TYPE Function

This function gets the Oracle Streams type of the invoker and returns one of the following types: **CAPTURE**, **APPLY**, or **ERROR_EXECUTION**.

If the invoker is not one of these types, then this function returns a NULL.

**Syntax**

```
DBMS_STREAMS.GET_STREAMS_TYPE
RETURN VARCHAR2;
```

**Usage Notes**

This function can be used in rule conditions, rule-based transformations, apply handlers, and error handlers. For example, you can use the **GET_STREAMS_TYPE** function to instruct a procedure DML handler to operate differently if it is processing.
messages from the error queue (ERROR_EXECUTION type) instead of the apply process’s queue (APPLY type).

164.3.15 GET_TAG Function

This function gets the binary tag for all redo entries generated by the current session.

**Note:**

- To execute this function, a user must be granted either EXECUTE_CATALOG_ROLE or EXECUTE privilege on the DBMS STREAMS ADM package.
- Instead of using the DBMS_STREAMS.GET_TAG function, Oracle recommends that you use the DBMS_STREAMS_ADM.GET_TAG function. See GET_TAG Function.

**See Also:**

Oracle Streams Replication Administrator’s Guide for more information about tags

### Syntax

```sql
DBMS_STREAMS.GET_TAG
RETURN RAW;
```

### Examples

The following example illustrates how to display the current logical change record (LCR) tag as output:

```sql
SET SERVEROUTPUT ON
DECLARE
    raw_tag RAW(2000);
BEGIN
    raw_tag := DBMS_STREAMS.GET_TAG();
    DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/
```

You can also display the value by querying the DUAL view:

```sql
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

164.3.16 MAX_COMPATIBLE Function

This function returns an integer that is greater than the highest possible compatibility constant for the current release of Oracle Database.

### Syntax

```sql
DBMS_STREAMS.MAX_COMPATIBLE
RETURN INTEGER;
```
Usage Notes

You can use this function with the \texttt{GET\_COMPATIBLE} member function for logical change records (LCRs) to specify behavior based on compatibility.

The \texttt{MAX\_COMPATIBLE} function always returns the maximum compatibility for the release of Oracle Database on which it is run. Therefore, when you use this function in rule conditions, the rule conditions do not need to be changed when you upgrade to a later release of Oracle Database.

\textbf{See Also:}

- \texttt{GET\_COMPATIBLE} Member Function
- \textit{Oracle Streams Concepts and Administration} for information about creating rules that discard changes that are not supported by Oracle Streams
- \textit{Oracle Database Reference} and \textit{Oracle Database Upgrade Guide} for more information about the \texttt{COMPATIBLE} initialization parameter

164.3.17 SET\_TAG Procedure

This procedure sets the binary tag for all redo entries subsequently generated by the current session.

Each redo entry generated by DML or DDL statements in the current session has this tag. This procedure affects only the current session.

\textbf{Note:}

- To execute this procedure, a user must be granted either \texttt{EXECUTE\_CATALOG\_ROLE} or \texttt{EXECUTE} privilege on the \texttt{DBMS\_STREAMS\_ADM} package.
- Instead of using the \texttt{DBMS\_STREAMS\_SET\_TAG} procedure, Oracle recommends that you use the \texttt{DBMS\_STREAMS\_ADM\_SET\_TAG} procedure. See \texttt{SET\_TAG Procedure}.

\textbf{See Also:}

\textit{Oracle Streams Replication Administrator's Guide} for more information about tags

\textbf{Syntax}

\begin{verbatim}
DBMS\_STREAMS\_SET\_TAG(
    tag IN RAW DEFAULT NULL);
\end{verbatim}
Parameters

Table 164-7  SET_TAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>The binary tag for all subsequent redo entries generated by the current session. A raw value is a sequence of bytes, and a byte is a sequence of bits. By default, the tag for a session is NULL. The size limit for a tag value is 2000 bytes.</td>
</tr>
</tbody>
</table>

Usage Notes

To set the tag to the hexadecimal value of '17' in the current session, run the following procedure:

EXEC DBMS_STREAMS.SET_TAG(tag => HEXTORAW('17'));

The following are considerations for the SET_TAG procedure:

- This procedure is not transactional. That is, the effects of SET_TAG cannot be rolled back.
- If the SET_TAG procedure is run to set a non-NULL session tag before a data dictionary build has been performed on the database, then the redo entries for a transaction that started before the dictionary build might not include the specified tag value for the session. Therefore, perform a data dictionary build before using the SET_TAG procedure in a session. A data dictionary build happens when the DBMS_CAPTURE_ADM.BUILD procedure is run. The BUILD procedure can be run automatically when a capture process is created.

See Also:

BUILD Procedure
DBMS_STREAMS_ADM

The DBMS_STREAMS_ADM package, one of a set of Oracle Streams packages, provides subprograms for configuring Oracle Streams environments.

This package also includes subprograms for adding and removing simple rules for capture, propagation, apply, and dequeue at the table, schema, and database level. This package also includes subprograms for configuring and managing XStream outbound servers and inbound servers.

This chapter contains the following topics:

- **Overview**
- **Deprecated Subprograms**
- **Security Model**
- **Procedures That Create Rules for Oracle Streams Clients and XStream Clients**
- **Procedures That Configure an Oracle Streams Environment**
- **Summary of DBMS_STREAMS_ADM Subprograms**

165.1 DBMS_STREAMS_ADM Overview

The DBMS_STREAMS_ADM package, one of a set of Oracle Streams packages, provides subprograms for configuring an Oracle Streams environment.

This package also includes subprograms for adding and removing simple rules for capture, propagation, apply, and dequeue at the table, schema, and database level. These rules support logical change records (LCRs), which include row LCRs and data definition language (DDL) LCRs. This package also contains subprograms for creating message rules for specific message types. This package also contains subprograms for creating queues, and for managing Oracle Streams metadata, such as data dictionary information.

If you require more sophisticated rules, then refer to DBMS_RULE package.

See Also:

- *Oracle Streams Concepts and Administration, Oracle Streams Replication Administrator’s Guide* for more information about this package and Oracle Streams
- DBMS_RULE
165.2 DBMS_STREAMS_ADM Deprecated Subprograms

These DBMS_STREAMS_ADM subprograms have been deprecated.

**Note:**

Oracle recommends that you do not use deprecated subprograms. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 10g Release 2 and later:

- **MAINTAIN_SIMPLE_TABLESPACE**
  
  This procedure is replaced by the MAINTAIN_SIMPLE_TTS procedure.

  **See Also:**

  MAINTAIN_SIMPLE_TTS Procedure

- **MAINTAIN_TABLESPACES**
  
  This procedure is replaced by the MAINTAIN_TTS procedure.

  **See Also:**

  MAINTAIN_TTS Procedure

165.3 DBMS_STREAMS_ADM Security Model

Security on this package can be controlled by either granting EXECUTE on this package to selected users or roles, or by granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

A user is associated with each Oracle Streams client.
**Note:**

The user must be granted additional privileges to perform some administrative tasks using the subprograms in this package, such as creating a synchronous capture. If additional privileges are required for a subprogram, then the privileges are documented in the section that describes the subprogram.

---

**Oracle Streams Administrator**

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

---

**See Also:**

*Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

---

**Capture User**

The following procedures can create a capture process:

- ADD_GLOBAL_RULES Procedure
- ADD_SCHEMA_RULES Procedure
- ADD_SUBSET_RULES Procedure
- ADD_TABLE_RULES Procedure

The following procedures can create a synchronous capture:

- ADD_SUBSET_RULES Procedure
- ADD_TABLE_RULES Procedure

If one of these procedures creates a capture process or a synchronous capture, then it configures the current user as the capture user. The capture user is the user in whose security domain a capture process or synchronous capture captures changes that satisfy its rule set(s) and runs custom rule-based transformations configured for these rules. This user must have the necessary privileges to capture changes. The procedure grants the capture user ENQUEUE privilege on the queue used by the capture process or synchronous capture and configures the user as a secure queue user of the queue.

---

**See Also:**

CREATE_CAPTURE Procedure and CREATE_SYNC_CAPTURE Procedure for information about the privileges required to capture changes (refer to the capture_user parameter)
Propagation User

The following procedures can create a propagation:

- `ADD_GLOBAL_PROPAGATION_RULES Procedure`
- `ADD_MESSAGE_PROPAGATION_RULE Procedure`
- `ADD_SCHEMA_PROPAGATION_RULES Procedure`
- `ADD_SUBSET_PROPAGATION_RULES Procedure`
- `ADD_TABLE_PROPAGATION_RULES Procedure`

When a propagation is created, a propagation job also might be created. If a propagation job is created when one of these procedures is run, then the user who runs the procedure owns the propagation job. Each propagation job is an Oracle Scheduler job. You can adjust the schedule of a propagation job using Oracle Scheduler.

**Note:**

- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users might or might not be the same.
- For a propagation to work properly, the owner of the source queue must have the necessary privileges to propagate messages.

**See Also:**

- `CREATE_PROPAGATION Procedure` for more information about the required privileges
- "* Procedures That Create Rules for Oracle Streams Clients and XStream Clients" for information about when a propagation job is created

Apply User for an Oracle Streams Apply Process

The following procedures can create an apply process:

- `ADD_GLOBAL_RULES Procedure`
- `ADD_MESSAGE_RULE Procedure`
- `ADD_SCHEMA_RULES Procedure`
- `ADD_SUBSET_RULES Procedure`
- `ADD_TABLE_RULES Procedure`

If one of these procedures creates an apply process, then it configures the current user as the apply user. For an apply process, the apply user is the user in whose security domain an apply process dequeues messages that satisfy its rule sets.
An apply user applies messages directly to database objects, runs custom rule-based transformations configured for apply process rules, and runs apply handlers configured for the apply process. This user must have the necessary privileges to apply changes. The procedure grants the apply user `DEQUEUE` privilege on the queue used by the apply process and configures the user as a secure queue user of the queue.

See Also:

- `CREATE_APPLY Procedure` for information about the privileges required to apply changes (refer to the `apply_user` parameter)

Apply User for an XStream Inbound Server

The following procedures can create an XStream inbound server:

- `ADD_GLOBAL_RULES Procedure`
- `ADD_SCHEMA_RULES Procedure`
- `ADD_SUBSET_RULES Procedure`
- `ADD_TABLE_RULES Procedure`

Note:

- Oracle recommends using procedures in the `DBMS_XSTREAM_ADM` package to create or add rules to an XStream inbound server. See `DBMS_XSTREAM_ADM`.
- These procedures cannot create an outbound server.

If the `streams_name` parameter is set to `NULL` and no relevant apply process, inbound server, or outbound server exists, then the procedure creates an apply process automatically with a system-generated name.

The apply process remains an apply process if it receives captured logical change records (LCRs) from a capture process. The apply process can become an inbound server if an XStream client application attaches to it before it receives captured LCRs from a capture process. After the initial contact, an apply process cannot be changed into an inbound server, and an inbound server cannot be changed into an apply process.

If one of these procedures creates an inbound server, then it configures the current user as the apply user. The apply user is the user in whose security domain an XStream client application attaches to an Oracle database.

An apply user applies changes directly to database objects, runs custom rule-based transformations configured for inbound server rules, and runs apply handlers configured for the inbound server. This user must have the necessary privileges to apply changes. The procedure grants the apply user `DEQUEUE` privilege on the queue used by the inbound server and configures the user as a secure queue user.
Each inbound server must have a unique name. The name cannot be used by an apply process, outbound server, or messaging client in the same database, and the name cannot be used by another inbound server in the same database.

If a relevant apply process, inbound server, or outbound server exists, then the procedure does not create an inbound server. Instead, the procedure uses the relevant apply process, inbound server, or outbound server. If the streams_name parameter specifies an existing apply process, inbound server, or outbound server, then the specified client is used.

When streams_name parameter is NULL and the streams_type parameter is set to apply, the relevant apply process, inbound server, or outbound server is identified in one of the following ways:

- If one existing apply process or outbound server has the source database specified in the source_database parameter and uses the queue specified in the queue_name parameter, then the procedure uses this apply process or outbound server.
- If the source_database parameter is set to NULL and one existing apply process, inbound server, or outbound server is using the queue specified in the queue_name parameter, then the procedure uses this apply process, inbound server, or outbound server.

If the streams_name parameter is set to NULL and multiple relevant apply processes, inbound servers, or outbound servers exist, then the procedure raises an error.

---

**Note:**

Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

**Messaging Client User**

The following procedures can create a messaging client:

- ADD_GLOBAL_RULES Procedure
- ADD_MESSAGE_RULE Procedure
- ADD_SCHEMA_RULES Procedure
- ADD_SUBSET_RULES Procedure
- ADD_TABLE_RULES Procedure

If one of these procedures creates a messaging client, then the user who runs this procedure is granted the privileges to dequeue from the queue using the messaging client. The procedure configures this user as a secure queue user of the queue, and only this user can use the messaging client.
165.4 Procedures That Create Rules for Oracle Streams Clients and XStream Clients

Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients. XStream clients include XStream outbound servers and inbound servers. Some of the procedures in the `DBMS_STREAMS_ADM` package add rules to the rule sets of Oracle Streams clients and XStream clients. The rules can pertain to changes in the redo log, to data manipulation language (DML) changes made to a table, to logical change records (LCRs), or to user messages.

An LCR represents either a row change that results from a DML operation or a data definition language (DDL) change. An LCR that represents a row change is a row LCR, and an LCR that represents a DDL change is a DDL LCR. LCRs can either represent changes that were captured by a capture process or a synchronous capture, or they can represent changes created by a user or application. A user message is a custom message that is based on a user-defined type and created by users or applications.

A capture process, propagation, apply process, messaging client, outbound server, or inbound server can have both positive and negative rule sets. A synchronous capture can have only a positive rule set.

For all of the procedures except the ones that create subset rules, and for all clients except synchronous captures, you use the `inclusion_rule` parameter to specify the type of rule set (either positive or negative) for the created rules. If the client does not have a rule set of the specified type, then a rule set is created automatically, and the rules are added to the rule set. Other rules in an existing rule set for the client are not affected. Additional rules can be added to a rule set using either the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package. If an client has both a positive and a negative rule set, then the negative rule set is always evaluated first.

**Note:**

Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.

**See Also:**

*Oracle Streams Concepts and Administration* for more information about how rules are used in Oracle Streams

Capture Process Rules for Changes in the Redo Log

The following procedures add rules to a rule set of a capture process when you specify *capture* for the `streams_type` parameter:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all changes made to a source database. See `ADD_GLOBAL_RULES` Procedure.
- The **ADD_SCHEMA_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for changes made to a specified schema. See **ADD_SCHEMA_RULES Procedure**.

- The **ADD_SUBSET_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for DML changes made to a subset of rows in a specified table. See **ADD_SUBSET_RULES Procedure**.

- The **ADD_TABLE_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for changes made to a specified table. See **ADD_TABLE_RULES Procedure**.

If one of these procedures adds rules to the positive rule set for a capture process, then the capture process captures row changes resulting from DML changes, or DDL changes, or both from a source database and enqueues these changes into the specified queue. If one of these procedures adds rules to the negative rule set for a capture process, then the capture process discards row changes, or DDL changes, or both from a source database.

A capture process can capture changes locally at a source database or remotely at a downstream database. Therefore, for capture process rules, you should execute the procedure either at the source database or at a downstream database.

If the capture process is a local capture process, or if the capture process is a downstream capture process that uses a database link to the source database, then these procedures automatically prepare the appropriate database objects for instantiation:

- **ADD_GLOBAL_RULES** invokes the **PREPARE_GLOBAL_INSTANTIATION** procedure in the **DBMS_CAPTURE_ADM** package at the source database.

- **ADD_SCHEMA_RULES** invokes the **PREPARE_SCHEMA_INSTANTIATION** procedure in the **DBMS_CAPTURE_ADM** package at the source database.

- **ADD_SUBSET_RULES** and **ADD_TABLE_RULES** invoke the **PREPARE_TABLE_INSTANTIATION** procedure in the **DBMS_CAPTURE_ADM** package at the source database.

These procedures also enable supplemental logging for the primary key, unique key, foreign key, and bitmap index columns in the tables prepared for instantiation. The primary key columns are unconditionally logged. The unique key, foreign key, and bitmap index columns are conditionally logged.

If the capture process is a downstream capture process that does not use a database link to the source database, then you must prepare the appropriate objects for instantiation and specify the necessary supplemental logging manually at the source database.

If one of these procedures is executed at a downstream database, then you specify the source database using the **source_database** parameter, and the specified capture process must exist. The procedure cannot create a capture process if it is run at a downstream database. You can create a capture process at a downstream database using the **CREATE_CAPTURE** procedure in the **DBMS_CAPTURE_ADM** package.
Synchronous Capture Rules for DML Changes to Tables

The following procedures add rules to the rule set of a synchronous capture when you specify `sync_capture` for the `streams_type` parameter:

- The `ADD_SUBSET_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for DML changes made to a subset of rows in a specified table. See `ADD_SUBSET_RULES Procedure`.
- The `ADD_TABLE_RULES` procedure adds a rule whose rule condition evaluates to `TRUE` for DML changes made to a specified table. See `ADD_TABLE_RULES Procedure`.

If one of these procedures adds rules to the positive rule set for a synchronous capture, then the synchronous capture captures row changes resulting from DML changes to the table at the source database and enqueues these changes into the specified queue. A synchronous capture cannot have a negative rule set.

A synchronous capture captures changes locally at the database where it is configured. This database is the source database for changes captured by the synchronous capture. Therefore, for synchronous capture rules, you should execute the procedure at the source database.

These procedures automatically prepare the appropriate tables for instantiation by invoking the `PREPARE_SYNC_INSTANTIATION` function in the `DBMS_CAPTURE_ADM` package at the source database.

Note:

- A synchronous capture ignores rules in its rule set that were created by a procedure other than `ADD_SUBSET_RULES` or `ADD_TABLE_RULES`.
- When the `ADD_TABLE_RULES` or the `ADD_SUBSET_RULES` procedure adds rules to a synchronous capture rule set, the procedure must obtain an exclusive lock on the specified table. If there are outstanding transactions on the specified table, then the procedure waits until it can obtain a lock.

Propagation Rules for LCRs

The following procedures add propagation rules for LCRs to a rule set of a propagation:

- The `ADD_GLOBAL_PROPAGATION_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all LCRs in a source queue. See `ADD_GLOBAL_PROPAGATION_RULES Procedure`.
- The `ADD_SCHEMA_PROPAGATION_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in a source queue containing changes made to a specified schema. See `ADD_SCHEMA_PROPAGATION_RULES Procedure`.

Summary of DBMS_CAPTURE_ADM Subprograms for more information about the `CREATE_CAPTURE` procedure and the procedures that prepare database objects for instantiation.
The **ADD_SUBSET_PROPAGATION_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for row LCRs in a source queue containing the results of DML changes made to a subset of rows in a specified table. See "**ADD_SUBSET_PROPAGATION_RULES Procedure**".

The **ADD_TABLE_PROPAGATION_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for LCRs in a source queue containing changes made to a specified table. See "**ADD_TABLE_PROPAGATION_RULES Procedure**".

If one of these procedures adds rules to the positive rule set for the propagation, then the rules specify that the propagation propagates LCRs in a source queue to a destination queue. If one of these procedures adds rules to the negative rule set for the propagation, then the rules specify that the propagation discards LCRs in a source queue. When you create rules with one of these procedures, and you specify a value for the **source_database** parameter, then the rules include conditions for the specified source database.

**Propagation Rules for User Messages**

The **ADD_MESSAGE_PROPAGATION_RULE** procedure adds a message rule to a rule set of a propagation. If this procedure adds a rule to the positive rule set for the propagation, then the rule specifies that the propagation propagates the user messages of a specific message type that evaluate to **TRUE** for the rule condition from a source queue to a destination queue. If this procedure adds a rule to the negative rule set for the propagation, then the rule specifies that the propagation discards the user messages in a source queue of a specific message type that evaluate to **TRUE** for the rule condition. This procedure generates a rule name for the rule.

**Apply Process Rules for LCRs**

The following procedures add rules to a rule set of an apply process when you specify apply for the **streams_type** parameter and an apply process for the **streams_name** parameter:

- The **ADD_GLOBAL_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for all LCRs in the apply process's queue. See "**ADD_GLOBAL_RULES Procedure**".

- The **ADD_SCHEMA_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for LCRs in the apply process's queue containing changes made to a specified schema. See "**ADD_SCHEMA_RULES Procedure**".

- The **ADD_SUBSET_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for row LCRs in the apply process's queue containing the results of DML changes made to a subset of rows in a specified table. See "**ADD_SUBSET_RULES Procedure**".

- The **ADD_TABLE_RULES** procedure adds rules whose rule condition evaluates to **TRUE** for LCRs in the apply process's queue containing changes made to a specified table. See "**ADD_TABLE_RULES Procedure**".
If one of these procedures adds rules to the positive rule set for the apply process, then the rules specify that the apply process applies LCRs in its queue. If one of these procedures adds rules to the negative rule set for the apply process, then the rules specify that the apply process discards LCRs in its queue. For apply process rules, you should execute these procedures at the destination database.

Changes applied by an apply process created by one of these procedures generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to alter the tag value after the apply process is created, if necessary.

An apply process can apply captured LCRs from only one source database. If one of these procedures creates an apply process, then specify the source database for the apply process using the source_database parameter. If the source_database parameter is NULL, and one of these procedures creates an apply process, then the source database name of the first LCR received by the apply process is used for the source database.

The rules in the apply process rule sets determine which messages are dequeued by the apply process. When you create rules with one of these procedures, and you specify a value for the source_database parameter, then the rules include conditions for the specified source database. If the apply process dequeues an LCR with a source database that is different from the source database for the apply process, then an error is raised. In addition, when adding rules to an existing apply process, the database specified in the source_database parameter cannot be different from the source database for the apply process. You can determine the source database for an apply process by querying the DBA_APPLY_PROGRESS data dictionary view.

An apply process created by one of these procedures can apply messages only at the local database and can apply only captured messages. To create an apply process that applies messages at a remote database or an apply process that applies user messages, use the CREATE_APPLY procedure in the DBMS_APPLY_ADM package.

You can also use the DBMS_APPLY_ADM.CREATE_APPLY procedure to specify nondefault values for the apply_capture, apply_user, apply_database_link, and apply_tag parameters when you run that procedure. You can use one of the procedures in the DBMS_STREAMS_ADM package to add rules to a rule set used by the apply process after you create it.

See Also:

- "ALTER_APPLY Procedure"
- "CREATE_APPLY Procedure"

Apply Process Rules for User Messages

The ADD_MESSAGE_RULE procedure adds a message rule to a rule set of an apply process when you specify apply for the streams_type parameter. For an apply process rule, you should execute this procedure at the destination database.

If this procedure adds a rule to the positive rule set for an apply process, then the apply process dequeues user messages of a specific message type that satisfy the apply process rule and sends these messages to its message handler. If no message handler is specified for the apply process, then use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to set the message handler. If this procedure adds a rule to the negative rule set for an apply
process, then the apply process discards user messages of a specific message type that satisfy the apply process rule.

See Also:

- ADD_MESSAGE_RULE Procedure
- ALTER_APPLY Procedure

Messaging Client Rules for LCRs

The following procedures add rules to a rule set of a messaging client when you specify dequeue for the streams_type parameter:

- The ADD GLOBAL RULES procedure adds rules whose rule condition evaluates to TRUE for all LCRs in the messaging client's queue. See "ADD_GLOBAL_RULES Procedure".
- The ADD SCHEMA RULES procedure adds rules whose rule condition evaluates to TRUE for LCRs in the messaging client's queue containing changes made to a specified schema. See "ADD_SCHEMA_RULES Procedure".
- The ADD SUBSET RULES procedure adds rules whose rule condition evaluates to TRUE for row LCRs in the messaging client's queue containing the results of DML changes made to a subset of rows in a specified table. See "ADD_SUBSET_RULES Procedure".
- The ADD TABLE RULES procedure adds rules whose rule condition evaluates to TRUE for LCRs in the messaging client's queue containing changes made to a specified table. See "ADD_TABLE_RULES Procedure".

If one of these procedures adds rules to the positive rule set for a messaging client, then the messaging client can dequeue persistent row LCRs, or DDL LCRs, or both that originated at the source database that matches the source_database parameter. If one of these procedures adds rules to the negative rule set for a messaging client, then the messaging client discards persistent row LCRs, or DDL LCRs, or both that originated at the source database that matches the source_database parameter. You should execute these procedures at the database where you want to dequeue the messages with the messaging client.

Messaging Client Rules for User Messages

The ADD MESSAGE RULE procedure adds a message rule to a rule set of a messaging client when you specify dequeue for the streams_type parameter. You should execute this procedure at the database that will dequeue messages.

If this procedure adds a rule to the positive rule set for a messaging client, then the messaging client dequeues user messages of a specific message type that satisfy the message rule. If this procedure adds a rule to the negative rule set for a messaging client, then the messaging client discards user messages of a specific message type that satisfy the message rule.
XStream Outbound Server Rules for LCRs

When you specify `apply` for the `streams_type` parameter and an XStream outbound server for the `streams_name` parameter, the following procedures add rules to a rule set of the specified outbound server:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for all LCRs.
- The `ADD_SCHEMA_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for LCRs that contain changes made to a specified schema.
- The `ADD_SUBSET_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for row LCRs that contain the results of DML changes made to a subset of rows in a specified table.
- The `ADD_TABLE_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for LCRs that contain changes made to a specified table.

These rules are evaluated against LCRs in the outbound server's queue.

If one of the preceding procedures adds rules to the positive rule set for the outbound server, then the rules specify that the outbound server sends LCRs in its queue to the XStream client application. If one of these procedures adds rules to the negative rule set for the outbound server, then the rules specify that the outbound server discards LCRs in its queue. For outbound server rules, execute these procedures at the database to which the XStream client application attaches.

An outbound server can process captured LCRs from only one source database. The source database is the database where the changes originated. If one of these procedures adds rules to the rule set of an outbound server, then specify the source database for the outbound server using the `source_database` parameter.

The rules in the outbound server's rule sets determine which LCRs are dequeued by the outbound server. When you create rules with one of these procedures, and you specify a value for the `source_database` parameter, then the rules include conditions for the specified source database. If the outbound server dequeues an LCR with a source database that is different from the source database for the outbound server, then an error is raised. In addition, when adding rules to an existing outbound server, the database specified in the `source_database` parameter cannot be different from the source database for the outbound server. You can determine the source database for an outbound server by querying the `DBA_XSTREAM_OUTBOUND` data dictionary view.

**Note:**

These procedures cannot create an XStream outbound server. You can use one of the procedures in the `DBMS_STREAMS_ADM` package to add rules to a rule set used by the outbound server after you create it.
See Also:

*Oracle Database XStream Guide* for information about creating an outbound server

XStream Inbound Server Rules for LCRs

When you specify `apply` for the `streams_type` parameter and an XStream inbound server for the `streams_name` parameter, the following procedures add rules to a rule set of the specified inbound server:

- **The ADD_GLOBAL_RULES procedure** adds rules whose rule conditions evaluate to `TRUE` for all LCRs sent to the inbound server.

- **The ADD_SCHEMA_RULES procedure** adds rules whose rule conditions evaluate to `TRUE` for LCRs sent to the inbound server that contain changes made to a specified schema.

- **The ADD_SUBSET_RULES procedure** adds rules whose rule condition evaluates to `TRUE` for row LCRs sent to the inbound server that contain the results of data definition language (DML) changes made to a subset of rows in a specified table.

- **The ADD_TABLE_RULES procedure** adds rules whose rule condition evaluates to `TRUE` for LCRs sent to the inbound server that contain changes made to a specified table.

If one of the preceding procedures adds rules to the positive rule set for the inbound server, then the rules specify that the inbound server applies LCRs sent to it by the XStream client application. If one of these procedures adds rules to the negative rule set for the inbound server, then the rules specify that the inbound server discards LCRs sent to it by the XStream client application. For inbound server rules, execute these procedures at the database to which the XStream client application attaches. If an inbound server has no rule sets, then it applies all of the LCRs sent to it by the XStream client application.

Changes applied by an inbound server created by one of these procedures generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package to alter the tag value after the inbound server is created, if necessary.

The rules in the XStream inbound server rule sets determine which LCRs are either applied or discarded after the LCRs are received from the XStream client application. An inbound server can only process LCRs sent from an XStream client application.

When one of these procedures creates rules for an inbound server, the procedure ignores the `source_database` parameter.
Note:

If the name specified in the streams_name parameter does not exist, then these procedures always create an apply process. The apply process remains an apply process if it receives captured LCRs from a capture process. The apply process can become an inbound server if an XStream client application attaches to it before it receives LCRs from a capture process. After the initial contact, an apply process cannot be changed into an inbound server, and an inbound server cannot be changed into an apply process.

See Also:

Oracle Database XStream Guide for information about creating an inbound server

165.5 Procedures That Configure an Oracle Streams Environment

These procedures in this package configure an environment that is maintained by Oracle Streams.

- **MAINTAIN_CHANGE_TABLE Procedure** configures an Oracle Streams environment that records in a change table the data manipulation language (DML) changes made to a source table. Optionally, this procedure can also configure one-way replication of the table from the source database to the destination database.

- **MAINTAIN_GLOBAL Procedure** configures an Oracle Streams environment that replicates changes at the database level between two databases.

- **MAINTAIN_SCHEMAS Procedure** configures an Oracle Streams environment that replicates changes to specified schemas between two databases.

- **MAINTAIN_SIMPLE_TTS Procedure** clones a simple tablespace from a source database at a destination database and configures an Oracle Streams environment that replicates changes to specified tablespace between these two databases.

- **MAINTAIN_TABLES Procedure** configures an Oracle Streams environment that replicates changes to specified tables between two databases.

- **MAINTAIN_TTS Procedure** clones a set of tablespaces from a source database at a destination database and configures an Oracle Streams environment that replicates changes to specified tablespaces between these two databases.

- **PRE_INSTANTIATION_SETUP Procedure and POST_INSTANTIATION_SETUP Procedure**

  The **PRE_INSTANTIATION_SETUP** and **POST_INSTANTIATION_SETUP** procedures must be used together to complete the Oracle Streams replication configuration. Typically, the **PRE_INSTANTIATION_SETUP** and **POST_INSTANTIATION_SETUP** procedures are used to perform database maintenance operations with little or no down time. See Oracle Streams Concepts and Administration for more information.
Automatic Platform Conversion

If the source and destination databases run on different platforms, then these procedures, or the scripts generated by these procedures, convert transferred datafiles to the appropriate platform automatically.

Actions Performed by These Procedures

To view all of the actions performed by one of these procedures in detail, use the procedure to generate a script, and view the script in a text editor.

Configuration Progress and Recoverability

When one of these procedures is run with the perform_actions parameter set to TRUE, metadata about its configuration actions is recorded in the following data dictionary views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then you can use the RECOVER_OPERATION procedure to complete the configuration after you correct the conditions that caused the error.

Note:

When one of these procedures is run with the perform_actions parameter set to FALSE, these views are not populated. Also, the views are not populated when a script generated by one of these procedures is run.

Requirements for Running These Procedures

Meet the following requirements when you use one of these procedures:

• Run the procedure at the capture database. The capture database is the database that will contain the capture process that captures changes made to the source database. If the capture database is the same as the source database, then a local capture process is configured. If the capture database is different from the source database, then a downstream capture process is configured. See Oracle Streams Replication Administrator's Guide for more information about the capture database.
• The user who runs the procedure must be able to use a database link from the source database to the destination database. This database link should have the same name as the global name of the destination database.

• If the procedure configures downstream capture, then the corresponding user at the capture database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.

• If the procedure configures downstream capture, and the capture database is different from the destination database, then the corresponding user at the capture database must be able to use a database link to access the destination database. This database link should have the same name as the global name of the destination database.

• Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the destination_database parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.

• Grant the user who runs the procedure the DBA role. This user must have the necessary privileges to complete the following actions:
  – Create ANYDATA queues, capture processes, propagations, and apply processes.
  – Specify supplemental logging
  – Run subprograms in the DBMS_STREAMS_ADM and DBMS_AQADM packages.
  – Access the database specified in the destination_database parameter through a database link. This database link should have the same name as the global name of the destination database.

Typically, the DBA role can be revoked from the user, if necessary, after the configuration is complete.

• The procedure, or the scripts generated by these procedure, must be run at an Oracle Database 10g Release 2 or later database.

• If the perform_actions parameter is set to TRUE in the procedure to configure the Oracle Streams environment directly, then all of the databases configured by the procedure must be Oracle Database 10g Release 2 or later databases.

• If the perform_actions parameter is set to FALSE in the procedure, and the environment is configured with a generated script, then the databases configured by the script must be Oracle Database 10g Release 1 or later databases. If the script configures an Oracle Database 10g Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10g Release 2 or later, such as queue-to-queue propagation.

• Each specified directory object must be created using the SQL statement CREATE DIRECTORY, and the user who invokes the procedure must have READ and WRITE privilege on each one.

• For procedures that include the bi_directional parameter, if the bi_directional parameter is set to TRUE, or if the source database is not the capture database, then the source_database parameter must specify a database that contains the database objects to be shared. The database specified in the destination_database parameter might or might not contain these database objects. If the destination database does not contain the shared database objects, then the procedure instantiates the database objects at the destination database (excluding the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures).
• For procedures that include the `bi_directional` parameter, if the `bi_directional` parameter is set to `TRUE` or if a network instantiation will be performed, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.

To ensure that the user who runs these procedures has the necessary privileges, you should configure an Oracle Streams administrator at each database, and each database link should be created in the Oracle Streams administrator's schema.

![See Also:

*Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator](image)

### Common Parameters for the Configuration Procedures

*Table 165-1* describes the common parameters for the procedures in this package that configure an Oracle Streams environment. Some of the procedures do not include all of the parameters in *Table 165-1*.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `perform_actions`  | If `TRUE`, then the procedure performs the necessary actions to configure the environment directly. If `FALSE`, then the procedure does not perform the necessary actions to configure the environment directly. Specify `FALSE` when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify `FALSE` and either of the following parameters is `NULL`:  
  • `script_name`  
  • `script_directory_object`  

| `script_name`      | If `non-NULL` and the `perform_actions` parameter is `FALSE`, then specify the name of the script generated by this procedure. The script contains all of the statements used to configure the environment. If a file with the specified script name exists in the specified directory for the `script_directory_object` parameter, then the procedure appends the statements to the existing file.  
If `non-NULL` and the `perform_actions` parameter is `TRUE`, then the procedure generates the specified script and performs the actions to configure the environment directly.  
If `NULL` and the `perform_actions` parameter is `TRUE`, then the procedure performs the actions to configure the environment directly and does not generate a script.  
If `NULL` and the `perform_actions` parameter is `FALSE`, then the procedure raises an error. |
### Table 165-1 (Cont.) Common Parameters for Configuration Procedures

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>script_directory_object</code></td>
<td>The directory object for the directory on the local computer system into which the generated script is placed. If the <code>script_name</code> parameter is <code>NULL</code>, then the procedure ignores this parameter and does not generate a script. If <code>NULL</code> and the <code>script_name</code> parameter is non-<code>NULL</code>, then the procedure raises an error. Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
<tr>
<td><code>capture_name</code></td>
<td>The name of each capture process configured to capture changes. Do not specify an owner. If the <code>bi_directional</code> parameter is set to <code>TRUE</code>, then each capture process created by this procedure has the specified name. If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database to the positive capture process rule set. If <code>NULL</code>, then the system generates a name for each capture process it creates. Note: The capture process name cannot be altered after the capture process is created.</td>
</tr>
<tr>
<td><code>capture_queue_table</code></td>
<td>The name of the queue table for each queue used by a capture process, specified as <code>[schema_name.]queue_table_name</code>. For example, <code>strmadmin.streams_queue_table</code>. If the schema is not specified, then the current user is the default. If <code>NULL</code>, then the system generates a name for the queue table of each queue used by a capture process, and the current user is the owner of each queue table.</td>
</tr>
<tr>
<td><code>capture_queue_name</code></td>
<td>The name of each queue used by a capture process, specified as <code>[schema_name.]queue_name</code>. For example, <code>strmadmin.streams_queue</code>. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If <code>NULL</code>, then the system generates a name for each queue used by a capture process.</td>
</tr>
<tr>
<td><code>capture_queue_user</code></td>
<td>The name of the user who requires <code>ENQUEUE</code> and <code>DEQUEUE</code> privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the <code>GRANT</code> option. If <code>NULL</code>, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the <code>DBMS_AQADM</code> package.</td>
</tr>
</tbody>
</table>
### Table 165-1  (Cont.) Common Parameters for Configuration Procedures

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of each propagation configured to propagate changes. Do not specify an owner. If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the positive propagation rule set. If NULL, then the system generates a name for each propagation it creates. <strong>Note:</strong> The propagation name cannot be altered after the propagation is created.</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of each apply process configured to apply changes. Do not specify an owner. If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the positive apply process rule set. The specified name must not match the name of an existing messaging client at the destination database. If NULL, then the system generates a name for each apply process it creates. When set to NULL, no apply process that applies changes from the source database can exist on the destination database. If an apply process that applies changes from the source database exists at the destination database, then specify a non-NULL value for this parameter. <strong>Note:</strong> The apply process name cannot be altered after the apply process is created.</td>
</tr>
<tr>
<td>apply_queue_table</td>
<td>The name of the queue table for each queue used by an apply process, specified as [schema_name.]queue_table_name. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the system generates a name for the queue table of each queue used by an apply process, and the current user is the owner of each queue table.</td>
</tr>
<tr>
<td>apply_queue_name</td>
<td>The name of each queue used by an apply process, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If NULL, then the system generates a name for each queue used by an apply process.</td>
</tr>
<tr>
<td>apply_queue_user</td>
<td>The name of the user who requires <strong>ENQUEUE</strong> and <strong>DEQUEUE</strong> privileges for the queue at the destination database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the <strong>GRANT</strong> option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the <strong>DBMS_AQADM</strong> package.</td>
</tr>
</tbody>
</table>
Table 165-1  (Cont.) Common Parameters for Configuration Procedures

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| bi_directional | Specify **TRUE** to configure bi-directional replication between the database specified in `source_database` and the database specified in `destination_database`. Both databases are configured as source and destination databases, a capture and apply process is configured to capture changes to both databases, and propagations are configured to propagate these changes. If **TRUE**, then a database link from the destination database to the source database with the same global name as the source database must exist.  
Specify **FALSE** to configure one way replication from the database specified in `source_database` and the database specified in `destination_database`. A capture process is configured at the current database and an apply process is configured at the destination database. A propagation is configured if necessary.  
**See Also**: Oracle Streams Replication Administrator's Guide for information about when propagations are configured.                                                                                             |
| include_ddl   | Specify **TRUE** to configure an Oracle Streams replication environment that maintains both DML and DDL changes. Specify **FALSE** to configure an Oracle Streams replication environment that maintains DML changes only. When this parameter is set to **FALSE**, DDL changes, such as `ALTER TABLE`, are not replicated.                                                                                               |

165.6 Summary of DBMS_STREAMS_ADM Subprograms

This table lists the **DBMS_STREAMS_ADM** subprograms and briefly describes them.

Table 165-2  DBMS_STREAMS_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADD_COLUMN Procedure</strong></td>
<td>Either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule</td>
</tr>
<tr>
<td><strong>ADD_GLOBAL_PROPAGATION_RULES Procedure</strong></td>
<td>Either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist</td>
</tr>
<tr>
<td><strong>ADD_GLOBAL_RULES Procedure</strong></td>
<td>Adds global rules to either the positive or negative rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist</td>
</tr>
<tr>
<td><strong>ADD_MESSAGE_PROPAGATION_RULE Procedure</strong></td>
<td>Either adds a message rule to the positive rule set for a propagation, or adds a message rule to the negative rule set for a propagation, and creates the specified propagation if it does not exist</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ADD.MESSAGE_RULE Procedure</td>
<td>Adds a message rule to either the positive or negative rule set of an apply process or messaging client, and creates the specified apply process or messaging client if it does not exist</td>
</tr>
<tr>
<td>ADD_SCHEMA_PROPAGATION_RULES Procedure</td>
<td>Either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist</td>
</tr>
<tr>
<td>ADD_SCHEMA_RULES Procedure</td>
<td>Adds schema rules to either the positive or negative rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist</td>
</tr>
<tr>
<td>ADD_SUBSET_PROPAGATION_RULES Procedure</td>
<td>Adds subset rules to the positive rule set for a propagation, and creates the specified propagation if it does not exist</td>
</tr>
<tr>
<td>ADD_SUBSET_RULES Procedure</td>
<td>Adds subset rules to the positive rule set of a capture process, synchronous capture, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist</td>
</tr>
<tr>
<td>ADD_TABLE_PROPAGATION_RULES Procedure</td>
<td>Either adds table rules to the positive rule set for a propagation, or adds table rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist</td>
</tr>
<tr>
<td>CLEANUP_INSTANTIATION_SETUP Procedure</td>
<td>Removes an Oracle Streams replication configuration that was set up by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures in this package</td>
</tr>
<tr>
<td>DELETE_COLUMN Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which deletes a column from a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td>GET_MESSAGE_TRACKING Function</td>
<td>Returns the tracking label for the current session</td>
</tr>
<tr>
<td>GET_SCN_MAPPING Procedure</td>
<td>Gets information about the system change number (SCN) values to use for Oracle Streams capture and apply processes in an Oracle Streams replication environment</td>
</tr>
<tr>
<td>KEEP_COLUMNS Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which keeps a list of columns in a row LCR that satisfies the specified rule</td>
</tr>
</tbody>
</table>
### Table 165-2 (Cont.) DBMS_STREAMS_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAINTAIN_CHANGE_TABLE Procedure</td>
<td>Configures an Oracle Streams environment that records in a change table the data manipulation language (DML) changes made to a source table. Optionally, this procedure can also configure one-way replication of the table from the source database to the destination database.</td>
</tr>
<tr>
<td>MAINTAIN_GLOBAL Procedure</td>
<td>Configures an Oracle Streams environment that replicates changes at the database level between two databases.</td>
</tr>
<tr>
<td>MAINTAIN_SCHEMAS Procedure</td>
<td>Configures an Oracle Streams environment that replicates changes to specified schemas between two databases.</td>
</tr>
<tr>
<td>MAINTAIN_SIMPLE_TABLESPACE Procedure</td>
<td>Clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases. This procedure is deprecated.</td>
</tr>
<tr>
<td>MAINTAIN_SIMPLE_TTS Procedure</td>
<td>Clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases.</td>
</tr>
<tr>
<td>MAINTAIN_TABLES Procedure</td>
<td>Configures an Oracle Streams environment that replicates changes to specified tables between two databases.</td>
</tr>
<tr>
<td>MAINTAIN_TABLESPACES Procedure</td>
<td>Clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases. This procedure is deprecated.</td>
</tr>
<tr>
<td>MAINTAIN_TTS Procedure</td>
<td>Clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases.</td>
</tr>
<tr>
<td>MERGEStreams Procedure</td>
<td>Merges a stream flowing from one capture process with a stream flowing from another capture process.</td>
</tr>
<tr>
<td>MERGE_STREAMS_JOB Procedure</td>
<td>Determines whether the original capture process and the cloned capture are within the specified merge threshold and, if they are, runs the MERGE_STREAMS procedure to merge the two streams.</td>
</tr>
<tr>
<td>POST_INSTANTIATION_SETUP Procedure</td>
<td>Performs the actions required after instantiation to configure an Oracle Streams replication environment.</td>
</tr>
<tr>
<td>PRE_INSTANTIATION_SETUP Procedure</td>
<td>Performs the actions required before instantiation to configure an Oracle Streams replication environment.</td>
</tr>
<tr>
<td>PURGE_SOURCE_CATALOG Procedure</td>
<td>Removes all Oracle Streams data dictionary information at the local database for the specified object.</td>
</tr>
<tr>
<td>RECOVER_OPERATION Procedure</td>
<td>Provides options for an Oracle Streams replication configuration operation that stopped because it encountered an error. This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation.</td>
</tr>
<tr>
<td>REMOVE_QUEUE Procedure</td>
<td>Removes the specified ANYDATA queue.</td>
</tr>
</tbody>
</table>
Table 165-2  (Cont.) DBMS_STREAMS_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOVE_RULE Procedure</td>
<td>Removes the specified rule or all rules from the rule set associated with the specified capture process, synchronous capture, propagation, apply process, or messaging client.</td>
</tr>
<tr>
<td>REMOVE_STREAMS_CONFIGURATION Procedure</td>
<td>Removes the Oracle Streams configuration at the local database</td>
</tr>
<tr>
<td>RENAME_COLUMN Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which renames a column in a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td>RENAME_SCHEMA Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which renames a schema in a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td>RENAME_TABLE Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which renames a table in a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td>SET_MESSAGE_NOTIFICATION Procedure</td>
<td>Sets a notification for messages that can be dequeued by a specified Oracle Streams messaging client from a specified queue</td>
</tr>
<tr>
<td>SET_MESSAGE_TRACKING Procedure</td>
<td>Sets the tracking label for logical change records (LCRs) produced by the current session</td>
</tr>
<tr>
<td>SET_RULE_TRANSFORM_FUNCTION Procedure</td>
<td>Sets or removes the transformation function name for a rule-based transformation</td>
</tr>
<tr>
<td>SET_TAG Procedure</td>
<td>Sets the binary tag for all redo entries subsequently generated by the current session</td>
</tr>
<tr>
<td>SET_UP_QUEUE Procedure</td>
<td>Creates a queue table and a queue for use with the capture, propagate, and apply functionality of Oracle Streams</td>
</tr>
<tr>
<td>SPLIT_STREAMS Procedure</td>
<td>Splits one stream flowing from a capture process off from all of the other streams flowing from the capture process</td>
</tr>
</tbody>
</table>

Note:

All subprograms commit unless specified otherwise.

165.6.1 ADD_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to TRUE, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.
This procedure is overloaded. The `column_value` and `column_function` parameters are mutually exclusive.

**Note:**
- **ADD_COLUMN** transformations cannot add columns of the following data types: BLOB, CLOB, NCLOB, BFILE, LONG, LONG RAW, ROWID, user-defined types (including object types, REFS, varrays, nested tables), and Oracle-supplied types (including any types, XML types, spatial types, and media types).
- Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

**See Also:**
*Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

**Syntax**

```sql
DBMS_STREAMS_ADM.ADD_COLUMN(
    rule_name     IN  VARCHAR2,
    table_name    IN  VARCHAR2,
    column_name   IN  VARCHAR2,
    column_value  IN  ANYDATA,
    value_type    IN  VARCHAR2     DEFAULT 'NEW',
    step_number   IN  NUMBER       DEFAULT 0,
    operation     IN  VARCHAR2     DEFAULT 'ADD');

DBMS_STREAMS_ADM.ADD_COLUMN(
    rule_name     IN  VARCHAR2,
    table_name    IN  VARCHAR2,
    column_name   IN  VARCHAR2,
    column_function IN  VARCHAR2,
    value_type    IN  VARCHAR2     DEFAULT 'NEW',
    step_number   IN  NUMBER       DEFAULT 0,
    operation     IN  VARCHAR2     DEFAULT 'ADD');
```
### Parameters

Table 165-3 ADD_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table to which the column is added in the row LCR, specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column added to each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>column_value</td>
<td>The value of the added column. Specify the appropriate ANYDATA function for the column datatype and the column value. For example, if the datatype of the column being added is NUMBER and the value is NULL, then specify the ANYDATA(ConvertNumber(NULL)) function. This parameter cannot be specified if the column_function parameter is specified.</td>
</tr>
<tr>
<td>column_function</td>
<td>Either the 'SYSDATE' or the 'SYSTIMESTAMP' SQL function. The 'SYSDATE' SQL function places the current date and time set for the operating system on which the database resides. The datatype of the returned value is DATE, and the format returned depends on the value of the NLS_DATE_FORMAT initialization parameter. The 'SYSTIMESTAMP' SQL function returns the system date, including fractional seconds and time zone, of the system on which the database resides. The return type is TIMESTAMP WITH TIME ZONE. The function executes when the rule evaluates to TRUE. This parameter cannot be specified if the column_value parameter is specified.</td>
</tr>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to add the column to the new values in the row LCR. Specify 'OLD' to add the column to the old values in the row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

### Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the add column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the ADD_COLUMN procedures when one or more of these parameters is NULL:
<table>
<thead>
<tr>
<th>table_name</th>
<th>column_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all add column transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all add column transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all add column transformations with the specified column_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all add column transformations with the specified table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all add column transformations with the specified column_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all add column transformations with the specified table_name and column_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all add column transformations with the specified table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all add column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>

### 165.6.2 ADD_GLOBAL_PROPAGATION_RULES Procedure

This procedure either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

**Syntax**

```sql
DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

```sql
DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            IN   VARCHAR2  DEFAULT NULL,
```
Parameters

Table 165-4  ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner.</td>
</tr>
<tr>
<td></td>
<td>If the specified propagation does not exist, then the procedure creates it automatically.</td>
</tr>
<tr>
<td></td>
<td>If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue, and the queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link, specified as [schema_name.]queue_name[@dblink_name], if the destination queue is in a remote database. The queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter <a href="mailto:strmadmin.streams_queue@dbs2.net">strmadmin.streams_queue@dbs2.net</a> for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
</tbody>
</table>
Table 165-4  (Cont.) ADD_GLOBAL_PROPAGATION RULES Procedure
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-Anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the changes originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
</tbody>
</table>
Table 165-4 (Cont.) ADD_GLOBAL_PROPAGATION_RULES Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| inclusion_rule  | If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation.  
If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation.  
In either case, the system creates the rule set if it does not exist. |
| and_condition   | If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:  
(system_condition) AND (and_condition) |

The variable in the specified condition must be :lcr. For example, to specify that the global rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:

:lc.r.get_tag() = HEXTORAW('02')

The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure the procedure only generates the appropriate rule.

Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.

See Also: Logical Change Record TYPES
Table 165-4  (Cont.) ADD_GLOBAL_PROPAGATION_RULES Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| queue_to_queue | If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database. If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment. The procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  
  • If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
  • If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
  • If NULL, then the procedure does not change the queue to queue property of the propagation. |

See Also: Oracle Streams Concepts and Administration for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on include_dml and include ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A propagation uses the rules for filtering.

See Also:

• "* Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
• "DBMS_STREAMS_ADM Security Model"

Examples

The following is an example of a global rule condition created for DML changes:

```sql
(:dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET')
```
ADD_GLOBAL_RULES Procedure

This procedure adds rules to an Oracle Streams client rule set.

It adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `capture`, this procedure adds capture process rules for capturing changes to an entire database.
- When the `streams_type` parameter is set to `apply` and the `streams_name` parameter specifies the name of an apply process, this procedure adds apply process rules for applying all logical change records (LCRs) in a queue. The rules can specify that the LCRs must be from a particular source database.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds messaging client rules for dequeuing all persistent LCRs from a queue. The rules can specify that the LCRs must be from a particular source database.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

**Note:**
If you add global rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects that are not support by Oracle Streams. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

**Note:**
Currently, messaging clients cannot dequeue buffered messages.

**Syntax**

```sql
DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type        IN   VARCHAR2,
    streams_name        IN   VARCHAR2  DEFAULT NULL,
    queue_name          IN   VARCHAR2  DEFAULT 'streams_queue',
    include_dml         IN   BOOLEAN   DEFAULT TRUE,
    include_ddl         IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
    source_database     IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name       OUT  VARCHAR2,
    ddl_rule_name       OUT  VARCHAR2,
    inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
    and_condition       IN   VARCHAR2  DEFAULT NULL);
```
Parameters

Table 165-5  ADD_GLOBAL_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| streams_type          | The type of Oracle Streams client:  
  · Specify capture for a capture process.  
  · Specify apply for an apply process.  
  · Specify dequeue for a messaging client.  |
| streams_name          | The name of the capture process, apply process, or messaging client. Do not specify an owner.  
  If NULL, if streams_type is capture or dequeue, and if one relevant capture process or messaging client for the queue exists, then the relevant Oracle Streams client is used. If no relevant Oracle Streams client exists for the queue, then an Oracle Streams client is created automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified streams_type for the queue exist, then the procedure raises an error.  
  If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:  
  · If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.  
  · If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process.  
  If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name.  
  If NULL and multiple relevant apply processes exist, then the procedure raises an error.  
  Each apply process and messaging client must have a unique name.  |
| queue_name            | The name of the local queue, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANYDATA type.  
  For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.  
  For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.  |
Table 165-5 (Cont.) ADD_GLOBAL_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle Streams Replication Administrator's Guide for more information about tags.</td>
</tr>
</tbody>
</table>
Table 165-5  (Cont.) ADD_GLOBAL_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules.</td>
</tr>
<tr>
<td></td>
<td>For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.</td>
</tr>
<tr>
<td></td>
<td>For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database.</td>
</tr>
<tr>
<td></td>
<td>For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information.</td>
</tr>
<tr>
<td></td>
<td>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the Oracle Streams client. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the Oracle Streams client. In either case, the system creates the rule set if it does not exist.</td>
</tr>
</tbody>
</table>
Table 165-5  (Cont.) ADD_GLOBAL_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| and_condition | If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:  

(system_condition) AND (and_condition)  

The variable in the specified condition must be :lcr. For example, to specify that the global rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:  

:lcr.get_tag() = HEXTORAW('02')  

The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.  

See Also: Logical Change Record TYPES

Usage Notes

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A capture process, apply process, or messaging client uses the rules for filtering.

See Also:

- "Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
- "Security Model"

Examples

The following is an example of a global rule condition created for DML changes:

(:dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET')
165.6.4 ADD_MESSAGE_PROPAGATION_RULE Procedure

This procedure adds a message rule to the positive rule set for a propagation, or adds a message rule to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains the OUT parameter rule_name, and the other does not.

Syntax

```sql
DBMS_STREAMS_ADM.ADD_MESSAGE_PROPAGATION_RULE(
    message_type            IN   VARCHAR2,
    rule_condition          IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    rule_name               OUT  VARCHAR2,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_type</td>
<td>The type of the message. The type can be an Oracle built-in type, such as VARCHAR2 or NUMBER, or it can be a user-defined type. If the type is not an Oracle built-in type, then it is specified as [schema_name.]type_name. If the schema is not specified, then the current user is the default. For example, to specify VARCHAR2, enter VARCHAR2(n), where n is the size specification. To specify a type named usr_msg in the strmadmin schema, enter strmadmin.usr_msg for this parameter. The following data types require a size specification: VARCHAR2, NVARCHAR2, and RAW. See Also: Oracle Database SQL Language Reference for more information about data types</td>
</tr>
<tr>
<td>rule_condition</td>
<td>The rule condition for this message type. The rule variable name specified in the rule condition must be the following: :msg See Also: &quot;Examples&quot;</td>
</tr>
</tbody>
</table>
### Table 165-6  (Cont.) ADD_MESSAGE_PROPAGATION_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner.</td>
</tr>
<tr>
<td></td>
<td>If the specified propagation does not exist, then the procedure creates it</td>
</tr>
<tr>
<td></td>
<td>automatically.</td>
</tr>
<tr>
<td></td>
<td>If NULL and a propagation exists for the same source queue and</td>
</tr>
<tr>
<td></td>
<td>destination queue (including database link), then the procedure uses this</td>
</tr>
<tr>
<td></td>
<td>propagation.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no propagation exists for the same source queue and</td>
</tr>
<tr>
<td></td>
<td>destination queue (including database link), then the procedure</td>
</tr>
<tr>
<td></td>
<td>creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The</td>
</tr>
<tr>
<td></td>
<td>current database must contain the source queue, and the queue must be</td>
</tr>
<tr>
<td></td>
<td>ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a source queue named streams_queue in the strmadmin</td>
</tr>
<tr>
<td></td>
<td>schema, enter strmadmin.streams_queue for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link, specified as</td>
</tr>
<tr>
<td></td>
<td>[schema_name.]queue_name[dbname], if the destination queue is in a remote</td>
</tr>
<tr>
<td></td>
<td>database. The queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a destination queue named streams_queue in the</td>
</tr>
<tr>
<td></td>
<td>strmadmin schema and use a database link named dbs2.net, enter strmadmin</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:streams_queue@dbs2.net">streams_queue@dbs2.net</a> for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>If the database link is omitted, then the procedure uses the global name</td>
</tr>
<tr>
<td></td>
<td>of the current database, and the source queue and destination queue must</td>
</tr>
<tr>
<td></td>
<td>be in the same database.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rule to the</td>
</tr>
<tr>
<td></td>
<td>positive rule set for the propagation.</td>
</tr>
<tr>
<td></td>
<td>If inclusion_rule is FALSE, then the procedure adds the rule to the</td>
</tr>
<tr>
<td></td>
<td>negative rule set for the propagation.</td>
</tr>
<tr>
<td>rule_name</td>
<td>Contains the rule name</td>
</tr>
</tbody>
</table>
### Table 165-6  (Cont.) ADD_MESSAGE_PROPAGATION_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| queue_to_queue       | If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.  
  If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.  
  This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  
  • If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
  • If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
  • If NULL, then the procedure does not change the queue to queue property of the propagation.  
  **See Also:** Oracle Streams Concepts and Administration for more information about queue-to-queue propagations |

### Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

When you use this procedure to create a rule set for a message rule, the new rule set does not have an evaluation context. If no evaluation context exists for the specified message type, then this procedure creates a new evaluation context and associates it with the new rule. The evaluation context also has a system-generated name. If you create new rules that use an existing message type, then the new rules use the existing evaluation context for the message type.

**See Also:**

- "* Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
- "DBMS_STREAMS_ADM Security Model"

### Examples

Suppose the message type is VARCHAR2(128). Given this type, the following rule condition can be specified:

```
':msg = ''HQ''
```
This rule condition evaluates to **true** if a user message of type `VARCHAR2(128)` has `HQ` for its value.

Suppose the message type is `usr_msg`, and that this type has the following attributes: `source_dbname`, `owner`, `name`, and `message`. Given this type, the following rule condition can be specified:

```
':msg.source_dbname = ''DBS1.NET'' AND ' || ':msg.owner = ''HR'' AND ' ||
':msg.name = ''EMPLOYEES''
```

This rule condition evaluates to **true** if a user message of type `usr_msg` has `DBS1.NET` for its `source_dbname` attribute, `HR` for its `owner` attribute, and `EMPLOYEES` for its `name` attribute.

**Note:**
The quotation marks in the preceding examples are all single quotation marks.

### 165.6.5 ADD_MESSAGE_RULE Procedure

This procedure adds a message rule to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `apply`, this procedure adds an apply process rule for dequeuing user messages of a specific message type from a queue.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds a messaging client rule for dequeuing user messages of a specific message type from a queue.

This procedure also creates the specified Oracle Streams client if it does not exist.

This procedure is overloaded. One version of this procedure contains the `OUT` parameter `rule_name`, and the other does not.

**Note:**
Currently, messaging clients cannot dequeue buffered messages.

**Syntax**

```sql
DBMS_STREAMS_ADM.ADD_MESSAGE_RULE(
  message_type    IN   VARCHAR2,
  rule_condition  IN   VARCHAR2,
  streams_type    IN   VARCHAR2,
  streams_name    IN   VARCHAR2  DEFAULT NULL,
  queue_name      IN   VARCHAR2  DEFAULT 'streams_queue',
  inclusion_rule  IN   BOOLEAN   DEFAULT TRUE,
  rule_name       OUT  VARCHAR2);
```
message_type  IN  VARCHAR2,
rule_condition IN  VARCHAR2,
streams_type  IN  VARCHAR2,
streams_name  IN  VARCHAR2  DEFAULT NULL,
queue_name    IN  VARCHAR2  DEFAULT 'streams_queue',
inclusion_rule IN  BOOLEAN   DEFAULT TRUE);

Parameters

Table 165-7  ADD_MESSAGE_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_type</td>
<td>The type of the message. The type can be an Oracle built-in type, such as VARCHAR2 or NUMBER, or it can be a user-defined type. If the type is not an Oracle built-in type, then it is specified as [schema_name.]type_name. If the schema is not specified, then the current user is the default. For example, to specify VARCHAR2, enter VARCHAR2(n), where n is the size specification. To specify a type named usr_msg in the strmadmin schema, enter strmadmin.usr_msg for this parameter. The following data types require a size specification: VARCHAR2, NVARCHAR2, and RAW. See Also: <em>Oracle Database SQL Language Reference</em> for more information about data types.</td>
</tr>
<tr>
<td>rule_condition</td>
<td>The rule condition for this message type. The rule variable name specified in the rule condition must be the following: :msg. See Also: &quot;Examples&quot;</td>
</tr>
<tr>
<td>streams_type</td>
<td>The type of message consumer, either apply for apply process or dequeue for messaging client</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the Oracle Streams apply process or messaging client. If the specified streams_type is apply, then specify the name of the apply process. Do not specify an owner. If the specified apply process does not exist, then the procedure creates it automatically with a system-generated name. If the specified streams_type is dequeue, then specify the messaging client. For example, if the user strmadmin is the messaging client, then specify strmadmin. If NULL and a relevant apply process or messaging client for the queue exists, then the procedure uses the relevant apply process or messaging client. If NULL and multiple relevant apply processes or messaging clients for the queue exist, then the procedure raises an error. If NULL and no Oracle Streams client of the specified streams_type exists for the queue, then the procedure creates an apply process or messaging client automatically with a system-generated name. An apply process and a messaging client cannot have the same name.</td>
</tr>
</tbody>
</table>

See Also: "Examples"
Table 165-7  (Cont.) ADD_MESSAGE_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the local queue from which messages will be dequeued, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rule to the positive rule set for the apply process or messaging client. If inclusion_rule is FALSE, then the procedure adds the rule to the negative rule set for the apply process or messaging client. In either case, the system creates the rule set if it does not exist.</td>
</tr>
<tr>
<td>rule_name</td>
<td>Contains the rule name</td>
</tr>
</tbody>
</table>

Usage Notes

If an apply process rule is added, then this procedure creates the apply process if it does not exist. An apply process created by this procedure can apply only user messages, and dequeued messages are sent to the message handler for the apply process. If a messaging client rule is added, then this procedure creates a messaging client if it does not exist.

When you use this procedure to create a rule set for a message rule, the new rule set does not have an evaluation context. If no evaluation context exists for the specified message type, then this procedure creates a new evaluation context and associates it with the new rule. The evaluation context also has a system-generated name. If you create new rules that use an existing message type, then the new rules use the existing evaluation context for the message type.

See Also:

- "Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
- "Security Model"
- ALTER_APPLY Procedure for more information about setting a message handler for an apply process

Examples

Suppose the message type is VARCHAR2(128). Given this type, the following rule condition can be specified:

'':msg = ''HQ'''

This rule condition evaluates to TRUE if a user message of type VARCHAR2(128) has HQ for its value.
Suppose the message type is `usr_msg`, and that this type has the following attributes:

- `source_dbname`
- `owner`
- `name`
- `message`

Given this type, the following rule condition can be specified:

```
':msg.source_dbname = ''DBS1.NET'' AND '|| ':msg.owner = ''HR'' AND '||
':msg.name = ''EMPLOYEES''
```

This rule condition evaluates to `TRUE` if a user message of type `usr_msg` has `DBS1.NET` for its `source_dbname` attribute, `HR` for its `owner` attribute, and `EMPLOYEES` for its `name` attribute.

**Note:**
The quotation marks in the preceding examples are all single quotation marks.

### 165.6.6 ADD_SCHEMA_PROPAGATION_RULES Procedure

This procedure either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

**Syntax**

```sql
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

```sql
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```
Parameters

Table 165-8  ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema. For example, hr.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link, specified as [schema_name.]queue_name[@dblink_name], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter <a href="mailto:strmadmin.streams_queue@dbs2.net">strmadmin.streams_queue@dbs2.net</a> for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. Note: Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
</tbody>
</table>
Table 165-8  (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
</tbody>
</table>
### Table 165-8  (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation.</td>
</tr>
<tr>
<td></td>
<td>If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation.</td>
</tr>
<tr>
<td></td>
<td>In either case, the system creates the rule set if it does not exist.</td>
</tr>
<tr>
<td>and_condition</td>
<td>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</td>
</tr>
<tr>
<td></td>
<td>(system_condition) AND (and_condition)</td>
</tr>
<tr>
<td></td>
<td>The variable in the specified condition must be :lcr. For example, to specify that the schema rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</td>
</tr>
<tr>
<td></td>
<td>:lcr.get_tag() = HEXTORAW('02')</td>
</tr>
<tr>
<td></td>
<td>The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</td>
</tr>
<tr>
<td></td>
<td>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> Logical Change Record TYPES</td>
</tr>
</tbody>
</table>
Table 165-8 (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| queue_to_queue           | If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database. If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:   
   - If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.   
   - If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.   
   - If NULL, then the procedure does not change the queue to queue property of the propagation. |

See Also: Oracle Streams Concepts and Administration for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on include_dml and include ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A propagation uses the rules for filtering.

See Also:

- "* Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
- "DBMS_STREAMS_ADM Security Model"

Examples

The following is an example of a schema rule condition created for DML changes:

```
((:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET')
```
165.6.7 ADD_SCHEMA_RULES Procedure

This procedure adds rules to Oracle Streams clients rule sets.

It adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the *streams_type* parameter is set to *capture*, this procedure adds capture process rules for capturing changes to a specified schema.
- When the *streams_type* parameter is set to *apply* and the *streams_name* parameter specifies the name of an apply process, this procedure adds apply process rules for applying logical change records (LCRs) in a queue that contain changes to a specified schema. The rules can specify that the LCRs must be from a particular source database.
- When the *streams_type* parameter is set to *dequeue*, this procedure adds messaging client rules for dequeuing persistent LCRs from a queue that contain changes to a specified schema. The rules can specify that the LCRs must be from a particular source database.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

---

**Note:**

If you add schema rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects in the schema that are not support by Oracle Streams. Query the DBA_STREAMS_UNSUPPORTED data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

---

**Note:**

Currently, messaging clients cannot dequeue buffered messages.

Syntax

```
DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name         IN   VARCHAR2,
    streams_type        IN   VARCHAR2,
    streams_name        IN   VARCHAR2  DEFAULT NULL,
    queue_name          IN   VARCHAR2  DEFAULT 'streams_queue',
    include_dml         IN   BOOLEAN   DEFAULT TRUE,
    include_ddl         IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
    source_database     IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name       OUT  VARCHAR2,
    ddl_rule_name       OUT  VARCHAR2,
)
```
Parameters

**Table 165-9 ADD_SCHEMA_RULES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema. For example, hr. You can specify a schema that does not yet exist, because Oracle Streams does not validate the existence of the schema.</td>
</tr>
</tbody>
</table>
| streams_type      | The type of Oracle Streams client:  
• Specify capture for a capture process.  
• Specify apply for an apply process.  
• Specify dequeue for a messaging client.                                                                                                           |
| streams_name      | The name of the capture process, apply process, or messaging client. Do not specify an owner.                                                                                                                 |
| source_database   | If NULL, if streams_type is capture or dequeue, and if one relevant capture process or messaging client for the queue exists, then the relevant Oracle Streams client is used. If no relevant Oracle Streams client exists for the queue, then an Oracle Streams client is created automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified streams_type for the queue exist, then the procedure raises an error. If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:  
• If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.  
• If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process.  
If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error. Each apply process and messaging client must have a unique name. |
Table 165-9  (Cont.) ADD_SCHEMA_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the local queue, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
</tbody>
</table>

See Also: Oracle Streams Replication Administrator's Guide for more information about tags.
Table 165-9  (Cont.) ADD_SCHEMA_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database. For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the Oracle Streams client. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the Oracle Streams client. In either case, the system creates the rule set if it does not exist.</td>
</tr>
</tbody>
</table>
Table 165-9  (Cont.) ADD_SCHEMA_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and_condition</td>
<td>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</td>
</tr>
<tr>
<td></td>
<td>(system_condition) AND (and_condition)</td>
</tr>
<tr>
<td></td>
<td>The variable in the specified condition must be :lcr. For example, to specify that the schema rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</td>
</tr>
<tr>
<td></td>
<td>:lcr.get_tag() = HEXTORAW('02')</td>
</tr>
<tr>
<td></td>
<td>The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A capture process, apply process, or messaging client uses the rules for filtering.

**See Also:** Logical Change Record TYPES

**Examples**

The following is an example of a schema rule condition created for DML changes:


```
({:dml.get_object_owner() = 'HR'} and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET')
```

---

**Summary of DBMS_STREAMS_ADM Subprograms**

Chapter 165

165-52
165.6.8 ADD_SUBSET_PROPAGATION_RULES Procedure

This procedure adds propagation rules that propagate the logical change records (LCRs) related to a subset of the rows in the specified table in a source queue to a destination queue, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains three OUT parameters, and the other does not.

Syntax

```sql
DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES(
  table_name               IN   VARCHAR2,
  dml_condition            IN   VARCHAR2,
  streams_name             IN   VARCHAR2  DEFAULT NULL,
  source_queue_name        IN   VARCHAR2,
  destination_queue_name   IN   VARCHAR2,
  include_tagged_lcr       IN   BOOLEAN   DEFAULT FALSE,
  source_database          IN   VARCHAR2  DEFAULT NULL,
  insert_rule_name         OUT  VARCHAR2,
  update_rule_name         OUT  VARCHAR2,
  delete_rule_name         OUT  VARCHAR2,
  queue_to_queue           IN   BOOLEAN   DEFAULT NULL);
```

```sql
DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES(
  table_name               IN   VARCHAR2,
  dml_condition            IN   VARCHAR2,
  streams_name             IN   VARCHAR2  DEFAULT NULL,
  source_queue_name        IN   VARCHAR2,
  destination_queue_name   IN   VARCHAR2,
  include_tagged_lcr       IN   BOOLEAN   DEFAULT FALSE,
  source_database          IN   VARCHAR2  DEFAULT NULL,
  queue_to_queue           IN   BOOLEAN   DEFAULT NULL);
```

Parameters

Table 165-10  ADD_SUBSET_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>The specified table must exist in the same database as the propagation. Also, the specified table cannot have any LOB, LONG RAW, or XMLType columns currently or in the future.</td>
</tr>
<tr>
<td>dml_condition</td>
<td>The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL. For example, to specify rows in the hr.employees table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition: 'salary &gt; 4000 and job_id = '&quot;&quot;SA_MAN&quot;&quot;' '</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The quotation marks in the preceding example are all single quotation marks.</td>
</tr>
</tbody>
</table>
Table 165-10  (Cont.) ADD_SUBSET_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link, specified as [schema_name.]queue_name[@dblink_name], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter <a href="mailto:strmadmin.streams_queue@dbs2.net">strmadmin.streams_queue@dbs2.net</a> for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. <strong>Note:</strong> Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then an LCR is considered for propagation only when the LCR contains a NULL tag. A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database. <strong>See Also:</strong> <em>Oracle Streams Replication Administrator’s Guide</em> for more information about tags</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.</td>
</tr>
</tbody>
</table>
### Table 165-10  (Cont.) ADD_SUBSET_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert_rule_name</td>
<td>Contains the system-generated INSERT rule name. This rule handles inserts and updates that must be converted into inserts.</td>
</tr>
<tr>
<td>update_rule_name</td>
<td>Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.</td>
</tr>
<tr>
<td>delete_rule_name</td>
<td>Contains the system-generated DELETE rule name. This rule handles deletes and updates that must be converted into deletes.</td>
</tr>
</tbody>
</table>
| queue_to_queue     | If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database. If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  
  - If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
  - If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
  - If NULL, then the procedure does not change the queue to queue property of the propagation. See Also: Oracle Streams Concepts and Administration for more information about queue-to-queue propagations |

### Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

Running this procedure generates three rules for the specified propagation: one for INSERT statements, one for UPDATE statements, and one for DELETE statements. For INSERT and DELETE statements, only row LCRs that satisfy the condition specified for the dml_condition parameter are propagated. For UPDATE statements, the following variations are possible:

- If both the new and old values in a row LCR satisfy the specified dml_condition, then the row LCR is propagated without any changes.
- If neither the new or old values in a row LCR satisfy the specified dml_condition, then the row LCR is not propagated.
- If the old values for a row LCR satisfy the specified dml_condition, but the new values do not, then the update row LCR is converted into a delete row LCR.
- If the new values for a row LCR satisfy the specified dml_condition, but the old values do not, then the update row LCR is converted to an insert row LCR.

When an update is converted into an insert or a delete, it is called row migration.
A propagation uses the rules for filtering. If the propagation does not have a positive rule set, then the procedure creates a positive rule set automatically, and the rules for propagating changes to the table are added to the positive rule set. A subset rule can be added to positive rule set only, not to a negative rule set. Other rules in an existing positive rule set for the propagation are not affected. Additional rules can be added using either the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The ADD_SUBSET_RULES procedure is overloaded, and the system-generated rule names for INSERT, UPDATE, and DELETE statements are returned.

When you create propagation subset rules for a table, you should create an unconditional supplemental log group at the source database with all the columns in the table. Supplemental logging is required if an update must be converted to an insert. The propagation rule must have all the column values to be able to perform this conversion correctly.

**Note:**

Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so might have unpredictable results because row migration would not be performed on LCRs that are not discarded by the negative rule set.

**See Also:**

- “* Procedures That Create Rules for Oracle Streams Clients and XStream Clients”
- “DBMS_STREAMS_ADM Security Model”

**Examples**

The following is an example of a rule condition created for filtering a row LCR containing an update operation when the dml_condition is region_id = 2, the table_name is hr.regions, and the source_database is dbs1.net:

```sql
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```
165.6.9 ADD_SUBSET_RULES Procedure

This procedure adds rules to an Oracle Streams client.

It adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `capture`, this procedure adds capture process rules for capturing changes to a subset of rows in a specified table.
- When the `streams_type` parameter is set to `sync_capture`, this procedure adds rules for capturing changes to a subset of rows in a specified table.
- When the `streams_type` parameter is set to `apply` and the `streams_name` parameter specifies the name of an apply process, this procedure adds apply process rules for applying logical change records (LCRs) in a queue that contain changes to a subset of rows in a specified table. The rules can specify that the LCRs must be from a particular source database.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds messaging client rules for dequeuing persistent LCRs from a queue that contain changes to a subset of rows in a specified table. The rules can specify that the LCRs must be from a particular source database.

This procedure creates the specified capture process, synchronous capture, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains three `OUT` parameters, and the other does not.

```
Note:

- Currently, messaging clients cannot dequeue buffered messages.
- The invoking user must be granted the DBA role to create a synchronous capture.
```

Syntax

```
DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
  table_name          IN   VARCHAR2,
  dml_condition       IN   VARCHAR2,
  streams_type        IN   VARCHAR2 DEFAULT 'apply',
  streams_name        IN   VARCHAR2 DEFAULT NULL,
  queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
  include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
  source_database     IN   VARCHAR2 DEFAULT NULL,
  insert_rule_name    OUT  VARCHAR2,
  update_rule_name    OUT  VARCHAR2,
  delete_rule_name    OUT  VARCHAR2);
```

```
DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
  table_name          IN   VARCHAR2,
  dml_condition       IN   VARCHAR2,
  streams_type        IN   VARCHAR2 DEFAULT 'apply',
  streams_name        IN   VARCHAR2 DEFAULT NULL,
  queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
```

Parameters

Table 165-11  ADD_SUBSET_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name.object_name. For example, hr.employees. If the schema is not specified, then the current user is the default. The specified table must exist in the same database as the capture process, synchronous capture, apply process, or messaging client. Also, the specified table cannot have any LOB, LONG, LONG RAW, or XMLType columns currently or in the future.</td>
</tr>
<tr>
<td>dml_condition</td>
<td>The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL. For example, to specify rows in the hr.employees table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition: ' salary &gt; 4000 and job_id = ''SA_MAN'' '</td>
</tr>
<tr>
<td>streams_type</td>
<td>The type of Oracle Streams client: • Specify capture for a capture process. • Specify sync_capture for a synchronous capture. • Specify apply for an apply process. • Specify dequeue for a messaging client.</td>
</tr>
</tbody>
</table>

Note: The quotation marks in the preceding example are all single quotation marks.
Table 165-11  (Cont.) ADD_SUBSET_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| streams_name  | The name of the capture process, synchronous capture, apply process, or messaging client. Do not specify an owner.  
If NULL, if streams_type is capture, sync_capture, or dequeue, and if one relevant capture process, synchronous capture, or messaging client for the queue exists, then the procedure uses the relevant Oracle Streams client. If no relevant Oracle Streams client exists for the queue, then the procedure creates an Oracle Streams client automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified streams_type for the queue exist, then the procedure raises an error.  
If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:  
• If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.  
• If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process.  
If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name.  
If NULL and multiple relevant apply processes exist, then the procedure raises an error.  
Each apply process and messaging client must have a unique name. |
| queue_name    | The name of the local queue, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANYDATA type.  
For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.  
For capture process or synchronous capture rules, this is the queue into which a capture process or synchronous capture enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages. |
Table 165-11  (Cont.) ADD_SUBSET_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the Oracle Streams client performs its action regardless of the tag:</td>
</tr>
<tr>
<td></td>
<td>• A redo entry is always considered for capture by a capture process, regardless of whether the redo entry has a non-NULL tag.</td>
</tr>
<tr>
<td></td>
<td>• A change is always considered for capture by a synchronous capture, regardless of whether the session that makes the change has a non-NULL tag.</td>
</tr>
<tr>
<td></td>
<td>• An LCR is always considered for apply by an apply process or dequeue by a messaging client, regardless of whether redo entry or LCR has a non-NULL tag.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then an Oracle Streams client performs its action only when the tag is NULL:</td>
</tr>
<tr>
<td></td>
<td>• A redo entry is considered for capture by a capture process only when the redo entry contains a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>• A change is considered for capture by a synchronous capture only when the session that makes the change has a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>• An LCR is considered for apply by an apply process or dequeue by a messaging client only if the LCR contains a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database.</td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle Streams Replication Administrator’s Guide for more information about tags</td>
</tr>
</tbody>
</table>
Table 165-11 (Cont.) ADD_SUBSET_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For synchronous capture rules, specify the name of the local database. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database. For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>insert_rule_name</td>
<td>Contains the system-generated INSERT rule name. This rule handles inserts and updates that must be converted into inserts.</td>
</tr>
<tr>
<td>update_rule_name</td>
<td>Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.</td>
</tr>
<tr>
<td>delete_rule_name</td>
<td>Contains the system-generated DELETE rule name. This rule handles deletes and updates that must be converted into deletes</td>
</tr>
</tbody>
</table>

Usage Notes

Running this procedure generates three rules for the specified capture process, synchronous capture, apply process, or messaging client: one for INSERT statements, one for UPDATE statements, and one for DELETE statements. For INSERT and DELETE statements, only DML changes that satisfy the condition specified for the dml_condition parameter are captured, applied, or dequeued. For UPDATE statements, the following variations are possible:

- If both the new and old values in a DML change satisfy the specified dml_condition, then the DML change is captured, applied, or dequeued without any changes.
- If neither the new or old values in a DML change satisfy the specified dml_condition, then the DML change is not captured, applied, or dequeued.
- If the old values for a DML change satisfy the specified dml_condition, but the new values do not, then the DML change is converted into a delete.
If the new values for a DML change satisfy the specified dml_condition, but the old values do not, then the DML change is converted to an insert.

When an update is converted into an insert or a delete, it is called row migration.

A capture process, synchronous capture, apply process, or messaging client uses the rules for filtering. If the Oracle Streams client does not have a positive rule set, then this procedure creates a positive rule set automatically, and adds the rules for the table to the positive rule set. A subset rule can be added to positive rule set only, not to a negative rule set. Other rules in an existing rule set for the process are not affected. Additional rules can be added using either the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The ADD_SUBSET_RULES procedure is overloaded, and the system-generated rule names for INSERT, UPDATE, and DELETE statements are returned.

**Note:**

Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so might have unpredictable results because row migration would not be performed on LCRs that are not discarded by the negative rule set.

**See Also:**

- "Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
- "Security Model"

**Examples**

The following is an example of a rule condition created for filtering DML changes containing an update operation when the dml_condition is region_id = 2, the table_name is hr.regions, and the source_database is dbs1.net:

```sql
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```
165.6.10 ADD_TABLE_PROPAGATION_RULES Procedure

This procedure adds table rules to the positive rule set for a propagation, or adds table rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

Syntax

DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);

DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);

Parameters

Table 165-12   ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
### Table 165-12  (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically.</td>
</tr>
<tr>
<td></td>
<td>If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue, and the queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link, specified as [schema_name.]queue_name[@dblink_name], if the destination queue is in a remote database. The queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter <a href="mailto:strmadmin.streams_queue@dbs2.net">strmadmin.streams_queue@dbs2.net</a> for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td></td>
<td>The generated rule evaluates to <strong>TRUE</strong> for any DDL change that operates on the table or on an object that is part of the table, such as an index or trigger on the table. The rule evaluates to <strong>FALSE</strong> for any DDL change that either does not refer to the table or refers to the table in a subordinate way. For example, the rule evaluates to <strong>FALSE</strong> for changes that create synonyms or views based on the table. The rule also evaluates to <strong>FALSE</strong> for a change to a PL/SQL subprogram that refers to the table.</td>
</tr>
</tbody>
</table>
Table 165-12 (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NUL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. See Also: Oracle Streams Replication Administrator’s Guide for more information about tags.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation. In either case, the system creates the rule set if it does not exist.</td>
</tr>
</tbody>
</table>
Table 165-12  (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| and_condition| If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:  
\[(system\_condition) AND (and\_condition)\]  
The variable in the specified condition must be :lcr. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:  
\[:lcr.get\_tag() = HEXTORAW(''02'')\]  
The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.  
See Also: Logical Change Record TYPES  
|
| queue_to_queue| If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database. If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  
• If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
• If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
• If NULL, then the procedure does not change the queue to queue property of the propagation.  
See Also: Oracle Streams Concepts and Administration for more information about queue-to-queue propagations  
|

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.
This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A propagation uses the rules for filtering.

See Also:

- "Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
- "Security Model"

Examples

The following is an example of a table rule condition created for filtering DML statements:

```sql
((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS')
 and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET')
```

165.6.11 ADD_TABLE_RULES Procedure

This procedure adds rules to a rule set for Oracle Streams clients. It adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the streams_type parameter is set to capture, this procedure adds capture process rules for capturing changes to a specified table.
- When the streams_type parameter is set to sync_capture, this procedure adds rules for capturing changes to a specified table.
- When the streams_type parameter is set to apply and the streams_name parameter specifies the name of an apply process, this procedure adds apply process rules for applying logical change records (LCRs) in a queue that contain changes to a specified table. The rules can specify that the LCRs must be from a particular source database.
- When the streams_type parameter is set to dequeue, this procedure adds messaging client rules for dequeuing persistent LCRs from a queue that contain changes to a specified table. The rules can specify that the LCRs must be from a particular source database.

This procedure creates the specified capture process, synchronous capture, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.
Note:

- Currently, messaging clients cannot dequeue buffered messages.
- The invoking user must be granted the DBA role to create a synchronous capture.

Syntax

```sql
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
  table_name          IN   VARCHAR2,
  streams_type        IN   VARCHAR2,
  streams_name        IN   VARCHAR2 DEFAULT NULL,
  queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
  include_dml         IN   BOOLEAN  DEFAULT TRUE,
  include_ddl         IN   BOOLEAN  DEFAULT FALSE,
  include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
  source_database     IN   VARCHAR2 DEFAULT NULL,
  dml_rule_name       OUT  VARCHAR2,
  ddl_rule_name       OUT  VARCHAR2,
  inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
  and_condition       IN   VARCHAR2  DEFAULT NULL);
```

Parameters

**Table 165-13  ADD_TABLE_RULES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default. You can specify a table that does not yet exist, because Oracle Streams does not validate the existence of the table.</td>
</tr>
</tbody>
</table>
| streams_type | The type of Oracle Streams client:  
  - Specify capture for a capture process.  
  - Specify sync_capture for a synchronous capture.  
  - Specify apply for an apply process.  
  - Specify dequeue for a messaging client. |
### Table 165-13  (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| streams_name  | The name of the capture process, synchronous capture, apply process, or messaging client. Do not specify an owner. If NULL, if streams_type is capture, sync_capture, or dequeue, and if one relevant capture process, synchronous capture, or messaging client for the queue exists, then the procedure uses the relevant Oracle Streams client. If no relevant Oracle Streams client exists for the queue, then the procedure creates an Oracle Streams client automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified streams_type for the queue exist, then the procedure raises an error. If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:
- If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.
- If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process.
If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error. Each apply process and messaging client must have a unique name. |
| queue_name    | The name of the local queue, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default. For capture process or synchronous capture rules, this is the queue into which a capture process or synchronous capture enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages. |
| include_dml   | If TRUE, then the procedure creates a DML rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted. |
Table 165-13  (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a DDL rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted. The generated rule evaluates to TRUE for any DDL change that operates on the table or on an object that is part of the table, such as an index or trigger on the table. The rule evaluates to FALSE for any DDL change that either does not refer to the table or refers to the table in a subordinate way. For example, the rule evaluates to FALSE for changes that create synonyms or views based on the table. The rule also evaluates to FALSE for a change to a PL/SQL subprogram that refers to the table.</td>
</tr>
</tbody>
</table>
Table 165-13  (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| include_tagged_lcr   | If **TRUE**, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to **TRUE** regardless of whether a redo entry, session, or LCR has a non-**NULL** tag. If the rules are added to the positive rule set for the Oracle Streams client, then the Oracle Streams client performs its action regardless of the tag:  
  - A redo entry is always considered for capture by a capture process, regardless of whether the redo entry has a non-**NULL** tag.  
  - A change is always considered for capture by a synchronous capture, regardless of whether the session that makes the change has a non-**NULL** tag.  
  - An LCR is always considered for apply by an apply process or dequeue by a messaging client, regardless of whether redo entry or LCR has a non-**NULL** tag.  

If the rules are added to a positive rule set, then setting this parameter to **TRUE** is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the Oracle Streams client, then whether a database change is discarded does not depend on the tag.  

If **FALSE**, then the procedure adds a condition to each generated rule that causes the rule to evaluate to **TRUE** only if a redo entry, session, or LCR has a **NULL** Oracle Streams tag. If the rules are added to the positive rule set for an Oracle Streams client, then the Oracle Streams client performs its action only when the tag is **NULL**:  
  - A redo entry is considered for capture by a capture process only when the redo entry contains a **NULL** tag.  
  - A change is considered for capture by a synchronous capture only when the session that makes the change has a **NULL** tag.  
  - An LCR is considered for apply by an apply process or dequeue by a messaging client only if the LCR contains a **NULL** tag.  

If the rules are added to a positive rule set, then setting this parameter to **FALSE** might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the Oracle Streams client, then a database change can be discarded only if it has a **NULL** tag.  

A setting of **FALSE** is often specified in update-anywhere configurations to avoid sending a change back to its source database.  

**Usually, specify **TRUE** for this parameter if the inclusion_rule parameter is set to **FALSE**.**

**See Also:** Oracle Streams Replication Administrator’s Guide for more information about tags.
Table 165-13  (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| source_database | The global name of the source database. If **NULL**, then the procedure does not add a condition regarding the source database to the generated rules.  
For capture process rules, specify **NULL** or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.  
For synchronous capture rules, specify the name of the local database.  
For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database.  
For messaging client rules, specify **NULL** if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information.  
If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. |
| dml_rule_name   | If **include_dml** is **TRUE**, then this parameter contains the DML rule name.  
If **include_dml** is **FALSE**, then this parameter contains a **NULL**. |
| ddl_rule_name   | If **include_ddl** is **TRUE**, then this parameter contains the DDL rule name.  
If **include_ddl** is **FALSE**, then this parameter contains a **NULL**. |
| inclusion_rule  | If **inclusion_rule** is **TRUE**, then the procedure adds the rules to the positive rule set for the Oracle Streams client.  
If **inclusion_rule** is **FALSE**, then the procedure adds the rules to the negative rule set for the Oracle Streams client. A synchronous capture cannot have a negative rule set. Specifying **FALSE** for a synchronous capture raises an error.  
In either case, the system creates the rule set if it does not exist. |
Table 165-13  (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and_condition</td>
<td>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</td>
</tr>
<tr>
<td></td>
<td>(system_condition) AND (and_condition)</td>
</tr>
<tr>
<td></td>
<td>The variable in the specified condition must be :lcr. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</td>
</tr>
<tr>
<td></td>
<td>:lcr.get_tag() = HEXTORAW('02')</td>
</tr>
<tr>
<td></td>
<td>The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</td>
</tr>
<tr>
<td></td>
<td>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</td>
</tr>
<tr>
<td></td>
<td>See Also: Logical Change Record TYPEs</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A capture process, synchronous capture, apply process, or messaging client uses the rules for filtering.

See Also:

- "Procedures That Create Rules for Oracle Streams Clients and XStream Clients"
- "Security Model"

Examples

The following is an example of a table rule condition created for DML changes:

```sql
{{:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'}}
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET'
```
165.6.12 CLEANUP_INSTANTIATION_SETUP Procedure

This procedure removes an Oracle Streams replication configuration that was set up by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures in this package. This procedure either remove the configuration directly, or it can generate a script that removes the configuration.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

**Note:**

When the CLEANUP_INSTANTIATION_SETUP procedure is run, the parameter values must match the parameter values specified when the corresponding PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures were run, except for the values of the following parameters: perform_actions, script_name, and script_directory_object.

**See Also:**

- "PRE_INSTANTIATION_SETUP Procedure"
- "POST_INSTANTIATION_SETUP Procedure"
- "Procedures That Configure an Oracle Streams Environment" for more information about this procedure

**Syntax**

```sql
DBMS_STREAMS_ADM.CLEANUP_INSTANTIATION_SETUP(
    maintain_mode  IN VARCHAR2,
    tablespace_names IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
    source_database IN VARCHAR2,
    destination_database IN VARCHAR2,
    perform_actions IN BOOLEAN   DEFAULT TRUE,
    script_name             IN VARCHAR2  DEFAULT NULL,
    script_directory_object IN VARCHAR2  DEFAULT NULL,
    capture_name            IN VARCHAR2  DEFAULT NULL,
    capture_queue_table     IN VARCHAR2  DEFAULT NULL,
    capture_queue_name      IN VARCHAR2  DEFAULT NULL,
    capture_queue_user      IN VARCHAR2  DEFAULT NULL,
    propagation_name        IN VARCHAR2  DEFAULT NULL,
    apply_name              IN VARCHAR2  DEFAULT NULL,
    apply_queue_table       IN VARCHAR2  DEFAULT NULL,
    apply_queue_name        IN VARCHAR2  DEFAULT NULL,
    apply_queue_user        IN VARCHAR2  DEFAULT NULL,
    bi_directional          IN BOOLEAN   DEFAULT FALSE,
    change_global_name      IN BOOLEAN   DEFAULT FALSE);
```
Parameters

Table 165-14  CLEANUP_INSTANTIATION_SETUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maintain_mode</td>
<td>Specify one of the following:</td>
</tr>
<tr>
<td></td>
<td>·  GLOBAL to clean up the Oracle Streams configuration that maintained the entire database in both the source and destination databases</td>
</tr>
<tr>
<td></td>
<td>·  TRANSPORTABLE TABLESPACES to cleanup the Oracle Streams configuration that maintained a set of tablespaces at both the source and destination database</td>
</tr>
<tr>
<td>tablespace_names</td>
<td>If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the local tablespace set to be cloned at the destination database and maintained by Oracle Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects. If maintain_mode is set to GLOBAL, then specify an empty tablespace set. Regardless of the maintain_mode setting, an error is raised if the tablespace_names parameter is not set or is set to NULL.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure uses the global name of the local database.</td>
</tr>
<tr>
<td>destination_database</td>
<td>The global name of the destination database. A database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>perform_actions</td>
<td>If TRUE, then this procedure performs the necessary actions to clean up the Oracle Streams configuration directly. If FALSE, then the procedure does not perform the necessary actions to clean up the Oracle Streams configuration directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</td>
</tr>
<tr>
<td></td>
<td>·  script_name</td>
</tr>
<tr>
<td></td>
<td>·  script_directory_object</td>
</tr>
</tbody>
</table>

See Also: TABLESPACE_SET Table Type
Table 165-14  (Cont.) CLEANUP_INSTANTIATION_SETUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to clean up the Oracle Streams configuration. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the statements are appended to the existing file. If non-NULL and the perform_actions parameter is TRUE, then this procedure generates the specified script and performs the actions to clean up the Oracle Streams configuration directly. If NULL and the perform_actions parameter is TRUE, then this procedure directly performs the actions to clean up the Oracle Streams configuration without generating a script. If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.</td>
</tr>
<tr>
<td>script_directory_object</td>
<td>The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then this parameter is ignored, and this procedure does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error. Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
<tr>
<td>capture_name</td>
<td>The name of the capture processes configured to capture changes in the Oracle Streams configuration. Do not specify an owner. If NULL, then the procedure automatically identifies the capture processes with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.</td>
</tr>
<tr>
<td>capture_queue_table</td>
<td>The name of the queue table for each queue used by a capture process, specified as [schema_name.]queue_table_name. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the procedure automatically identifies the capture queue tables with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.</td>
</tr>
<tr>
<td>capture_queue_name</td>
<td>The name of each queue used by a capture process, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If NULL, then the procedure automatically identifies the capture queues with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.</td>
</tr>
<tr>
<td>capture_queue_user</td>
<td>The name of the user who has ENQUEUE and DEQUEUE privileges for the queue at the source database. This user is a secure queue user of the queue.</td>
</tr>
</tbody>
</table>

Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.
Table 165-14  (Cont.) CLEANUP_INSTANTIATION_SETUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagations configured to propagate changes in the Oracle Streams configuration. Do not specify an owner. If NULL, then the procedure automatically identifies the propagations with system-generated names created by the \PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP\ procedures.</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of the apply processes configured to apply changes in the Oracle Streams configuration. Do not specify an owner. If NULL, then the procedure automatically identifies the apply processes with system-generated names created by the \PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP\ procedures.</td>
</tr>
<tr>
<td>apply_queue_table</td>
<td>The name of the queue table for each queue used by an apply process, specified as [schema_name.]queue_table_name. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the procedure automatically identifies the apply queue tables with system-generated names created by the \PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP\ procedures.</td>
</tr>
<tr>
<td>apply_queue_name</td>
<td>The name of each queue used by an apply process, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue. If NULL, then the procedure automatically identifies the apply queues with system-generated names created by the \PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP\ procedures.</td>
</tr>
<tr>
<td>apply_queue_user</td>
<td>The name of the user who has ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user is a secure queue user of the queue.</td>
</tr>
<tr>
<td>bi_directional</td>
<td>Specify TRUE if the Oracle Streams replication configuration is bi-directional between the database specified in source_database and the database specified in destination_database. Specify FALSE if the Oracle Streams replication configuration is one way replication from the current database to the database specified in destination_database.</td>
</tr>
<tr>
<td>change_global_name</td>
<td>If TRUE, then the procedure changes the global name of the database specified in destination_database to match the global name of the current database. If FALSE, then the procedure does not change the global name of the database specified in destination_database.</td>
</tr>
</tbody>
</table>
165.6.13 DELETE_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which deletes a column from a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

**Note:**
- The `DELETE_COLUMN` procedure supports the same data types supported by Oracle Streams capture processes.
- The `DELETE_COLUMN` procedure is useful when you want to delete a relatively small number of columns in a row LCR. To delete most of the columns in a row LCR and keep a relatively small number of columns, consider using the `KEEP_COLUMNS` procedure in this package.
- Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

**See Also:**
- Oracle Streams Concepts and Administration for more information about declarative rule-based transformations and about the data types supported by Oracle Streams capture processes
- "KEEP_COLUMNS Procedure"

**Syntax**

```sql
DBMS_STREAMS_ADM.DELETE_COLUMN(
  rule_name     IN  VARCHAR2,
  table_name    IN  VARCHAR2,
  column_name   IN  VARCHAR2,
  value_type    IN  VARCHAR2   DEFAULT '*",
  step_number   IN  NUMBER     DEFAULT 0,
  operation     IN  VARCHAR2   DEFAULT 'ADD');
```
Parameters

Table 165-15  DELETE_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then</td>
</tr>
<tr>
<td></td>
<td>the procedure raises an error. For example, to specify a rule in the hr</td>
</tr>
<tr>
<td></td>
<td>schema named employees12, enter hr.employees12. If the schema is not</td>
</tr>
<tr>
<td></td>
<td>specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table from which the column is deleted in the row LCR,</td>
</tr>
<tr>
<td></td>
<td>specified as [schema_name.]object_name. For example, hr.employees. If the</td>
</tr>
<tr>
<td></td>
<td>schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column deleted from each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to delete the column from the new values in the row LCR.</td>
</tr>
<tr>
<td></td>
<td>Specify 'OLD' to delete the column from the old values in the row LCR.</td>
</tr>
<tr>
<td></td>
<td>Specify '*' to delete the column from both the old and new values in the</td>
</tr>
<tr>
<td></td>
<td>row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule.</td>
</tr>
<tr>
<td></td>
<td>Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle Streams Concepts and Administration for more information</td>
</tr>
<tr>
<td></td>
<td>about transformation ordering</td>
</tr>
</tbody>
</table>

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the delete column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the DELETE_COLUMN procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>table_name</th>
<th>column_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all delete column transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NUL</td>
<td>Remove all delete column transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NUL</td>
<td>non-NUL</td>
<td>Remove all delete column transformations with the specified column_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NUL</td>
<td>NULL</td>
<td>non-NUL</td>
<td>Remove all delete column transformations with the specified table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NUL</td>
<td>NULL</td>
<td>Remove all delete column transformations with the specified column_name for the specified rule.</td>
</tr>
</tbody>
</table>
### 165.6.14 GET_MESSAGE_TRACKING Function

The `GET_MESSAGE_TRACKING` Function returns the tracking label for the current session.

**See Also:**

- `SET_MESSAGE_TRACKING Procedure`

**Syntax**

```
DBMS_STREAMS_ADM.GET_MESSAGE_TRACKING
RETURN VARCHAR2;
```

### 165.6.15 GET_SCN_MAPPING Procedure

This procedure gets information about the system change number (SCN) values to use for Oracle Streams capture and apply processes in an Oracle Streams replication environment.

This information can be used for the following purposes:

- To recover transactions after point-in-time recovery is performed on a source database in a multiple source Oracle Streams environment
- To run flashback queries for the corresponding SCN at a source database and destination database in an Oracle Streams single source replication environment

**See Also:**

- *Oracle Streams Replication Administrator’s Guide* for information about point-in-time recovery in an Oracle Streams replication environment
- *Oracle Streams Replication Administrator’s Guide* for information about flashback queries in an Oracle Streams replication environment
Syntax

```sql
DBMS_STREAMS_ADM.GET_SCN_MAPPING(
    apply_name             IN  VARCHAR2,
    src_pit_scn            IN  NUMBER,
    dest_instantiation_scn OUT NUMBER,
    dest_start_scn         OUT NUMBER,
    dest_skip_txn_ids      OUT DBMS_UTILITY.NAME_ARRAY);
```

Parameters

Table 165-16  GET_SCN_MAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_name</td>
<td>Name of the apply process which applies logical change records (LCRs) from the source database. The procedure raises an error if the specified apply process does not exist.</td>
</tr>
<tr>
<td>src_pit_scn</td>
<td>The SCN at the source database. For point-in-time recovery, specify the point-in-time recovery SCN at the source database.</td>
</tr>
<tr>
<td></td>
<td>If the specified SCN is greater than the source commit SCN of the last applied transaction, then NULL is returned for both dest_start_scn and dest_instantiation_scn. In this case, no values can be returned for these parameters because the corresponding transaction has not been applied at the destination database yet.</td>
</tr>
<tr>
<td>dest_instantiation_scn</td>
<td>The SCN at the destination database that corresponds to the specified src_pit_scn at the source database. For point-in-time recovery, use this value for the instantiation SCNs at the source database during recovery.</td>
</tr>
<tr>
<td>dest_start_scn</td>
<td>For point in time recovery, the SCN to use for the start_scn parameter for the recovery capture process.</td>
</tr>
<tr>
<td>dest_skip_txn_ids</td>
<td>Transaction IDs of transactions that were skipped at the dest_instantiation_scn because the apply process was applying nondependent transactions out of order. For point in time recovery, these transaction IDs should be ignored by the recovery apply process. This parameter is relevant only if the commit_serialization for the apply process that applied these transactions was set to DEPENDENT_TRANSACTIONS, and the transactions were applied out of order.</td>
</tr>
</tbody>
</table>
165.6.16 GET_TAG Function

This function gets the binary tag for all redo entries generated by the current session.

See Also:

- "SET_TAG Procedure"
- Oracle Streams Replication Administrator’s Guide for more information about tags

Syntax

DBMS_STREAMS_ADM.GET_TAG
RETURN RAW;

Examples

The following example illustrates how to display the current logical change record (LCR) tag as output:

SET SERVEROUTPUT ON
DECLARE
    raw_tag RAW(2000);
BEGIN
    raw_tag := DBMS_STREAMS_ADM.GET_TAG();
    DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/

You can also display the value by querying the DUAL view:

SELECT DBMS_STREAMS_ADM.GET_TAG FROM DUAL;

165.6.17 KEEP_COLUMNS Procedure

This procedure either adds or removes a declarative rule-based transformation which keeps a list of columns in a row logical change record (LCR) that satisfies the specified rule. The transformation deletes columns that are not in the list from the row LCR.

For the transformation to be performed when the specified rule evaluates to TRUE, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

This procedure is overloaded. The column_list parameter is type VARCHAR2 and the column_table parameter is type DBMS_UTILITY.LNAME_ARRAY. These parameters enable you to enter the list of columns in different ways and are mutually exclusive.
Note:

- The `KEEP_COLUMNS` procedure supports the same data types supported by Oracle Streams capture processes.
- The `KEEP_COLUMNS` procedure is useful when you want to keep a relatively small number of columns in a row LCR. To keep most of the columns in a row LCR and delete a relatively small number of columns, consider using the `DELETE_COLUMN` procedure in this package.
- Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

See Also:

- Oracle Streams Concepts and Administration for more information about declarative rule-based transformations
- Oracle Streams Concepts and Administration for more information about the data types supported by Oracle Streams capture processes
- "DELETE_COLUMN Procedure"

Syntax

```sql
DBMS_STREAMS_ADM.KEEP_COLUMNS(
    rule_name     IN  VARCHAR2,
    table_name    IN  VARCHAR2,
    column_list   IN  VARCHAR2,
    value_type    IN  VARCHAR2 DEFAULT '*",
    step_number   IN  NUMBER DEFAULT 0,
    operation     IN  VARCHAR2 DEFAULT 'ADD');

DBMS_STREAMS_ADM.KEEP_COLUMNS(
    rule_name     IN  VARCHAR2,
    table_name    IN  VARCHAR2,
    column_table  IN  DBMS_UTILITY.LNAME_ARRAY,
    value_type    IN  VARCHAR2 DEFAULT '*",
    step_number   IN  NUMBER DEFAULT 0,
    operation     IN  VARCHAR2 DEFAULT 'ADD');
```
Parameters

Table 165-17  KEEP_COLUMNS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current</td>
</tr>
<tr>
<td></td>
<td>user is the default.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table for which the columns are kept in the row LCR, specified as [schema_name.]object_name. For example, hr.employees. If the</td>
</tr>
<tr>
<td></td>
<td>schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>column_list</td>
<td>The names of the columns kept for each row LCR that satisfies the rule. Specify a comma-delimited list of type VARCHAR2. The transformation</td>
</tr>
<tr>
<td></td>
<td>removes columns that are not in the list from the row LCR.</td>
</tr>
<tr>
<td></td>
<td>If this parameter is set to NULL, and the column_table parameter is also set to NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>column_table</td>
<td>The names of the columns kept for each row LCR that satisfies the rule. Specify a PL/SQL associative array of type DBMS_UTILITY.LNAME_ARRAY,</td>
</tr>
<tr>
<td></td>
<td>where each element is the name of a column. The first schema should be in position 1. The last position must be NULL. The transformation</td>
</tr>
<tr>
<td></td>
<td>removes columns that are not in the table from the row LCR.</td>
</tr>
<tr>
<td></td>
<td>If this parameter is set to NULL, and the column_list parameter is also set to NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to keep the columns in the new values in the row LCR.</td>
</tr>
<tr>
<td></td>
<td>Specify 'OLD' to keep the columns in the old values in the row LCR.</td>
</tr>
<tr>
<td></td>
<td>Specify '*' to keep the columns in both the old and new values in the row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule.</td>
</tr>
<tr>
<td></td>
<td>Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
<tr>
<td></td>
<td>For more information about transformation ordering, see Oracle Streams Concepts and Administration. See &quot;Usage Notes&quot; for more information</td>
</tr>
<tr>
<td></td>
<td>about this parameter.</td>
</tr>
</tbody>
</table>

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the keep columns declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_list, column_table, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the KEEP_COLUMNS procedure when one or more of these parameters is NULL:
<table>
<thead>
<tr>
<th>table_name</th>
<th>column_list/ column_table</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified column_list/ column_table and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations with the specified column_list/ column_table for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations with the specified table_name and column_list/column_table for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations with the specified table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified table_name, column_list/column_table, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>

### 165.6.18 MAINTAIN_CHANGE_TABLE Procedure

This procedure configures an Oracle Streams environment that uses change handlers to record in a change table the data manipulation language (DML) changes made to a source table. Optionally, this procedure can also configure one-way replication of the table from the source database to the destination database.

This procedure can either configure the environment directly, or it can generate a script that configures the environment.

A change handler is a special type of statement DML handler that tracks table changes and was created by either this MAINTAIN_CHANGE_TABLE procedure or the DBMS_APPLY_ADM.SET_CHANGE_HANDLER procedure. Information about change handlers is stored in the ALL_APPLY_CHANGE_HANDLERS and DBA_APPLY_CHANGE_HANDLERS views.

The change table can reside in the same database as the source table or in a different database.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.
**Note:**

The environment configured by this procedure does not record or replicate data definition language (DDL) changes made to the source table.

**See Also:**

"Procedures That Configure an Oracle Streams Environment" for more information about this procedure

**Syntax**

```sql
DBMS_STREAMS_ADM.MAINTAIN_CHANGE_TABLE(
    change_table_name        IN VARCHAR2,
    source_table_name        IN VARCHAR2,
    column_type_list         IN VARCHAR2,
    extra_column_list        IN VARCHAR2  DEFAULT 'command_type, value_type',
    capture_values           IN VARCHAR2,
    options_string           IN VARCHAR2  DEFAULT NULL,
    script_name              IN VARCHAR2  DEFAULT NULL,
    script_directory_object  IN VARCHAR2  DEFAULT NULL,
    perform_actions          IN BOOLEAN   DEFAULT TRUE,
    capture_name             IN VARCHAR2  DEFAULT NULL,
    propagation_name         IN VARCHAR2  DEFAULT NULL,
    apply_name               IN VARCHAR2  DEFAULT NULL,
    source_database          IN VARCHAR2  DEFAULT NULL,
    destination_database     IN VARCHAR2  DEFAULT NULL,
    keep_change_columns_only IN BOOLEAN   DEFAULT TRUE,
    execute_lcr              IN BOOLEAN   DEFAULT FALSE);
```

**Parameters**

**Table 165-18  MAINTAIN_CHANGE_TABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>change_table_name</td>
<td>The table that records changes to the source table. This table is maintained by Oracle Streams after configuration. Specify the table as <code>[schema_name.]table_name</code>. For example, <code>hr.jobs_change_table</code>. If the schema is not specified, then the current user is the default. If <code>NULL</code>, then the procedure raises an error. If the specified table exists at the database specified in the <code>destination_database</code> parameter, then the procedure raises an error.</td>
</tr>
<tr>
<td>source_table_name</td>
<td>The table at the source database for which changes are recorded. Specify the table as <code>[schema_name.]table_name</code>. For example, <code>hr.jobs</code>. If the schema is not specified, then the current user is the default. If <code>NULL</code>, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
Table 165-18  (Cont.) MAINTAIN_CHANGE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_type_list</td>
<td>A list of the columns in the source table for which changes are recorded. Specify a comma-delimited list of each column and its datatype.</td>
</tr>
<tr>
<td></td>
<td>For example, specify the following for the hr.jobs table:</td>
</tr>
<tr>
<td></td>
<td>job_id VARCHAR2(10), job_title VARCHAR2(35), min_salary NUMBER(6), max_salary NUMBER(6)</td>
</tr>
<tr>
<td></td>
<td>The procedure automatically places columns with names that match the source database columns into an unconditional supplemental log group.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
Table 165-18 (Cont.) MAINTAIN_CHANGE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra_column_list</td>
<td>A comma-delimited list of metadata attributes to include in the change table. The column name for a metadata attribute is in the format of attribute name followed by a $ symbol. For example, the source_database_name attribute is stored in the source_database_name$ column in the change table. The following metadata attributes can be included: value_type, source_database_name, command_type, object_owner, object_name, tag, transaction_id, scn, commit_scn, compatible, instance_number, message_number, row_text, row_id, serial#, session#, source_time, thread#, tx_name, username. All of these metadata attributes, except for value_type and message_number, are row LCR attributes that can be stored in row LCRs. For information about LCR attributes, see Oracle Streams Concepts and Administration. The value_type$ column in the change table contains either OLD or NEW, depending on whether the column value is the original column value or the new column value, respectively. The message_number$ column in the change table contains the identification number of each row LCR within a transaction. The message number increases incrementally for each row LCR within a transaction and shows the order of the row LCRs within a transaction. The procedure automatically configures the source database to place information about extra attributes, such as serial#, into the redo log so that the information can be captured and recorded.</td>
</tr>
</tbody>
</table>
Table 165-18  (Cont.) MAINTAIN_CHANGE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_values</td>
<td>Specify which values to capture when update operations are performed on the source table:</td>
</tr>
<tr>
<td></td>
<td>· old - To capture the original values for an updated column in the source table</td>
</tr>
<tr>
<td></td>
<td>· new - To capture the new values for an updated column in the source table</td>
</tr>
<tr>
<td></td>
<td>· * - To capture both the original and the new values for an updated column in the source table</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>Note:</td>
<td>For insert operations, only new column values can be captured. For delete operations, only old column values can be captured.</td>
</tr>
<tr>
<td>options_string</td>
<td>String of options passed to the CREATE TABLE statement that creates the change table. The string is appended to the generated CREATE TABLE statement after the closing parenthesis that defines the columns of the table. The string must be syntactically correct.</td>
</tr>
<tr>
<td>script_name</td>
<td>If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to configure the environment. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file.</td>
</tr>
<tr>
<td></td>
<td>If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to configure the replication environment directly.</td>
</tr>
<tr>
<td></td>
<td>If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to configure the replication environment directly and does not generate a script.</td>
</tr>
<tr>
<td></td>
<td>If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.</td>
</tr>
<tr>
<td>script_directory_object</td>
<td>The directory object for the directory on the local computer system into which the generated script is placed.</td>
</tr>
<tr>
<td></td>
<td>If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script.</td>
</tr>
<tr>
<td></td>
<td>If NULL and the script_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>Note:</td>
<td>The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>perform_actions</td>
<td>If TRUE, then the procedure performs the necessary actions to configure the environment directly. If FALSE, then the procedure does not perform the necessary actions to configure the environment directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL: • script_name • script_directory_object</td>
</tr>
<tr>
<td>capture_name</td>
<td>The name of each capture process configured to capture changes. Do not specify an owner. If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database to the positive capture process rule set. If NULL, then the system generates a name for each capture process it creates. <strong>Note:</strong> The capture process name cannot be altered after the capture process is created.</td>
</tr>
<tr>
<td>propagation_name</td>
<td>The name of the propagation configured to propagate changes from the source database to the destination database. Do not specify an owner. If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the positive propagation rule set. If NULL, then the system generates a name for the propagation. If non-NULL and the source_database and destination_database are set to the same value, then this procedure raises an error. When the capture process and apply process are in the same database, they can use the same queue, and a propagation is not needed. <strong>Note:</strong> The propagation name cannot be altered after the propagation is created.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of each apply process configured to apply changes. Do not specify an owner. If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the positive apply process rule set. The specified name must not match the name of an existing messaging client at the destination database. If NULL, then the system generates a name for the apply process. When set to NULL, no apply process that applies changes from the source database can exist on the destination database. If an apply process that applies changes from the source database exists at the destination database, then specify a non-NULL value for this parameter. Note: The apply process name cannot be altered after the apply process is created.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database. If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure uses the global name of the local database.</td>
</tr>
<tr>
<td>destination_database</td>
<td>The global name of the destination database. If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure uses the global name of the local database.</td>
</tr>
</tbody>
</table>
Table 165-18  (Cont.) MAINTAIN_CHANGE_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep_change_columns_only</td>
<td>If TRUE, then this procedure adds a declarative rule-based transformation which keeps the list of columns specified in the column_type_list parameter. The columns that are not specified in the column_type_list parameter are removed from each row LCR captured by the capture process. &lt;br&gt; If FALSE, then this procedure does not create a declarative rule-based transformation, and all of the columns in the row LCRs are kept. &lt;br&gt; Specify FALSE when information about columns that are not included in the column_type_list parameter is needed at the destination database. For example, if the execute_lcr parameter is set to TRUE and the configuration will replicate all of the columns in a source table, but the column_type_list parameter includes a subset of these columns, then the keep_change_columns_only parameter should be set to FALSE. &lt;br&gt; <strong>Note:</strong> When this parameter is set to TRUE, a declarative rule-based transformation is always created, even if the column_type_list includes all of the columns in the source table.</td>
</tr>
<tr>
<td>execute_lcr</td>
<td>If TRUE, then this procedure creates a change handler that executes each row LCR at the destination database. &lt;br&gt; If FALSE, then the row LCRs are not executed at the destination database.</td>
</tr>
</tbody>
</table>

Usage Notes

The following are usage notes for this procedure:

Types of Oracle Streams Environments Configured by the Procedure

This procedure can configure the following types of Oracle Streams environments:

- **Local capture and apply on one database**: Specify the same global name for the source_database and the destination_database parameter.
- **Local capture and remote apply**: Specify the global name of the local database for the source_database parameter and a remote database for the destination_database parameter.
- **Downstream capture and local apply**: Specify a remote database for the source_database parameter and the local database for the destination_database parameter.
- **Downstream capture and remote apply**: Specify a remote database for the source_database parameter and a remote database for the destination_database parameter.

Optional One-Way Replication With This Procedure

To configure one-way replication of the table, in addition to recording changes to the table, set the execute_lcr parameter to TRUE. The apply process executes each row LCR and applies the change in the row LCR to the replica table at the destination database.
database. In this case, ensure that the source table is instantiated at the destination database before running the procedure. Specifically, the source table must be prepared for instantiation, the instantiation SCN must be set for the replica table at the destination database, and, usually, the source table replica table should be consistent.

See Also:
Oracle Streams Replication Administrator's Guide

Statement DML Handlers, the Change Table, and Row LCR Execution

This procedure configures one or more statement DML handlers that perform the following actions:

- Record changes in the change table using the information in row LCRs.
- Execute row LCRs if the `execute_lcr` parameter is set to `TRUE`.

The procedure ensures that the row LCRs contain the required attributes to record the changes specified in the `capture_type_list`, `extra_column_list`, and `capture_values` parameters. The procedure adds the statement DML handlers to the apply process specified in the `apply_name` parameter.

See Also:
DBMS_STREAMS_HANDLER_ADM

165.6.19 MAINTAIN_GLOBAL Procedure

This procedure configures an Oracle Streams environment that replicates changes at the database level between two databases. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.
Note:

- This procedure automatically excludes database objects that are not supported by Oracle Streams from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
- If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the destination database while the `MAINTAIN_GLOBAL` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.
- A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. This procedure does not configure replication for these schemas.

See Also:

"Procedures That Configure an Oracle Streams Environment" for more information about this procedure

Syntax

```sql
DBMS_STREAMS_ADM.MAINTAIN_GLOBAL(
    source_directory_object      IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database              IN VARCHAR2,
    destination_database         IN VARCHAR2,
    perform_actions              IN BOOLEAN   DEFAULT TRUE,
    script_name                  IN VARCHAR2  DEFAULT NULL,
    script_directory_object      IN VARCHAR2  DEFAULT NULL,
    dump_file_name               IN VARCHAR2  DEFAULT NULL,
    capture_name                 IN VARCHAR2  DEFAULT NULL,
    capture_queue_table          IN VARCHAR2  DEFAULT NULL,
    capture_queue_name           IN VARCHAR2  DEFAULT NULL,
    capture_queue_user           IN VARCHAR2  DEFAULT NULL,
    propagation_name             IN VARCHAR2  DEFAULT NULL,
    apply_name                    IN VARCHAR2  DEFAULT NULL,
    apply_queue_table            IN VARCHAR2  DEFAULT NULL,
    apply_queue_name             IN VARCHAR2  DEFAULT NULL,
    apply_queue_user             IN VARCHAR2  DEFAULT NULL,
    log_file                      IN VARCHAR2  DEFAULT NULL,
    bi_directional               IN BOOLEAN   DEFAULT FALSE,
    include_ddl                  IN BOOLEAN   DEFAULT FALSE,
    instantiation               IN INTEGER   DEFAULT
)
```

```sql
DBMS_STREAMS_ADM.INSTANTIATION_FULL);
```
Parameters

See Also:

"Procedures That Configure an Oracle Streams Environment" for descriptions of the common procedure parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_directory_object</td>
<td>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remains in this directory after the procedure completes. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the source_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the procedure raises an error. <strong>Note:</strong> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
<tr>
<td>destination_directory_object</td>
<td>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred. If the source database and destination database run on the same computer system, then the source and destination directories must be different. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In these cases, specify NULL for the destination_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the procedure raises an error. <strong>Note:</strong> The specified directory object cannot point to an Oracle ASM disk group.</td>
</tr>
</tbody>
</table>
Table 165-19  (Cont.) MAINTAIN_GLOBAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| source_database       | The global name of the source database.  
If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.  
If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.  
If NULL, then the procedure uses the global name of the local database. |
| destination_database  | The global name of the destination database.  
If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.  
If NULL, then the procedure raises an error. |
| dump_file_name        | The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error.  
This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE.  
If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the export dump file name is generated by the system. In this case, the export dump file name is expat.nn.dmp, where nn is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory. |
| log_file              | The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file.  
This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE.  
If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_FULL, then the log file name is the same name as the export dump file name with an extension of .clg. |
Table 165-19  (Cont.) MAINTAIN_GLOBAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instantiation</td>
<td>Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation:</td>
</tr>
<tr>
<td></td>
<td><strong>DBMS_STREAMS_ADM.INSTANTIATION_FULL</strong> performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then the user who runs this procedure must have EXECUTE privilege on the DBMS_FILE_TRANSFER package.</td>
</tr>
<tr>
<td></td>
<td><strong>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</strong> performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</td>
</tr>
<tr>
<td></td>
<td><strong>DBMS_STREAMS_ADM.INSTANTIATION_NONE</strong> does not perform an instantiation. This setting is valid only if the perform_actions parameter is set to FALSE, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly. If you use the RMAN DUPLICATE or CONVERT DATABASE command for database instantiation, then the destination database cannot be the capture database. If this parameter is set to <strong>DBMS_STREAMS_ADM.INSTANTIATION_FULL</strong> or <strong>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</strong>, then the database objects being instantiated must exist at the source database.</td>
</tr>
<tr>
<td></td>
<td>If an instantiated database object does not exist at the destination database, then it is imported at the destination database, including its supplemental logging specifications from the source database and its supporting database objects, such as indexes and triggers. However, if the database object exists at the destination database before instantiation, then it is not imported at the destination database. Therefore, the supplemental logging specifications from the source database are not specified for the database object at the destination database, and the supporting database objects are not imported.</td>
</tr>
</tbody>
</table>
165.6.20 MAINTAIN_SCHEMAS Procedure

This procedure configures an Oracle Streams environment that replicates changes to specified schemas between two databases.

This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

This procedure is overloaded. One `schema_names` parameter is type `VARCHAR2` and the other `schema_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the list of schemas in different ways and are mutually exclusive.

**Note:**

- This procedure automatically excludes database objects that are not supported by Oracle Streams in the schemas from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

- If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the shared database objects at the destination database while the `MAINTAIN_SCHEMAS` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.

**See Also:**

"Procedures That Configure an Oracle Streams Environment" for more information about this procedure

**Syntax**

```sql
DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 IN VARCHAR2,
    source_directory_object      IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database              IN VARCHAR2,
    destination_database         IN VARCHAR2,
    perform_actions              IN BOOLEAN   DEFAULT TRUE,
    script_name                  IN VARCHAR2  DEFAULT NULL,
    script_directory_object      IN VARCHAR2  DEFAULT NULL,
    dump_file_name               IN VARCHAR2  DEFAULT NULL,
    capture_name                 IN VARCHAR2  DEFAULT NULL,
    capture_queue_table          IN VARCHAR2  DEFAULT NULL,
    capture_queue_name           IN VARCHAR2  DEFAULT NULL,
)
```
Parameters

See Also:

"Procedures That Configure an Oracle Streams Environment" for descriptions of the common procedure parameters that are not in Table 165-20
Table 165-20 MAINTAIN_SCHEMAS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_names</td>
<td>The schemas to be configured for replication and maintained by Oracle Streams after configuration. The schemas can be specified in the following ways:</td>
</tr>
<tr>
<td></td>
<td>- Comma-delimited list of type VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>- A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a schema. The first schema should be in position 1. The last position must be NULL.</td>
</tr>
<tr>
<td></td>
<td>This procedure raises an error in any of the following cases:</td>
</tr>
<tr>
<td></td>
<td>- When a specified schema does not exist at the source database</td>
</tr>
<tr>
<td></td>
<td>- When the schema_names parameter is set to NULL</td>
</tr>
<tr>
<td>source_directory_object</td>
<td>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remains in this directory after the procedure completes.</td>
</tr>
<tr>
<td></td>
<td>This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE.</td>
</tr>
<tr>
<td></td>
<td>In this case, specify NULL for the source_directory_object parameter.</td>
</tr>
<tr>
<td></td>
<td>If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
<tr>
<td>destination_directory_object</td>
<td>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred.</td>
</tr>
<tr>
<td></td>
<td>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</td>
</tr>
<tr>
<td></td>
<td>This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE.</td>
</tr>
<tr>
<td></td>
<td>In this case, specify NULL for the destination_directory_object parameter.</td>
</tr>
<tr>
<td></td>
<td>If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>Note: The specified directory object cannot point to an Oracle ASM disk group.</td>
</tr>
</tbody>
</table>
Table 165-20  (Cont.) MAINTAIN_SCHEMAS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database. If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database. If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure uses the global name of the local database.</td>
</tr>
<tr>
<td>destination_database</td>
<td>The global name of the destination database. If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>dump_file_name</td>
<td>The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the export dump file name is expatnn.dmp, where nn is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</td>
</tr>
<tr>
<td>capture_queue_user</td>
<td>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</td>
</tr>
</tbody>
</table>
Table 165-20  (Cont.) MAINTAIN_SCHEMAS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_file</td>
<td>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA, then the log file name is the same name as the export dump file name with an extension of .clg.</td>
</tr>
</tbody>
</table>
Table 165-20  (Cont.) MAINTAIN_SCHEMAS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| instantiation           | Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation:  
  DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then the user who runs this procedure must have EXECUTE privilege on the DBMS_FILE_TRANSFER package.  
  DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. The instantiation SCN is set for the shared database objects during import. If the instantiation parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.  
  DBMS_STREAMS_ADM.INSTANTIATION_NONE does not perform an instantiation. This setting is valid only if the perform_actions parameter is set to FALSE, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly.  
  If this parameter is set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA or DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK, then the database objects being instantiated must exist at the source database, and the tablespaces that contain the schemas must exist at the destination database.  
  If an instantiated database object does not exist at the destination database, then it is imported at the destination database, including its supplemental logging specifications from the source database and its supporting database objects, such as indexes and triggers. However, if the database object exists at the destination database before instantiation, then it is not imported at the destination database. Therefore, the supplemental logging specifications from the source database are not specified for the database object at the destination database, and the supporting database objects are not imported. |
165.6.21 MAINTAIN_SIMPLE_TABLESPACE Procedure

This procedure clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases.

This procedure can either perform these actions directly, or it can generate a script that performs these actions. Run this procedure at the source database.

Note:

This procedure is deprecated. It is replaced by the MAINTAIN_SIMPLE_TTS procedure.

See Also:

- "Deprecated Subprograms"
- MAINTAIN_SIMPLE_TTS Procedure

Syntax

DBMS_STREAMS_ADM.MAINTAIN_SIMPLE_TABLESPACE(
  tablespace_name  IN VARCHAR2,
  source_directory_object  IN VARCHAR2,
  destination_directory_object  IN VARCHAR2,
  destination_database  IN VARCHAR2,
  setup_streams  IN BOOLEAN   DEFAULT TRUE,
  script_name  IN VARCHAR2  DEFAULT NULL,
  script_directory_object  IN VARCHAR2  DEFAULT NULL,
  bi_directional  IN BOOLEAN   DEFAULT FALSE);

Parameters

Table 165-21  MAINTAIN_SIMPLE_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>The local simple tablespace to be cloned at the destination database and maintained by Oracle Streams.</td>
</tr>
<tr>
<td></td>
<td>The tablespace must exist at the source database, but it must not exist at the destination database.</td>
</tr>
<tr>
<td></td>
<td>A directory object must exist for the directory that contains the datafile for the tablespace. The user who invokes this procedure must have READ privilege on this directory object. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| source_directory_object | The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are placed. These files remain in this directory after the procedure completes.  
If NULL, then the procedure raises an error.  
**Note:** The specified directory object cannot point to an Oracle ASM disk group.                                                                                                                       |
| destination_directory_object | The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are transferred.  
If the source database and destination database run on the same computer system, then the source and destination directories must be different.  
If NULL, then the procedure raises an error.  
**Note:** The specified directory object cannot point to an Oracle ASM disk group.                                                                                                                       |
| destination_database  | The global name of the destination database. A database link from the source database to the destination database with the same name as the global name of the destination database must exist.  
If NULL, then the procedure raises an error.                                                                                                                                                  |
| setup_streams         | If TRUE, then the procedure performs the necessary actions to maintain the tablespace directly.  
If FALSE, then the procedure does not perform the necessary actions to maintain the tablespace directly.  
Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:  
  * script_name  
  * script_directory_object                                                                                                                                           |
Table 165-21  (Cont.) MAINTAIN_SIMPLE_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>If non-NULL and the setup_streams parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to maintain the specified tablespace. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file. If non-NULL and the setup_streams parameter is TRUE, then this procedure generates the specified script and performs the actions to maintain the specified tablespace directly. If NULL and the setup_streams parameter is TRUE, then this procedure does not generate a script and performs the actions to maintain the specified tablespace directly. If NULL and the setup_streams parameter is FALSE, then the procedure raises an error.</td>
</tr>
<tr>
<td>script_directory_object</td>
<td>The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error. Note: The specified directory object cannot point to an Oracle ASM disk group.</td>
</tr>
<tr>
<td>bi_directional</td>
<td>Specify TRUE to configure bi-directional replication between the current database and the database specified in destination_database. Both databases are configured as source and destination databases, a capture and apply process is configured at both databases, and propagations are configured between the databases to propagate messages. Specify FALSE to configure one way replication from the current database to the database specified in destination_database. A capture process is configured at the current database, a propagation is configured to propagate messages from the current database to the destination database, and an apply process is configured at the destination database.</td>
</tr>
</tbody>
</table>

Usage Notes

The specified tablespace must be a simple tablespace. A simple tablespace is a single, self-contained tablespace that uses only one datafile. A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. This procedure cannot be used for a non simple tablespace or a set of tablespaces.
DDL Changes Not Maintained

This procedure does not configure the Oracle Streams environment to maintain DDL changes to the tablespace nor to the database objects in the tablespace. For example, the Oracle Streams environment is not configured to replicate \texttt{ALTER TABLESPACE} statements on the tablespace, nor is it configured to replicate \texttt{ALTER TABLE} statements on tables in the tablespace. You can configure the Oracle Streams environment to maintain DDL changes manually or modify generated scripts to achieve this.

Additional Documentation for this Procedure

The following documentation applies to the \texttt{MAINTAIN\_SIMPLE\_TABLESPACE} procedure:

- "Procedures That Create Rules for Oracle Streams Clients and XStream Clients" for information about automatic platform conversion
- \textit{Oracle Streams Replication Administrator's Guide}

Requirements for Running this Procedure

Meet the following requirements when run the \texttt{MAINTAIN\_SIMPLE\_TABLESPACE} procedure:

- Run the procedure at the source database.
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the \texttt{destination\_database} parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- Grant the user who runs this procedure the \texttt{DBA} role. This user must have the necessary privileges to complete the following actions:
  - Create \texttt{ANYDATA} queues, capture processes, propagations, and apply processes.
  - Specify supplemental logging
  - Run subprograms in the \texttt{DBMS\_STREMS\_ADM} and \texttt{DBMS\_AQADM} packages.
  - Access the database specified in the \texttt{destination\_database} parameter through a database link. This database link should have the same name as the global name of the destination database.
  - Run subprograms in the \texttt{DBMS\_STREMS\_TABLESPACES\_ADM} package
  - The necessary privileges to run the \texttt{CLONE\_SIMPLE\_TABLESPACE} procedure in the \texttt{DBMS\_STREMS\_TABLESPACES\_ADM} package at the source database. See \texttt{CLONE\_SIMPLE\_TABLESPACE Procedure} for the list of required privileges.
  - The necessary privileges to run the \texttt{ATTACH\_SIMPLE\_TABLESPACE} procedure in the \texttt{DBMS\_STREMS\_TABLESPACES\_ADM} package at the destination database. See \texttt{ATTACH\_SIMPLE\_TABLESPACE Procedure} for the list of required privileges.

To ensure that the user who runs this procedure has the necessary privileges, you should configure an Oracle Streams administrator at each database, and each database link should be created in the Oracle Streams administrator's schema.

Typically, the \texttt{DBA} role can be revoked from the user, if necessary, after the configuration is complete.

- If the \texttt{bi\_directional} parameter is set to \texttt{TRUE}, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.
• Each specified directory object must be created using the SQL statement `CREATE DIRECTORY`, and the user who invokes this procedure must have `READ` and `WRITE` privilege on each one.

• The databases configured by this procedure must be Oracle Database 10g Release 2 or later databases when this procedure is run under the following conditions:
  – The procedure is run at an Oracle Database 10g Release 2 or later database.
  – The `setup_streams` parameter is set to `TRUE` to configure the Oracle Streams replication environment directly.

• The databases configured by this procedure must be Oracle Database 10g Release 1 or later databases when this procedure is run under the following conditions:
  – The procedure is run at an Oracle Database 10g Release 2 or later database.
  – The `setup_streams` parameter is set to `FALSE` in this procedure, and the replication environment is configured with a generated script.

If the script configures an Oracle Database 10g Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10g Release 2 or later, such as queue-to-queue propagation.

• If the procedure is run at an Oracle Database 10g Release 1 database, then the databases configured by the procedure must be Oracle Database 10g Release 1 or later databases.

See Also:

*Oracle Streams Replication Administrator’s Guide* for information about configuring an Oracle Streams administrator

Default Values for Parameters Excluded From the MAINTAIN_SIMPLE_TABLESPACE Procedure

This procedure uses the default values for the parameters in the `MAINTAIN_TABLESPACES` procedure that do not exist in the `MAINTAIN_SIMPLE_TABLESPACE` procedure. For example, this procedure creates a capture process at the source database named `capture`, because that is the default value for the `capture_name` parameter in the `MAINTAIN_TABLESPACES` procedure.

See Also:

`MAINTAIN_TABLESPACES` Procedure

Configuration Progress and Recoverability

When this procedure is run with the `setup_streams` parameter set to `TRUE`, metadata about its configuration actions is recorded in the following data dictionary views: `DBA_RECOVERABLE_SCRIPT`, `DBA_RECOVERABLE_SCRIPT_PARAMS`,
DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then you can use the RECOVER_OPERATION procedure to complete the configuration after you correct the conditions that caused the error.

**Note:**

When this procedure is run with the setup_streams parameter set to FALSE, these views are not populated. Also, the views are not populated when a script generated by this procedure is run.

**See Also:**

"RECOVER_OPERATION Procedure"

### 165.6.22 MAINTAIN_SIMPLE_TTS Procedure

This procedure clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases.

This procedure can either perform these actions directly, or it can generate a script that performs these actions.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

**Note:**

- This procedure automatically excludes database objects that are not supported by Oracle Streams in the tablespace from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the DBA_STREAMS_UNSUPPORTED data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
- This procedure replaces the deprecated MAINTAIN_SIMPLE_TABLESPACE procedure.

**See Also:**

"Procedures That Configure an Oracle Streams Environment“ for more information about this procedure
Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_SIMPLE_TTS(
    tablespace_name              IN VARCHAR2,
    source_directory_object      IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database              IN VARCHAR2,
    destination_database         IN VARCHAR2,
    perform_actions              IN BOOLEAN   DEFAULT TRUE,
    script_name                  IN VARCHAR2  DEFAULT NULL,
    script_directory_object      IN VARCHAR2  DEFAULT NULL,
    bi_directional               IN BOOLEAN   DEFAULT FALSE);
```

Parameters

For procedures that configure an Oracle Streams environment, see Table 165-22 for a summary of the common procedure parameters that are not in Table 165-22.

### See Also:

- "Procedures That Configure an Oracle Streams Environment" for descriptions of the common procedure parameters that are not in Table 165-22.

### Table 165-22  MAINTAIN_SIMPLE_TTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>The local simple tablespace to be cloned at the destination database and maintained by Oracle Streams. The tablespace must exist at the source database, but it must not exist at the destination database. A directory object must exist for the directory containing the datafile for the tablespace. The user who invokes this procedure must have <code>READ</code> privilege on this directory object. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group. If <code>NULL</code>, then the procedure raises an error.</td>
</tr>
<tr>
<td>source_directory_object</td>
<td>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are placed. These files remain in this directory after the procedure completes. If <code>NULL</code>, then the procedure raises an error. <strong>Note:</strong> The specified directory object cannot point to an Oracle ASM disk group.</td>
</tr>
</tbody>
</table>
### Table 165-22 (Cont.) MAINTAIN_SIMPLE_TTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>destination_directory_object</strong></td>
<td>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are transferred. If the source database and destination database run on the same computer system, then the source and destination directories must be different. If <strong>NULL</strong>, then the procedure raises an error. <strong>Note:</strong> The specified directory object cannot point to an Oracle ASM disk group.</td>
</tr>
<tr>
<td><strong>source_database</strong></td>
<td>The global name of the source database. If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database. If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. If <strong>NULL</strong>, then the procedure uses the global name of the local database.</td>
</tr>
<tr>
<td><strong>destination_database</strong></td>
<td>The global name of the destination database. If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If <strong>NULL</strong>, then the procedure raises an error.</td>
</tr>
</tbody>
</table>

### Usage Notes

The specified tablespace must be a simple tablespace. A simple tablespace is a single, self-contained tablespace that uses only one datafile. A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. This procedure cannot be used for a non simple tablespace or a set of tablespaces.

**DDL Changes Not Maintained**

This procedure does not configure the Oracle Streams environment to maintain DDL changes to the tablespace nor to the database objects in the tablespace. For example, the Oracle Streams environment is not configured to replicate `ALTER TABLESPACE` statements on the tablespace, nor is it configured to replicate `ALTER TABLE` statements on tables in the tablespace. You can configure the Oracle Streams environment to maintain DDL changes manually or modify generated scripts to achieve this.
Additional Privileges Required by the MAINTAIN_SIMPLE_TTS Procedure

In addition to the required privileges described in "Procedures That Configure an Oracle Streams Environment", the user who runs the MAINTAIN_SIMPLE_TTS procedure must have the necessary privileges to complete the following actions:

- Run subprograms in the DBMS_STREAMS_TABLESPACES_ADM package
- The necessary privileges to run the CLONE_SIMPLE_TABLESPACE procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the source database. See CLONE_SIMPLE_TABLESPACE Procedure for the list of required privileges.
- The necessary privileges to run the ATTACH_SIMPLE_TABLESPACE procedure in the DBMS_STREAMS_TABLESPACES_ADM package at the destination database. See ATTACH_SIMPLE_TABLESPACE Procedure for the list of required privileges.

Default Values for Parameters Excluded From the MAINTAIN_SIMPLE_TTS Procedure

This procedure uses the default values for the parameters in the MAINTAIN_TTS procedure that do not exist in the MAINTAIN_SIMPLE_TTS procedure. For example, this procedure automatically generates the capture process name, because NULL is the default value for the capture_name parameter in the MAINTAIN_TTS procedure, and the procedure generates the capture process name when NULL is specified for capture_name.

See Also:
MAINTAIN_TTS Procedure

165.6.23 MAINTAIN_TABLES Procedure

This procedure configures an Oracle Streams environment that replicates changes to specified tables between two databases.

This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

This procedure is overloaded. One table_names parameter is type VARCHAR2 and the other table_names parameter is type DBMS_UTILITY.UNCL_ARRAY. These parameters enable you to enter the list of tables in different ways and are mutually exclusive.

Note:
If the bi_directional parameter is set to TRUE, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the shared database objects at the destination database while the MAINTAIN_TABLES procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.
See Also:

"Procedures That Configure an Oracle Streams Environment" for more information about this procedure

Syntax

DBMS_STREAMS_ADM.MAINTAIN_TABLES(
  table_names                  IN VARCHAR2,
  source_directory_object      IN VARCHAR2,
  destination_directory_object IN VARCHAR2,
  source_database              IN VARCHAR2,
  destination_database         IN VARCHAR2,
  perform_actions              IN BOOLEAN   DEFAULT TRUE,
  script_name                  IN VARCHAR2  DEFAULT NULL,
  script_directory_object      IN VARCHAR2  DEFAULT NULL,
  dump_file_name               IN VARCHAR2  DEFAULT NULL,
  capture_name                 IN VARCHAR2  DEFAULT NULL,
  capture_queue_table          IN VARCHAR2  DEFAULT NULL,
  capture_queue_name           IN VARCHAR2  DEFAULT NULL,
  capture_queue_user           IN VARCHAR2  DEFAULT NULL,
  propagation_name             IN VARCHAR2  DEFAULT NULL,
  apply_name                   IN VARCHAR2  DEFAULT NULL,
  apply_queue_table            IN VARCHAR2  DEFAULT NULL,
  apply_queue_name             IN VARCHAR2  DEFAULT NULL,
  apply_queue_user             IN VARCHAR2  DEFAULT NULL,
  log_file                     IN VARCHAR2  DEFAULT NULL,
  bi_directional               IN BOOLEAN   DEFAULT FALSE,
  include_ddl                  IN BOOLEAN   DEFAULT FALSE,
  instantiation                IN INTEGER   DEFAULT DBMS_STREAMS_ADM.INSTANTIATION_TABLE);

DBMS_STREAMS_ADM.MAINTAIN_TABLES(
  table_names                  IN DBMS_UTILITY.UNCL_ARRAY,
  source_directory_object      IN VARCHAR2,
  destination_directory_object IN VARCHAR2,
  source_database              IN VARCHAR2,
  destination_database         IN VARCHAR2,
  perform_actions              IN BOOLEAN   DEFAULT TRUE,
  script_name                  IN VARCHAR2  DEFAULT NULL,
  script_directory_object      IN VARCHAR2  DEFAULT NULL,
  dump_file_name               IN VARCHAR2  DEFAULT NULL,
  capture_name                 IN VARCHAR2  DEFAULT NULL,
  capture_queue_table          IN VARCHAR2  DEFAULT NULL,
  capture_queue_name           IN VARCHAR2  DEFAULT NULL,
  capture_queue_user           IN VARCHAR2  DEFAULT NULL,
  propagation_name             IN VARCHAR2  DEFAULT NULL,
  apply_name                   IN VARCHAR2  DEFAULT NULL,
  apply_queue_table            IN VARCHAR2  DEFAULT NULL,
  apply_queue_name             IN VARCHAR2  DEFAULT NULL,
  apply_queue_user             IN VARCHAR2  DEFAULT NULL,
  log_file                     IN VARCHAR2  DEFAULT NULL,
  bi_directional               IN BOOLEAN   DEFAULT FALSE,
  include_ddl                  IN BOOLEAN   DEFAULT FALSE,
  instantiation                IN INTEGER   DEFAULT DBMS_STREAMS_ADM.INSTANTIATION_TABLE);
Parameters

See Also:

"Procedures That Configure an Oracle Streams Environment" for descriptions of the common procedure parameters that are not in Table 165-23

Table 165-23  MAINTAIN_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_names</td>
<td>The tables to be configured for replication and maintained by Oracle Streams after configuration. The tables can be specified in the following ways:</td>
</tr>
<tr>
<td></td>
<td>• Comma-delimited list of type VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>• A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table. The first table should be in position 1. The last position must be NULL.</td>
</tr>
<tr>
<td></td>
<td>Each table should be specified as [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>This procedure raises an error in any of the following cases:</td>
</tr>
<tr>
<td></td>
<td>• When a specified table does not exist at the source database</td>
</tr>
<tr>
<td></td>
<td>• When the table_names parameter is set to NULL</td>
</tr>
<tr>
<td>source_directory_object</td>
<td>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remain in this directory after the procedure completes.</td>
</tr>
<tr>
<td></td>
<td>This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the source_directory_object parameter.</td>
</tr>
<tr>
<td></td>
<td>If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
</tbody>
</table>
Table 165-23  (Cont.) MAINTAIN_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_directory_object</td>
<td>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred. If the source database and destination database run on the same computer system, then the source and destination directories must be different. This parameter is ignored if instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. In this case, specify NULL for the destination_directory_object parameter. If NULL and instantiation is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE, then the procedure raises an error. Note: The specified directory object cannot point to an Oracle ASM disk group.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database. If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure uses the global name of the local database.</td>
</tr>
<tr>
<td>destination_database</td>
<td>The global name of the destination database. If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
### Table 165-23  (Cont.) MAINTAIN_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dump_file_name</td>
<td>The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the <em>source_directory_object</em> or <em>destination_directory_object</em> parameter, then the procedure raises an error. If <strong>NULL</strong> and <em>instantiation</em> is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE, then the export dump file name is generated by the system. In this case, the export dump file name is <code>expat.nn.dmp</code>, where <code>nn</code> is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</td>
</tr>
<tr>
<td>capture_queue_user</td>
<td>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the <strong>GRANT</strong> option. If <strong>NULL</strong>, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</td>
</tr>
<tr>
<td>log_file</td>
<td>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. This parameter is ignored if <em>instantiation</em> is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK or DBMS_STREAMS_ADM.INSTANTIATION_NONE. If <strong>NULL</strong> and <em>instantiation</em> is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE, then the log file name is the same name as the export dump file name with an extension of <code>.clg</code>.</td>
</tr>
</tbody>
</table>
### Table 165-23  (Cont.) MAINTAIN_TABLES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instantiation</td>
<td>Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation:</td>
</tr>
<tr>
<td></td>
<td>DBMS_STREAMS_ADM.INSTANTIATION_TABLE performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. If the instantiation parameter is set to this value, then the user who runs this procedure must have EXECUTE privilege on the DBMS_FILE_TRANSFER package.</td>
</tr>
<tr>
<td></td>
<td>DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. If the instantiation parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</td>
</tr>
<tr>
<td></td>
<td>DBMS_STREAMS_ADM.INSTANTIATION_NONE does not perform an instantiation. This setting is valid only if the perform_actions parameter is set to FALSE, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly.</td>
</tr>
<tr>
<td></td>
<td>If this parameter is set to DBMS_STREAMS_ADM.INSTANTIATION_TABLE or DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK, then the tables being instantiated must exist at the source database, and the tablespaces that contain the tables must exist at the destination database.</td>
</tr>
<tr>
<td></td>
<td>If an instantiated database object does not exist at the destination database, then it is imported at the destination database, including its supplemental logging specifications from the source database and its supporting database objects, such as indexes and triggers. However, if the database object exists at the destination database before instantiation, then it is not imported at the destination database. Therefore, the supplemental logging specifications from the source database are not specified for the database object at the destination database, and the supporting database objects are not imported.</td>
</tr>
<tr>
<td></td>
<td>Also, if an instantiated table does not exist at the destination database, then this procedure sets the instantiation SCN for the table. However, if an instantiated table exist at the destination database before instantiation, then this procedure does not set the instantiation SCN for the table. In this case, you must set the instantiation SCN for the table manually after the procedure completes.</td>
</tr>
</tbody>
</table>
165.6.24 MAINTAIN_TABLESPACES Procedure

This procedure clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases.

This procedure can either perform these actions directly, or it can generate a script that performs these actions. Run this procedure at the source database.

---

**Note:**

This procedure is deprecated. It is replaced by the `MAINTAIN_TTS` procedure.

---

**See Also:**

- "Deprecated Subprograms"
- `MAINTAIN_TTS` Procedure

---

**Syntax**

```sql
DBMS_STREAMS_ADM.MAINTAIN_TABLESPACES(
    tablespace_names             IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
    source_directory_object      IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    destination_database         IN VARCHAR2,
    setup_streams                IN BOOLEAN DEFAULT TRUE,
    script_name                  IN VARCHAR2 DEFAULT NULL,
    script_directory_object      IN VARCHAR2 DEFAULT NULL,
    dump_file_name               IN VARCHAR2 DEFAULT NULL,
    source_queue_table           IN VARCHAR2 DEFAULT 'streams_queue_table',
    source_queue_name            IN VARCHAR2 DEFAULT 'streams_queue',
    source_queue_user            IN VARCHAR2 DEFAULT NULL,
    destination_queue_table      IN VARCHAR2 DEFAULT 'streams_queue_table',
    destination_queue_name       IN VARCHAR2 DEFAULT 'streams_queue',
    destination_queue_user       IN VARCHAR2 DEFAULT NULL,
    capture_name                 IN VARCHAR2 DEFAULT 'capture',
    propagation_name             IN VARCHAR2 DEFAULT NULL,
    apply_name                    IN VARCHAR2 DEFAULT NULL,
    log_file                      IN VARCHAR2 DEFAULT NULL,
    bi_directional               IN BOOLEAN DEFAULT FALSE,
    include_ddl                   IN BOOLEAN DEFAULT FALSE);
```
### Parameters

**Table 165-24 MAINTAIN_TABLESPACES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `tablespace_names`         | The local tablespace set to be cloned at the destination database and maintained by Oracle Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group. If NULL, then the procedure raises an error.  
**See Also:** TABLESPACE_SET Table Type |
| `source_directory_object` | The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are placed. These files remain in this directory after the procedure completes. If NULL, then the procedure raises an error.  
**Note:** The specified directory object cannot point to an Oracle ASM disk group. |
| `destination_directory_object` | The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are transferred. If the source database and destination database run on the same computer system, then the source and destination directories must be different. If NULL, then the procedure raises an error.  
**Note:** The specified directory object cannot point to an Oracle ASM disk group. |
| `destination_database`    | The global name of the destination database. A database link from the source database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error. |
| `setup_streams`           | If TRUE, then the procedure performs the necessary actions to maintain the tablespaces directly. If FALSE, then the procedure does not perform the necessary actions to maintain the tablespaces directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:  
- `script_name`  
- `script_directory_object` |
### Table 165-24  (Cont.) MAINTAIN_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>If non-NULL and the <code>setup_streams</code> parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to maintain the specified tablespace set. If a file with the specified script name exists in the specified directory for the <code>script_directory_object</code> parameter, then the procedure appends the statements to the existing file. If non-NULL and the <code>setup_streams</code> parameter is TRUE, then this procedure generates the specified script and performs the actions to maintain the specified tablespace set directly. If <code>NULL</code> and the <code>setup_streams</code> parameter is TRUE, then this procedure does not generate a script and performs the actions to maintain the specified tablespace set directly. If <code>NULL</code> and the <code>setup_streams</code> parameter is FALSE, then the procedure raises an error.</td>
</tr>
<tr>
<td>script_directory_object</td>
<td>The directory object for the directory on the local computer system into which the generated script is placed. If the <code>script_name</code> parameter is <code>NULL</code>, then the procedure ignores this parameter and does not generate a script. If <code>NULL</code> and the <code>script_name</code> parameter is non-NULL, then the procedure raises an error. Note: The specified directory object cannot point to an Oracle ASM disk group.</td>
</tr>
<tr>
<td>dump_file_name</td>
<td>The name of the Data Pump export dump file that contains the specified tablespace set. If a file with the specified file name exists in the specified directory for the <code>source_directory_object</code> or <code>destination_directory_object</code> parameter, then the procedure raises an error. If <code>NULL</code>, then the export dump file name is generated by the system. In this case, the export dump file name is <code>expatnn.dmp</code>, where <code>nn</code> is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</td>
</tr>
<tr>
<td>source_queue_table</td>
<td>The name of the queue table for the queue at the source database, specified as <code>[schema_name.]queue_table_name</code>. For example, <code>strmadmin.streams_queue_table</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the queue at the source database that will function as the ANYDATA queue, specified as <code>[schema_name.]queue_name</code>. For example, <code>strmadmin.streams_queue</code>. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.</td>
</tr>
</tbody>
</table>
Table 165-24  (Cont.) MAINTAIN_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_queue_user</td>
<td>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</td>
</tr>
<tr>
<td>destination_queue_table</td>
<td>The name of the queue table for the queue at the destination database, specified as [schema_name.]queue_table_name. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the queue at the destination database that will function as the ANYDATA queue, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue. If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.</td>
</tr>
<tr>
<td>destination_queue_user</td>
<td>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</td>
</tr>
<tr>
<td>capture_name</td>
<td>The name of each capture process configured to capture changes to the database objects in the tablespace set. Do not specify an owner. If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database objects in the tablespace set to the positive capture process rule set. Note: The capture process name cannot be altered after the capture process is created.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>propagation_name</td>
<td>The name of each propagation configured to propagate changes to the database objects in the tablespace set. Do not specify an owner. If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the database objects in the tablespace set to the positive propagation rule set. If NULL, then the system generates a name for each propagation it creates. <strong>Note:</strong> The propagation name cannot be altered after the propagation is created.</td>
</tr>
<tr>
<td>apply_name</td>
<td>The name of each apply process configured to apply changes to the database objects in the tablespace set. Do not specify an owner. If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the database objects in the tablespace set to the positive apply process rule set. The specified name must not match the name of an existing messaging client at the destination database. If NULL, then the system generates a name for each apply process it creates. <strong>Note:</strong> The apply process name cannot be altered after the apply process is created.</td>
</tr>
<tr>
<td>log_file</td>
<td>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. If NULL, then the log file name is the same name as the export dump file name with an extension of .clg.</td>
</tr>
<tr>
<td>bi_directional</td>
<td>Specify <strong>TRUE</strong> to configure bi-directional replication between the current database and the database specified in destination_database. Both databases are configured as source and destination databases, a capture and apply process is configured at both databases, and propagations are configured between the databases to propagate messages. Specify <strong>FALSE</strong> to configure one way replication from the current database to the database specified in destination_database. A capture process is configured at the current database, a propagation is configured to propagate messages from the current database to the destination database, and an apply process is configured at the destination database.</td>
</tr>
</tbody>
</table>
Table 165-24  (Cont.) MAINTAIN_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_ddl</td>
<td>Specify TRUE to configure an Oracle Streams replication environment that maintains both DML and DDL changes. Specify FALSE to configure an Oracle Streams replication environment that maintains DML changes only. When this parameter is set to FALSE, DDL changes, such as ALTER TABLE, will not be replicated.</td>
</tr>
</tbody>
</table>

Usage Notes

The specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

See Also:

Oracle Database Administrator's Guide for more information about self-contained tablespace sets

Additional Documentation for this Procedure

The following documentation applies to the MAINTAIN_TABLESPACES procedure:

- "* Procedures That Configure an Oracle Streams Environment " for information about automatic platform conversion
- Oracle Streams Replication Administrator's Guide

Requirements for Running this Procedure

Meet the following requirements when run the MAINTAIN_TABLESPACES procedure:

- Run the procedure at the source database.
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the destination_database parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- The user who runs this procedure should be granted the DBA role. This user must have the necessary privileges to complete the following actions:
  - Create ANYDATA queues, capture processes, propagations, and apply processes.
  - Specify supplemental logging
  - Run subprograms in the DBMS_STREAMS_ADM and DBMS_AQADM packages.
  - Access the database specified in the destination_database parameter through a database link. This database link should have the same name as the global name of the destination database.
  - Run subprograms in the DBMS_STREAMS_TABLESPACES_ADM package
- The necessary privileges to run the `CLONE_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACES_ADM` package at the source database. See `CLONE_TABLESPACES Procedure` for the list of required privileges.

- The necessary privileges to run the `ATTACH_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACES_ADM` package at the destination database. See `ATTACH_TABLESPACES Procedure` for the list of required privileges.

To ensure that the user who runs this procedure has the necessary privileges, you should configure an Oracle Streams administrator at each database, and each database link should be created in the Oracle Streams administrator's schema.

- If the `bi_directional` parameter is set to `TRUE`, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.

- Each specified directory object must be created using the SQL statement `CREATE DIRECTORY`, and the user who invokes this procedure must have `READ` and `WRITE` privilege on each one.

- The databases configured by this procedure must be Oracle Database 10g Release 2 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10g Release 2 or later database.
  - The `setup_streams` parameter is set to `TRUE` to configure the Oracle Streams replication environment directly.

- The databases configured by this procedure must be Oracle Database 10g Release 1 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10g Release 2 or later database.
  - The `setup_streams` parameter is set to `FALSE` in this procedure, and the replication environment is configured with a generated script.

If the script configures an Oracle Database 10g Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10g Release 2 or later, such as queue-to-queue propagation.

- If the procedure is run at an Oracle Database 10g Release 1 database, then the databases configured by the procedure must be Oracle Database 10g Release 1 or later databases.

### See Also:

*Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

---

**Configuration Progress and Recoverability**

When this procedure is run with the `setup_streams` parameter set to `TRUE`, metadata about its configuration actions is recorded in the following data dictionary views:
If the procedure stops because it encounters an error, then you can use the RECOVER_OPERATION procedure to complete the configuration after you correct the conditions that caused the error.

Note: When this procedure is run with the setup_streams parameter set to FALSE, these views are not populated. Also, the views are not populated when a script generated by this procedure is run.

See Also:

"RECOVER_OPERATION Procedure"

165.6.25 MAINTAIN_TTS Procedure

This procedure clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases.

This procedure can either perform these actions directly, or it can generate a script that performs these actions.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

Note:

- This procedure automatically excludes database objects that are not supported by Oracle Streams in the tablespaces from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the DBA_STREAMS_UNSUPPORTED data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
- This procedure replaces the deprecated MAINTAIN_TABLESPACES procedure.

See Also:

"Procedures That Configure an Oracle Streams Environment" for more information about this procedure
Syntax

DBMS STREAMS ADM.MAINTAIN_TTS(
    tablespace_names            IN DBMS STREAMS TABLESPACE ADM.TABLESPACE_SET,
    source_directory_object     IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database             IN VARCHAR2,
    destination_database        IN VARCHAR2,
    perform_actions             IN BOOLEAN DEFAULT TRUE,
    script_name                 IN VARCHAR2 DEFAULT NULL,
    script_directory_object     IN VARCHAR2 DEFAULT NULL,
    dump_file_name              IN VARCHAR2 DEFAULT NULL,
    capture_name                IN VARCHAR2 DEFAULT NULL,
    capture_queue_table         IN VARCHAR2 DEFAULT NULL,
    capture_queue_name          IN VARCHAR2 DEFAULT NULL,
    capture_queue_user          IN VARCHAR2 DEFAULT NULL,
    propagation_name            IN VARCHAR2 DEFAULT NULL,
    apply_name                   IN VARCHAR2 DEFAULT NULL,
    apply_queue_table           IN VARCHAR2 DEFAULT NULL,
    apply_queue_name            IN VARCHAR2 DEFAULT NULL,
    apply_queue_user            IN VARCHAR2 DEFAULT NULL,
    log_file                     IN VARCHAR2 DEFAULT NULL,
    bi_directional              IN BOOLEAN DEFAULT FALSE,
    include_ddl                 IN BOOLEAN DEFAULT FALSE);

Parameters

See Also:

"Procedures That Configure an Oracle Streams Environment“ for
descriptions of the common procedure parameters that are not in
Table 165-25

Table 165-25  MAINTAIN_TTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_names</td>
<td>The local tablespace set to be cloned at the destination database and maintained by Oracle Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have read privilege on these directory objects. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group. If NULL, then the procedure raises an error. See Also: TABLESPACE_SET Table Type</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| source_directory_object       | The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are placed. These files remain in this directory after the procedure completes.  
|                               | If NULL, then the procedure raises an error.  
|                               | **Note:** The specified directory object cannot point to an Oracle ASM disk group.                                                                                                                                 |
| destination_directory_object  | The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are transferred.  
|                               | If the source database and destination database run on the same computer system, then the source and destination directories must be different.  
|                               | If NULL, then the procedure raises an error.  
|                               | **Note:** The specified directory object cannot point to an Oracle ASM disk group.                                                                                                                                 |
| source_database               | The global name of the source database.  
|                               | If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.  
|                               | If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.  
|                               | If NULL, then the procedure uses the global name of the local database.                                                                                                                                 |
| destination_database          | The global name of the destination database.  
|                               | If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.  
|                               | If NULL, then the procedure raises an error.                                                                                                                                                           |
### Table 165-25   (Cont.) MAINTAIN_TTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dump_file_name</td>
<td>The name of the Data Pump export dump file that contains the specified tablespace set. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error. If NULL, then the export dump file name is generated by the system. In this case, the export dump file name is expat.nn.dmp, where nn is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</td>
</tr>
<tr>
<td>log_file</td>
<td>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file. If NULL, then the log file name is the same name as the export dump file name with an extension of .clg.</td>
</tr>
</tbody>
</table>

### Usage Notes

The specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

### See Also:

*Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

### Additional Privileges Required by the MAINTAIN_TTS Procedure

In addition to the required privileges described in "* Procedures That Configure an Oracle Streams Environment", the user who runs the MAINTAIN_TTS procedure must have the necessary privileges to complete the following actions:

- Run subprograms in the `DBMS_STREAMS_TABLESPACES_ADM` package
- The necessary privileges to run the `CLONE_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACES_ADM` package at the source database. See `CLONE_TABLESPACES Procedure` for the list of required privileges.
- The necessary privileges to run the `ATTACH_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACES_ADM` package at the destination database. See `ATTACH_TABLESPACES Procedure` for the list of required privileges.
165.6.26 MERGE_STREAMS Procedure

This procedure merges a stream that is flowing from one capture process with a stream that is flowing from another capture process.

Typically, this procedure is used to merge two streams that were split using the SPLIT_STREAMS procedure in this package. The SPLIT_STREAMS procedure clones components of the original stream when it splits the streams. Therefore, the information in this section uses the following terminology:

- The stream before it was split off has the original queue, original capture process, and original propagation.
- The stream that was split off by the SPLIT_STREAMS procedure has a cloned queue, cloned capture process, and cloned propagation.

This procedure is called by the MERGE_STREAMS_JOB procedure. The MERGE_STREAMS_JOB procedure determines whether the streams are within a user-specified merge threshold so that the streams can be merged safely. If the streams are not within the merge threshold, then the MERGE_STREAMS_JOB procedure does nothing. Typically, it is best to run the MERGE_STREAMS_JOB procedure instead of running the MERGE_STREAMS procedure directly.

However, you can choose to run the MERGE_STREAMS procedure directly when the following conditions are met:

- The problem at the destination of the split stream has been corrected, and the destination queue can accept changes.
- The cloned capture process used by the split stream is started and is capturing changes.
- The apply process at the destination database is applying the changes captured by the cloned capture process.
- The CAPTURE_MESSAGE_CREATE_TIME in the GV$STREAMS_CAPTURE view of the cloned capture process has caught up to, or nearly caught up to, the CAPTURE_MESSAGE_CREATE_TIME of the original capture process. The cloned capture process might never completely catch up to the original capture process. Therefore, you can merge the split stream when the cloned capture process has nearly caught up to the original capture process.

The MERGE_STREAMS procedure performs the following actions:

1. Stops the cloned capture process.
2. Stops the original capture process.
3. Copies the cloned propagation back to the original propagation. The propagation has the same name as the original propagation after it is copied back.
4. Starts the original capture process from the lower SCN value of these two SCN values:
   - The acknowledged SCN of the cloned propagation.
   - The lowest acknowledged SCN of the other propagations that propagate changes captured by the original capture process.

When the original capture process is started, it might recapture changes that it already captured, or it might capture changes that were already captured by the cloned capture process. In either case, the relevant apply processes will discard any duplicate changes they receive.
5. Drops the cloned propagation.
6. Drops the cloned capture process.
7. Drops the cloned queue.

See Also:

- MERGE_STREAMS_JOB Procedure
- SPLIT_STREAMS Procedure

Syntax

```sql
DBMS_STREAMS_ADM.MERGE_STREAMS(
    cloned_propagation_name  IN  VARCHAR2,
    propagation_name         IN  VARCHAR2  DEFAULT NULL,
    queue_name               IN  VARCHAR2  DEFAULT NULL,
    perform_actions          IN  BOOLEAN   DEFAULT TRUE,
    script_name              IN  VARCHAR2  DEFAULT NULL,
    script_directory_object  IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 165-26  MERGE_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloned_propagation_name</td>
<td>The name of the cloned propagation used by the stream that was split off from the original stream using the SPLIT_STREAMS procedure. The name of the cloned propagation also identifies the cloned queue and capture process used by the cloned propagation. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
<tr>
<td>propagation_name</td>
<td>The name of the propagation that is merged back to the original stream. If NULL, then the name of the original propagation in the original stream is used. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value to use a name that is different from the original propagation name or if you are merging two streams that were not split by the SPLIT_STREAMS procedure. See &quot;Usage Notes&quot; for more information. If a non-NULL value is specified, then an error is raised under either of the following conditions:</td>
</tr>
<tr>
<td></td>
<td>• The queue specified in the queue_name parameter does not exist.</td>
</tr>
<tr>
<td></td>
<td>• The queue specified in the queue_name parameter exists but is not used by a capture process.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>queue_name</td>
<td>The name of the queue that is the source queue for the propagation that is merged back. If NULL, then the existing, original queue is the source queue for the propagation that is merged back. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value if you are merging two streams that were not split by the SPLIT_STREAMS procedure. Specify the name of the existing queue used by the capture process that will capture changes in the merged stream. See &quot;Usage Notes&quot; for more information.</td>
</tr>
</tbody>
</table>
| perform_actions | If TRUE, then the procedure performs the necessary actions to merge the streams directly. If FALSE, then the procedure does not perform the necessary actions to merge the streams directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:  
- script_name  
- script_directory_object  |
| script_name   | If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to merge the streams. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file. If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to split the stream directly. If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to merge the streams directly and does not generate a script. If NULL and the perform_actions parameter is FALSE, then the procedure raises an error. |
| script_directory_object | The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error. Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group. |

Usage Notes

You can use the MERGE_STREAMS procedure to merge two streams that were not split using the SPLIT_STREAMS procedure. Merging streams in this way can save resources and improve performance when a single database is running two or more capture processes.

The DBA_STREAMS_SPLIT_MERGE view contains information about split and merge operations.
165.6.27 MERGE_STREAMS_JOB Procedure

This procedure determines whether the original capture process and the cloned capture process are within the specified merge threshold. If they are within the merge threshold, then this procedure runs the MERGE_STREAMS procedure to merge the two streams.

Typically, this procedure is used to merge two streams that were split using the SPLIT_STREAMS procedure in this package. The SPLIT_STREAMS procedure clones components of the original stream when it splits the streams. Therefore, the information in this section uses the following terminology:

- The stream before it was split off has the original queue, original capture process, and original propagation.
- The stream that was split off by the SPLIT_STREAMS procedure has a cloned queue, cloned capture process, and cloned propagation.

If the auto_merge_threshold parameter was set to a positive number in the SPLIT_STREAMS procedure that split the streams, then a merge job runs the MERGE_STREAMS_JOB procedure automatically according to its schedule. The schedule name is specified for the schedule_name parameter, and the merge job name is specified for the merge_job_name parameter when the MERGE_STREAMS_JOB procedure is run automatically. The merge job and its schedule were created by the SPLIT_STREAMS procedure.

If the auto_merge_threshold parameter was set to NULL or 0 (zero) in the SPLIT_STREAMS procedure that split the streams, then you can run the MERGE_STREAMS_JOB procedure manually. In this case, it is not run automatically.

See Also:

- MERGE_STREAMS Procedure
- SPLIT_STREAMS Procedure
- Oracle Streams Replication Administrator's Guide for instructions on using the MERGE_STREAMS_JOB procedure

Syntax

```
DBMS_STREAMS_ADM.MERGE_STREAMS_JOB(
    cloned_propagation_name        IN VARCHAR2,
    propagation_name               IN VARCHAR2 DEFAULT NULL,
    queue_name                     IN VARCHAR2 DEFAULT NULL,
    merge_threshold                IN NUMBER,
    schedule_name                  IN VARCHAR2 DEFAULT NULL,
    merge_job_name                 IN VARCHAR2 DEFAULT NULL);
```
### Parameters

**Table 165-27  MERGE_STREAMS_JOB Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloned_propagation_name</td>
<td>The name of the cloned propagation used by the stream that was split off from the original stream using the SPLIT_STREAMS procedure. The name of the cloned propagation also identifies the cloned queue and capture process used by the cloned propagation. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
</tbody>
</table>
| propagation_name   | The name of the propagation that is merged back to the original stream. If NULL, then the name of the original propagation in the original stream is used. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value to use a name that is different from the original propagation name or if you are merging two streams that were not split by the SPLIT_STREAMS procedure. See "Usage Notes" for more information. If a non-NULL value is specified, then an error is raised under either of the following conditions:  
  - The queue specified in the queue_name parameter does not exist.  
  - The queue specified in the queue_name parameter exists but is not used by a capture process. |
| queue_name         | The name of the queue that is the source queue for the propagation that is merged back. If NULL, then the existing, original queue is the source queue for the propagation that is merged back. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value if you are merging two streams that were not split by the SPLIT_STREAMS procedure. Specify the name of the existing queue used by the capture process that will capture changes in the merged stream. See "Usage Notes" for more information. |
| merge_threshold    | The merge threshold in seconds. The value of the CAPTURE_MESSAGE_CREATE_TIME column for each capture process in the GV$STREAMS_CAPTURE dynamic performance view determines whether the streams are merged. Specifically, if the difference, in seconds, between the CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process and the original capture process is less than or equal to the value specified for this parameter, then this procedure runs the MERGE_STREAMS procedure to merge the streams. If the difference is greater than the value specified by this parameter, then this procedure does nothing. |
| schedule_name      | The name of the schedule for the merge job. If NULL, then no schedule name is specified. Typically, you set this parameter to NULL when the auto_merge_threshold parameter was set to NULL or 0 (zero) in the SPLIT_STREAMS procedure that split the streams. Specify NULL if you run this procedure manually. |
Table 165-27  (Cont.) MERGE_STREAMS_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge_job_name</td>
<td>The name of the job that merges the streams. If NULL, then no merge job name is specified. Typically, you set this parameter to NULL when the auto_merge_threshold parameter was set to NULL or 0 (zero) in the SPLIT_STREAMS procedure that split the streams. Specify NULL if you run this procedure manually.</td>
</tr>
</tbody>
</table>

**Usage Notes**

You can use the MERGE_STREAMS_JOB procedure to merge two streams that were not split using the SPLIT_STREAMS procedure. Merging streams in this way can save resources and improve performance when a single database is running two or more capture processes.

After the MERGE_STREAMS_JOB procedure completes, you can query the DBA_CAPTURE and DBA_PROPAGATION views to determine whether the streams were merged. If the streams were merged, then the cloned capture process and cloned propagation do not appear in these views.

If the streams were merged and the schedule_name and merge_job_name parameters were non-NULL, then the specified schedule and merge job are deleted automatically.

The DBA_STREAMS_SPLIT_MERGE view contains information about split and merge operations.

**165.6.28 POST_INSTANTIATION_SETUP Procedure**

This procedure performs the actions required after instantiation to configure an Oracle Streams replication environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

To complete the Oracle Streams replication configuration, follow these steps:

1. Run the PRE_INSTANTIATION_SETUP procedure at the source database.
2. Perform any necessary instantiation actions.
3. Run the POST_INSTANTIATION_SETUP procedure at the source database.

Typically, the Oracle Streams replication environment configured using these steps serves one of the following purposes:

- Replicates changes to shared database objects to keep the database objects synchronized at different databases.
- Replicates changes to database objects during a database maintenance operation, such migrating a database to a different platform. In this case, use the CLEANUP_INSTANTIATION_SETUP procedure to remove the replication environment after the maintenance operation is complete.
When the `POST_INSTANTIATION_SETUP` procedure is run, the parameter values must match the parameter values specified when the corresponding `PRE_INSTANTIATION_SETUP` procedure was run, except for the values of the following parameters: `perform_actions`, `script_name`, `script_directory_object`, and `start_processes`.

A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. This procedure does not configure replication for these schemas.

See Also:

- "PRE_INSTANTIATION_SETUP Procedure"
- "CLEANUP_INSTANTIATION_SETUP Procedure"
- "Procedures That Configure an Oracle Streams Environment" for more information about this procedure
- Oracle Streams Replication Administrator's Guide for information about setting up an Oracle Streams replication environment
- Oracle Streams Concepts and Administration for information about completing database maintenance operations

Syntax

```
DBMS_STREAMS_ADM.POST_INSTANTIATION_SETUP(
    maintain_mode   IN VARCHAR2,
    tablespace_names IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
    source_database  IN VARCHAR2,
    destination_database IN VARCHAR2,
    perform_actions IN BOOLEAN DEFAULT TRUE,
    script_name     IN VARCHAR2 DEFAULT NULL,
    script_directory_object IN VARCHAR2 DEFAULT NULL,
    capture_name    IN VARCHAR2 DEFAULT NULL,
    capture_queue_table IN VARCHAR2 DEFAULT NULL,
    capture_queue_name IN VARCHAR2 DEFAULT NULL,
    capture_queue_user IN VARCHAR2 DEFAULT NULL,
    propagation_name IN VARCHAR2 DEFAULT NULL,
    apply_name      IN VARCHAR2 DEFAULT NULL,
    apply_queue_table IN VARCHAR2 DEFAULT NULL,
    apply_queue_name IN VARCHAR2 DEFAULT NULL,
    apply_queue_user IN VARCHAR2 DEFAULT NULL,
    bi_directional  IN BOOLEAN DEFAULT FALSE,
    include_ddl     IN BOOLEAN DEFAULT FALSE,
    start_processes IN BOOLEAN DEFAULT FALSE,
    instantiation_scn IN NUMBER DEFAULT NULL,

```
exclude_schemas IN VARCHAR2 DEFAULT NULL,
exclude_flags IN BINARY_INTEGER DEFAULT NULL);

Parameters

See Also:
"Common Parameters for the Configuration Procedures" for descriptions of
the procedure parameters that are not in Table 165-28

Table 165-28 POST_INSTANTIATION_SETUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maintain_mode</td>
<td>Specify one of the following:</td>
</tr>
<tr>
<td></td>
<td>- GLOBAL to maintain the entire database by configuring replication between</td>
</tr>
<tr>
<td></td>
<td>the local database and the database specified in the destination_database</td>
</tr>
<tr>
<td></td>
<td>parameter</td>
</tr>
<tr>
<td></td>
<td>- TRANSPORTABLE TABLESPACES to maintain a set of tablespaces by configuring</td>
</tr>
<tr>
<td></td>
<td>replication between the local database and the database specified in the</td>
</tr>
<tr>
<td></td>
<td>destination_database parameter</td>
</tr>
<tr>
<td>tablespace_names</td>
<td>If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the</td>
</tr>
<tr>
<td></td>
<td>local tablespace set to be cloned at the destination database and</td>
</tr>
<tr>
<td></td>
<td>maintained by Oracle Streams. The tablespaces in the tablespace set</td>
</tr>
<tr>
<td></td>
<td>must exist at the source database, but these tablespaces must not exist</td>
</tr>
<tr>
<td></td>
<td>at the destination database. Also, a directory object must exist for each</td>
</tr>
<tr>
<td></td>
<td>directory that contains the datafiles for the tablespace set. The user</td>
</tr>
<tr>
<td></td>
<td>who invokes this procedure must have READ privilege on these directory</td>
</tr>
<tr>
<td></td>
<td>objects.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database.</td>
</tr>
<tr>
<td></td>
<td>If the specified global name is the same as the global name of the</td>
</tr>
<tr>
<td></td>
<td>local database, then the procedure configures a local capture process for</td>
</tr>
<tr>
<td></td>
<td>the source database.</td>
</tr>
<tr>
<td></td>
<td>If the specified global name is different from the global name of the</td>
</tr>
<tr>
<td></td>
<td>local database, then the procedure configures a downstream capture</td>
</tr>
<tr>
<td></td>
<td>process at the local database. In this case, a database link from the</td>
</tr>
<tr>
<td></td>
<td>local database to the source database with the same name as the global</td>
</tr>
<tr>
<td></td>
<td>name of the source database must exist and must be accessible to the</td>
</tr>
<tr>
<td></td>
<td>user who runs the procedure.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure uses the global name of the local database.</td>
</tr>
</tbody>
</table>
Table 165-28  (Cont.) POST_INSTANTIATION_SETUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_database</td>
<td>The global name of the destination database. If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>start_processes</td>
<td>If TRUE, then the procedure starts each capture process and apply process. Any disabled capture or apply process created by the PRE_INSTANTIATION_SETUP procedure also is started. If FALSE, then the procedure does not start any capture processes or apply processes.</td>
</tr>
<tr>
<td>instantiation_scn</td>
<td>Specify the instantiation SCN for the database objects at the destination database if the instantiation SCN was not set during instantiation. The instantiation SCN is not set automatically during RMAN instantiations, but the correct instantiation SCN value should be determined during an RMAN instantiation. See the Oracle Streams Replication Administrator's Guide for more information. Specify NULL if the instantiation SCN was set for the database objects at the destination database during instantiation. The instantiation SCN can be set during export/import instantiations.</td>
</tr>
<tr>
<td>exclude_schemas</td>
<td>A comma-delimited list of schemas to exclude from the Oracle Streams configuration. Schema rules are added to the negative rule sets of each capture process to exclude these schemas. Specify an asterisk (*) to exclude all of the schemas in the database. If NULL, then the procedure does not exclude any schemas in the database. This parameter is valid only if the MAINTAIN_MODE parameter is set to GLOBAL. If the MAINTAIN_MODE parameter is set to TRANSPORTABLE TABLESPACES, then the procedure ignores this parameter.</td>
</tr>
<tr>
<td>exclude_flags</td>
<td>Specify what is excluded from the replication configuration in the schemas specified by the exclude_schemas parameter. This parameter works the same way in the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures. See &quot;Usage Notes&quot; for the PRE_INSTANTIATION_SETUP procedure for more information.</td>
</tr>
</tbody>
</table>

Usage Notes

The following sections contain usage notes for this procedure.

Self-Contained Tablespace Sets
If the `maintain_mode` parameter is set to `TRANSPORTABLE TABLESPACES`, then the specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

**See Also:**

*Oracle Database Administrator’s Guide* for more information about self-contained tablespace sets

**Destination Database Renamed During RMAN Database Instantiation**

If the `maintain_mode` parameter is set to `GLOBAL`, then database instantiation is required before running the `POST_INSTANTIATION_SETUP` procedure. If the RMAN `DUPLICATE` or RMAN `CONVERT DATABASE` command is used for database instantiation, then the global name of the destination database can be renamed to the global name of the source database during instantiation. In this case, before you run the `POST_INSTANTIATION_SETUP` procedure, complete the following steps:

1. Rename the global name of the destination database back to the name specified in the `destination_database` parameter.

2. At the destination database, drop and re-create any loopback database links that existed on the source and were cloned on the destination database. For example, suppose the source database `dbs1.net` has a database link that refers to itself. Suppose the destination database is `dbs2.net`. At the destination database, drop and re-create this database link as a loopback database link that refers to itself (`dbs2.net`).

3. At the destination database, drop any database links that were cloned from the source database and are from the source database to the destination database. For example, if the source database is `dbs1.net` and the destination database is `dbs2.net`, then drop any database links on the destination database that are from `dbs1.net` to `dbs2.net`.

4. Create a database link from the destination database to the source database with the same name as the global name of the source database. The database link must be accessible to the Oracle Streams administrator at the destination database.

   This database link is required because the `POST_INSTANTIATION_SETUP` procedure runs the `SET_GLOBAL_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package at the destination database, and the `SET_GLOBAL_INSTANTIATION_SCN` procedure requires the database link. The instantiation SCN is set to the value specified in the `instantiation_scn` parameter of the `POST_INSTANTIATION_SETUP` procedure.
Note:

When the RMAN DUPLICATE or CONVERT DATABASE command is used for database instantiation, the destination database cannot be the capture database.

Oracle Streams Components Removed From the Destination Database

If the maintain_mode parameter is set to GLOBAL, then database instantiation is required before running the POST_INSTANTIATION_SETUP procedure. During database instantiation, Oracle Streams components created by the PRE_INSTANTIATION_SETUP procedure, such as Oracle Streams clients and queues, can be copied from the source database to the destination database. The POST_INSTANTIATION_SETUP procedure removes the Stream components created by the PRE_INSTANTIATION_SETUP procedure from the destination database.

In some cases, rule sets and rules created by the PRE_INSTANTIATION_SETUP procedure might not be removed from the destination database. The POST_INSTANTIATION_SETUP procedure does not associate these rule sets and rules with any Stream clients in the destination database. Optionally, you can remove these rule sets and rules from the destination database after the POST_INSTANTIATION_SETUP procedure, or the script generated by the procedure, completes.

Note:

The POST_INSTANTIATION_SETUP procedure only removes Oracle Streams components that were created by the PRE_INSTANTIATION_SETUP procedure. It does not remove Oracle Streams components that were created in a different way.

165.6.29 PRE_INSTANTIATION_SETUP Procedure

This procedure performs the actions required before instantiation to configure an Oracle Streams replication environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

To complete the Oracle Streams replication configuration, follow these steps:

1. Run the PRE_INSTANTIATION_SETUP procedure at the database that will be the source database in the Stream replication environment.
2. Perform any necessary instantiation actions.
3. Run the POST_INSTANTIATION_SETUP procedure at the source database.

Typically, the Oracle Streams replication environment configured using these steps serves one of the following purposes:

- Replicates changes to shared database objects to keep the database objects synchronized at different databases.
- Replicates changes to database objects during a database maintenance operation, such as migrating a database to a different platform. In this case, use the
CLEANUP_INSTANTIATION_SETUP procedure to remove the replication environment after the maintenance operation is complete.

**Note:**

- A capture process never captures changes in the SYS, SYSTEM, or CTXSYS schemas. This procedure does not configure replication for these schemas.
- When the RMAN DUPLICATE or CONVERT DATABASE command is used for database instantiation, the destination database cannot be the capture database.

**See Also:**

- "POST_INSTANTIATION_SETUP Procedure"
- "CLEANUP_INSTANTIATION_SETUP Procedure"
- "Procedures That Configure an Oracle Streams Environment" for more information about this procedure
- Oracle Streams Replication Administrator's Guide for information about setting up an Oracle Streams replication environment
- Oracle Streams Concepts and Administration for information about completing database maintenance operations

**Syntax**

```sql
DBMS_STREAMS_ADM.PRE_INSTANTIATION_SETUP(
    maintain_mode           IN VARCHAR2,
    tablespace_names        IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
    source_database         IN VARCHAR2,
    destination_database    IN VARCHAR2,
    perform_actions         IN BOOLEAN         DEFAULT TRUE,
    script_name             IN VARCHAR2        DEFAULT NULL,
    script_directory_object IN VARCHAR2        DEFAULT NULL,
    capture_name            IN VARCHAR2        DEFAULT NULL,
    capture_queue_table     IN VARCHAR2        DEFAULT NULL,
    capture_queue_name      IN VARCHAR2        DEFAULT NULL,
    capture_queue_user      IN VARCHAR2        DEFAULT NULL,
    propagation_name        IN VARCHAR2        DEFAULT NULL,
    apply_name              IN VARCHAR2        DEFAULT NULL,
    apply_queue_table       IN VARCHAR2        DEFAULT NULL,
    apply_queue_name        IN VARCHAR2        DEFAULT NULL,
    apply_queue_user        IN VARCHAR2        DEFAULT NULL,
    bi_directional          IN BOOLEAN         DEFAULT FALSE,
    include_ddl             IN BOOLEAN         DEFAULT FALSE,
    start_processes         IN BOOLEAN         DEFAULT FALSE,
    exclude_schemas         IN VARCHAR2        DEFAULT NULL,
    exclude_flags           IN BINARY_INTEGER  DEFAULT NULL);
```
Parameters

See Also:

"Common Parameters for the Configuration Procedures" for descriptions of the procedure parameters that are not in Table 165-29

Table 165-29  PRE_INSTANTIATION_SETUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maintain_mode</td>
<td>Specify one of the following:</td>
</tr>
<tr>
<td></td>
<td>• GLOBAL to maintain the entire database by configuring replication between the local database and the database specified in the destination_database parameter</td>
</tr>
<tr>
<td></td>
<td>• TRANSPORTABLE TABLESPACES to maintain a set of tablespaces by configuring replication between the local database and the database specified in the destination_database parameter</td>
</tr>
<tr>
<td>tablespace_names</td>
<td>If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the local tablespace set to be cloned at the destination database and maintained by Oracle Streams. The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database. Also, a directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects. If maintain_mode is set to GLOBAL, then specify an empty tablespace set. Regardless of the maintain_mode setting, an error is raised if the tablespace_names parameter is not set or is set to NULL.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database.</td>
</tr>
<tr>
<td></td>
<td>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database. If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure uses the global name of the local database.</td>
</tr>
</tbody>
</table>
Table 165-29  (Cont.) PRE_INSTANTIATION_SETUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_database</td>
<td>The global name of the destination database. If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>capture_queue_table</td>
<td>The name of the queue table for each queue used by a capture process, specified as [schema_name.]queue_table_name. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If NULL, then the system generates a name for the queue table of each queue used by a capture process, and the current user is the owner of each queue table.</td>
</tr>
<tr>
<td>start_processes</td>
<td>If TRUE, then the procedure starts each capture process and apply process. If FALSE, then the procedure does not start any capture processes or apply processes.</td>
</tr>
<tr>
<td>exclude_schemas</td>
<td>A comma-delimited list of schemas to exclude from the Oracle Streams configuration. Schema rules are added to the negative rule sets of each capture process to exclude these schemas. Specify an asterisk (*) to exclude all of the schemas in the database. If NULL, then the procedure does not exclude any schemas in the database. This parameter is valid only if the MAINTAIN_MODE parameter is set to GLOBAL. If the MAINTAIN_MODE parameter is set to TRANSPORTABLE TABLESPACES, then the procedure ignores this parameter.</td>
</tr>
<tr>
<td>exclude_flags</td>
<td>Specify what to exclude from the replication configuration in the schemas specified by the exclude_schemas parameter. See &quot;Usage Notes&quot; for more information.</td>
</tr>
</tbody>
</table>

Usage Notes

The following sections contain usage notes for this procedure.

Self-Contained Tablespace Sets

If the maintain_mode parameter is set to TRANSPORTABLE TABLESPACES, then the specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.
The exclude_flags Parameter

Specify one of the following values:

- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL` to exclude changes to the schemas and all of the database objects in the schemas
- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED` to exclude changes to the database objects that are not supported by Oracle Streams in the schemas

If both of these values are specified, then the procedure raises an error.

In addition to `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL` or `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED`, specify one or both of the following values:

- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML` to exclude data manipulation language (DML) changes made to the excluded database objects
- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL` to exclude data definition language (DDL) changes made to the excluded database objects

Use the plus sign (+) to specify more than one of these values. For example, to maintain DML changes to the tables in a schemas specified by the exclude_schemas parameter but exclude DDL changes to these schemas and the database objects in these schemas, specify the following for this parameter:

```
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL + DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL
```

To exclude DML and DDL changes made to unsupported database objects in the schemas specified by the exclude_schemas parameter, specify the following for this parameter:

```
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED + DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML + DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL
```

Rules for the excluded database objects are added to the negative rule set of each capture process. Therefore, changes to the excluded database objects will not be captured and replicated.

This parameter is valid only if the maintain_mode parameter is set to `GLOBAL` and the exclude_schemas parameter is set to a non-NULL value. If the maintain_mode parameter is set to `GLOBAL` and the exclude_schemas parameter is set to `NULL`, then the procedure ignores this parameter. If the maintain_mode parameter is set to `TRANSPORTABLE TABLESPACES`, then this the procedure ignores this parameter and excludes any database objects in the specified tablespace set that are not supported by Oracle Streams from the Oracle Streams configuration automatically.

Also, if schemas are specified in the exclude_schemas parameter, but the exclude_flags parameter is set to `NULL`, then the procedure does not add any rules to the negative rule set of any capture process, and the procedure includes the schemas specified in the exclude_schemas parameter in the replication environment.
165.6.30 PURGE_SOURCE_CATALOG Procedure

This procedure removes all Oracle Streams data dictionary information at the local database for the specified object.

You can use this procedure to remove Oracle Streams metadata that is not needed currently and will not be needed in the future.

Syntax

```sql
DBMS_STREAMS_ADM.PURGE_SOURCE_CATALOG(
    source_database     IN  VARCHAR2,
    source_object_name  IN  VARCHAR2,
    source_object_type  IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database containing the object. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>source_object_name</td>
<td>The name of the object specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>source_object_type</td>
<td>Type of the object. Currently, TABLE is the only possible object type.</td>
</tr>
</tbody>
</table>

Usage Notes

The global name of the source database containing the object must be specified for the source_database parameter. If the current database is not the source database for the object, then the procedure removes data dictionary information about the object from the current database, not the source database.

For example, suppose changes to the hr.employees table at the dbs1.net source database are being applied to the hr.employees table at the dbs2.net destination database. Also, suppose hr.employees at dbs2.net is not a source at all. In this case, specifying dbs2.net as the source_database for this table results in an error. However, specifying dbs1.net as the source_database for this table while running the PURGE_SOURCE_CATALOG procedure at the dbs2.net database removes data dictionary information about the table at dbs2.net.

Do not run this procedure at a database if either of the following conditions are true:

- Logical change records (LCRs) captured by the capture process for the object are or might be applied locally without reinstantiating the object.
- LCRs captured by the capture process for the object are or might be forwarded by the database without reinstantiating the object.
165.6.31 RECOVER_OPERATION Procedure

This procedure provides options for operations that stopped because they encountered an error. These operations include split and merge operations, Oracle Streams replication configuration operations, and Oracle Streams change table configuration operations. This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation.

This procedure only can perform these actions for the following operations:

- Split and merge operations using:
  - The `split_threshold` and `merge_threshold` capture process parameters set to non-NULL values to enable automatic split and merge
  - SPLIT_STREAMS Procedure
  - MERGE_STREAMS_JOB Procedure

- Change table configuration operations performed by the MAINTAIN_CHANGE_TABLE Procedure

- Replication configuration operations performed by the following procedures:
  - MAINTAIN_GLOBAL Procedure
  - MAINTAIN_SCHEMAS Procedure
  - MAINTAIN_SIMPLE_TABLESPACE Procedure
  - MAINTAIN_SIMPLE_TTS Procedure
  - MAINTAIN_TABLES Procedure
  - MAINTAIN_TABLESPACES Procedure
  - MAINTAIN_TTS Procedure
  - PRE_INSTANTIATION_SETUP Procedure
  - POST_INSTANTIATION_SETUP Procedure

Information about the operation is stored in the following data dictionary views when the operation is in process:

- DBA_RECOVERABLE_SCRIPT
- DBA_RECOVERABLE_SCRIPT_PARAMS
- DBA_RECOVERABLE_SCRIPT_BLOCKS
- DBA_RECOVERABLE_SCRIPT_ERRORS

For split and merge operations, the data dictionary views are populated at the database that contains the capture process. For the configuration operations, the data dictionary views are populated at the database where the replication configuration procedure was run.
When the operation completes successfully, metadata about the operation is moved from the `DBA_RECOVERABLE_SCRIPT` view to the `DBA_RECOVERABLE_SCRIPT_HIST` view. The other views, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`, retain information about the operation until it is purged automatically after 30 days.

When one of these operations encounters an error and stops, metadata about the operation remains in these views. In this case, you can either roll forward, roll back, or purge the metadata about the operation using the `RECOVER_OPERATION` procedure. If you choose to roll forward the operation, then correct conditions that caused the errors reported in `DBA_RECOVERABLE_SCRIPT_ERRORS` before proceeding.

For split and merge operations, run the `RECOVER_OPERATION` procedure at the database that contains the capture process. For the configuration operations, run the `RECOVER_OPERATION` procedure at the database where the replication configuration procedure was run.

---

**Note:**

- Regarding the configuration operations, the procedure must configure the environment directly (`perform_actions => TRUE`), not by generating a script, for information about the operation to be stored in the recoverable views and for the operation to be managed by the `RECOVER_OPERATION` procedure.
- To run the `RECOVER_OPERATION` procedure, both databases must be Oracle Database 10g Release 2 or later databases.

---

**See Also:**

*Oracle Streams Replication Administrator’s Guide*

---

**Syntax**

```
DBMS_STREAMS_ADM.RECOVER_OPERATION(
  script_id       IN  RAW,
  operation_mode  IN  VARCHAR2  DEFAULT 'FORWARD');
```

**Parameters**

**Table 165-31  RECOVER_OPERATION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_id</td>
<td>The operation id of the operation that is being rolled forward, rolled back, or purged. Query the <code>SCRIPT_ID</code> column of the <code>DBA_RECOVERABLE_SCRIPT</code> data dictionary view to determine the operation id.</td>
</tr>
</tbody>
</table>
Table 165-31  (Cont.) RECOVER_OPERATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| operation_mode    | If FORWARD, then the procedure rolls forward the operation. Specify FORWARD to try to complete the operation.  
                  | If ROLLBACK, then the procedure rolls back all of the actions performed in the operation. If the rollback is successful, then this option also moves the metadata about the operation from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view. The other views retain information about the operation for 30 days.  
                  | If PURGE, then the procedure moves the metadata about the operation from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view without rolling the operation back. The other views retain information about the operation for 30 days. |

165.6.32 REMOVE_QUEUE Procedure

This procedure removes the specified ANYDATA queue.

Specifically, this procedure performs the following actions:

1. Waits until all current enqueue and dequeue transactions commit.
2. Stops the queue, which means that no further enqueues into the queue or dequeues from the queue are allowed.
3. Drops the queue.
4. If the drop_unused_queue_table parameter is set to TRUE, then drops the queue table if it is empty and no other queues are using it.
5. If the cascade parameter is set to TRUE, then drops all of the Oracle Streams clients that are using the queue.

Note:
The specified queue must be a ANYDATA queue.

Syntax

```sql
DBMS_STREAMS_ADM.REMOVE_QUEUE(
    queue_name       IN  VARCHAR2,
    cascade          IN  BOOLEAN  DEFAULT FALSE,
    drop_unused_queue_table  IN  BOOLEAN  DEFAULT TRUE);
```
Parameters

Table 165-32 REMOVE_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the queue to remove, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>cascade</td>
<td>If TRUE, then the procedure drops any Oracle Streams clients that use the queue. If FALSE, then the procedure raises an error if there are any Oracle Streams clients that use the queue. Before you run this procedure with the cascade parameter set to FALSE, make sure no Oracle Streams clients are using the queue currently.</td>
</tr>
<tr>
<td>drop_unused_queue_table</td>
<td>If TRUE and the queue table for the queue is empty, then the procedure drops the queue table. The queue table is not dropped if it contains any messages or if it is used by another queue.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not drop the queue table.</td>
</tr>
</tbody>
</table>

165.6.33 REMOVE_RULE Procedure

This procedure removes the specified rule or all rules from the rule set associated with the specified capture process, synchronous capture, apply process, propagation, or messaging client.

If this procedure results in an empty positive rule set for a messaging client, then the procedure drops the messaging client automatically.

**Note:**
If a rule was automatically created by the system, and you want to drop the rule, then you should use this procedure to remove the rule instead of the DBMS_RULE_ADM.DROP_RULE procedure. If you use the DBMS_RULE_ADM.DROP_RULE procedure, then some metadata about the rule might remain.

Syntax

```
DBMS_STREAMS_ADM.REMOVE_RULE(
    rule_name         IN  VARCHAR2,
    streams_type      IN  VARCHAR2,
    streams_name      IN  VARCHAR2,
    drop_unused_rule  IN  BOOLEAN  DEFAULT TRUE,
    inclusion_rule    IN  BOOLEAN  DEFAULT TRUE);
```
### Parameters

#### Table 165-33  REMOVE_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule to remove, specified as [schema_name.]rule_name. If NULL, then the procedure removes all rules from the specified capture process, synchronous capture, apply process, propagation, or messaging client rule set. For example, to specify a rule in the hr schema named prop_rule1, enter hr.prop_rule1. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>streams_type</td>
<td>The type of Oracle Streams client: • Specify capture for a capture process. • Specify sync_capture for a synchronous capture. • Specify propagation for a propagation. • Specify apply for an apply process. • Specify dequeue for a messaging client.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the Oracle Streams client, which can be a capture process, synchronous capture, propagation, apply process, or messaging client. Do not specify an owner. If the specified Oracle Streams client does not exist, but there is metadata in the data dictionary that associates the rule with this client, then the procedure removes the metadata. If the specified Oracle Streams client does not exist, and there is no metadata in the data dictionary that associates the rule with this client, then the procedure raises an error.</td>
</tr>
<tr>
<td>drop_unused_rule</td>
<td>If TRUE and the rule is not in any rule set, then the procedure drops the rule from the database. If TRUE and the rule exists in any rule set, then the procedure does not drop the rule from the database. If FALSE, then the procedure does not drop the rule from the database.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure removes the rule from the positive rule set for the Oracle Streams client. If inclusion_rule is FALSE, then the procedure removes the rule from the negative rule set for the Oracle Streams client.</td>
</tr>
</tbody>
</table>

### 165.6.34 REMOVE_STREAMS_CONFIGURATION Procedure

This procedure removes the Oracle Streams configuration at the local database.

#### Syntax

```
DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION;
```

#### Usage Notes

Specifically, this procedure performs the following actions at the local database:

- Drops all capture processes
• If any tables have been prepared for instantiation, then aborts preparation for instantiation for the table using the `ABORT_TABLE_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package.

• If any schemas have been prepared for instantiation, then aborts preparation for instantiation for the schema using the `ABORT_SCHEMA_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package.

• If the database has been prepared for instantiation, then aborts preparation for instantiation for the database using the `ABORT_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package.

• Drops propagations that were created using either the `DBMS_STREAMS_ADM` package or the `DBMS_PROPAGATION_ADM` package. Before a propagation is dropped, its propagation job is disabled. Does not drop propagations that were created using the `DBMS_AQADM` package.

• Disables all propagation jobs used by propagations.

• Drops all apply processes. If there are apply errors in the error queue for an apply process, then this procedure deletes these apply errors before it drops the apply process.

• Removes specifications for DDL handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers.

• Removes specifications for message handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers.

• Removes specifications for precommit handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers.

• Removes the instantiation SCN and ignore SCN for each apply object and schema and for the entire database.

• Removes messaging clients.

• Unsets message notification specifications that were set using the `SET_MESSAGE_NOTIFICATION` procedure in the `DBMS_STREAMS_ADM` package.

• Removes specifications for procedure DML handlers and error handlers, but does not delete the PL/SQL procedures used by these handlers.

• Removes update conflict handlers.

• Removes specifications for substitute key columns for apply tables.

• Drops rule sets and rules that were created using the `DBMS_STREAMS_ADM` package.

• Drops unused rule sets that were used by capture processes, propagations, apply processes, and messaging clients, and removes the rules in these rule sets. These rules and rule sets are removed regardless of whether they were created using the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package.

This procedure stops capture processes and apply processes before it drops them.

This procedure does not drop rule sets or rules if they meet both of the following conditions:

• The rule sets or rules were created using the `DBMS_RULE_ADM` package.

• The rule sets or rules were not used by a capture process, propagation, apply process, or messaging client.
Note:
Running this procedure is dangerous. You should run this procedure only if you are sure you want to remove the entire Oracle Streams configuration at a database.

Note:
• Running this procedure repeatedly does not cause errors. If the procedure fails to complete, then you can run it again.
• This procedure commits multiple times.

See Also:
• STOP_CAPTURE Procedure in the DBMS_CAPTURE_ADM package
• STOP_APPLY Procedure in the DBMS_APPLY_ADM package
• REMOVE_RULE Procedure in the DBMS_STREAMS_ADM package

165.6.35 RENAME_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a column in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to TRUE, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

Note:
• The RENAME_COLUMN procedure supports the same data types supported by Oracle Streams capture processes.
• Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.
### See Also:

- *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations
- *Oracle Streams Concepts and Administration* for more information about the data types supported by Oracle Streams capture processes

### Syntax

```sql
DBMS_STREAMS_ADM.RENAME_COLUMN(
    rule_name         IN  VARCHAR2,
    table_name        IN  VARCHAR2,
    from_column_name  IN  VARCHAR2,
    to_column_name    IN  VARCHAR2,
    value_type        IN  VARCHAR2  DEFAULT '*',
    step_number       IN  NUMBER    DEFAULT 0,
    operation         IN  VARCHAR2  DEFAULT 'ADD');
```

### Parameters

**Table 165-34  RENAME_COLUMN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as <code>[schema_name.]rule_name</code>. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter <code>hr.employees12</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table in which the column is renamed in the row LCR, specified as <code>[schema_name.]object_name</code>. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>from_column_name</td>
<td>The name of the column to be renamed in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>to_column_name</td>
<td>The new name of the column in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to rename the column in the new values in the row LCR. Specify 'OLD' to rename the column in the old values in the row LCR. Specify '*' to rename the column in both the old and new values in the row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>
Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename column declarative rule-based transformations for the specified rule are removed that match the specified `table_name`, `column_name`, and `step_number` parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the `RENAME_COLUMN` procedure when one or more of these parameters is `NULL`:

<table>
<thead>
<tr>
<th>table_name</th>
<th>from_column_name</th>
<th>to_column_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename column transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename column transformations with the specified to_column_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename column transformations with the specified table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified column_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name and column_name for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>
### 165.6.36 RENAME_SCHEMA Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a schema in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

#### Note:

Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

#### See Also:

*Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

#### Syntax

```sql
DBMS_STREAMS_ADM.RENAME_SCHEMA(
    rule_name         IN  VARCHAR2,
    from_schema_name  IN  VARCHAR2,
    to_schema_name    IN  VARCHAR2,
    step_number       IN  NUMBER    DEFAULT 0,
    operation         IN  VARCHAR2  DEFAULT 'ADD');
```
Parameters

Table 165-35  RENAME_SCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>from_schema_name</td>
<td>The name of the schema to be renamed in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>to_schema_name</td>
<td>The new name of the schema in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule.</td>
</tr>
<tr>
<td></td>
<td>Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename schema declarative rule-based transformations for the specified rule are removed that match the specified from_schema_name, to_schema_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_SCHEMA procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>from_schema_name</th>
<th>to_schema_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename schema transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename schema transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename schema transformations with the specified to_schema_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename schema transformations with the specified from_schema_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename schema transformations with the specified to_schema_name for the specified rule.</td>
</tr>
</tbody>
</table>
This procedure either adds or removes a declarative rule-based transformation which renames a table in a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to TRUE, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

### Note:
Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

### See Also:
*Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

**Syntax**

```sql
DBMS_STREAMS_ADM.RENAME_TABLE(
    rule_name IN VARCHAR2,
    from_table_name IN VARCHAR2,
    to_table_name IN VARCHAR2,
    step_number IN NUMBER DEFAULT 0,
    operation IN VARCHAR2 DEFAULT 'ADD');
```
Parameters

Table 165-36  RENAME_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>from_table_name</td>
<td>The name of the table to be renamed in each row LCR that satisfies the rule, specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>to_table_name</td>
<td>The new name of the table in each row LCR that satisfies the rule, specified as [schema_name.]object_name. For example, humres.staff. The transformation can rename the table only, the schema only, or the table and the schema. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename table declarative rule-based transformations for the specified rule are removed that match the specified from_table_name, to_table_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_TABLE procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>from_table_name</th>
<th>to_table_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename table transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename table transformations with the specified to_table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename table transformations with the specified from_table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>from_table_name</td>
<td>to_table_name</td>
<td>step_number</td>
<td>Result</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations with the specified to_table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations with the specified from_table_name and to_table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations with the specified from_table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename table transformations with the specified from_table_name, to_table_name, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>

### 165.6.38 SET_MESSAGE_NOTIFICATION Procedure

This procedure sets a notification for messages that can be dequeued by a specified Oracle Streams messaging client from a specified queue.

A notification is sent when a message is enqueued into the specified queue and the specified messaging client can dequeue the message because the message satisfies its rule sets.

**Note:**

- Currently, messaging clients cannot dequeue buffered messages.
- The [DBMS_AQ](https://docs.oracle.com/en/database/oracle/database/121/appdev/sqlrefشعب/09228.html) package can also configure notifications. The [DBMS_AQ](https://docs.oracle.com/en/database/oracle/database/121/appdev/sqlrefشعب/09228.html) package provides some notification features that are not available in [DBMS_STREAMS_ADM](https://docs.oracle.com/en/database/oracle/database/121/appdev/sqlrefشعب/09228.html) package, such as buffered message notifications and notification grouping by time.

**Syntax**

```sql
DBMS_STREAMS_ADM.SET_MESSAGE_NOTIFICATION(
  streams_name          IN  VARCHAR2,
  notification_action   IN  VARCHAR2,
  notification_type     IN  VARCHAR2     DEFAULT 'PROCEDURE',
  notification_context  IN  ANYDATA      DEFAULT NULL,
  include_notification  IN  BOOLEAN      DEFAULT TRUE,
  queue_name            IN  VARCHAR2     DEFAULT 'streams_queue');
```
Parameters

Table 165-37  SET_MESSAGE_NOTIFICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_name</td>
<td>The name of the Oracle Streams messaging client. Do not specify an owner. For example, if the user strmadmin is the messaging client, then specify strmadmin.</td>
</tr>
</tbody>
</table>
| notification_action | The action to be performed on message notification. Specify one of the following:  
  - For URL notifications, specify a URL without the prefix http://. For example, to specify the URL http://www.company.com:8080, enter the following:  
    www.company.com:8080  
  - For email notifications, specify an email address. For example, to specify an the email address xyz@company.com, enter the following:  
    xyz@company.com  
  - For PL/SQL procedure notifications, specify an existing user-defined PL/SQL procedure in the form [schema_name.]procedure_name. If the schema_name is not specified, then the user who invokes the SET MESSAGE NOTIFICATION procedure is the default. The procedure must be a PLSQLCALLBACK data structure. For example, to specify a procedure named notify_orders in the oe schema, enter the following:  
    oe.notify_orders  

See Also: Examples for more information about message notification procedures

| notification_type | The type of notification. Specify one of the following:  
  - HTTP if you specified a URL for notification_action  
  - MAIL if you specified an email address for notification_action  
  - PROCEDURE if you specified a user-defined procedure for notification_action  
  The type must match the specification for the notification_action parameter. |

| notification_context | The context of the notification. The context must be specified using RAW datatype information. For example, to specify the hexadecimal equivalent of 'FF', enter the following:  
    ANYDATA.ConvertRaw(HEXTORAW('FF'))  
  The notification context is passed the PL/SQL procedure in procedure notifications and is not relevant for mail or HTTP notifications. |
Table 165-37  (Cont.) SET_MESSAGE_NOTIFICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_notification</td>
<td>If TRUE, then the procedure adds this notification for the specified streams_name and queue_name. That is, specifying TRUE turns on the notification for the streams_name and queue_name. If FALSE, then the procedure removes this notification for the specified streams_name and queue_name. That is, specifying FALSE turns off the notification for the streams_name and queue_name. If you specify FALSE, then this procedure ignores any specified values for the notification_action or notification_context parameters.</td>
</tr>
<tr>
<td>queue_name</td>
<td>The name of a local ANYDATA queue, specified as [schema_name.][queue_name]. The current database must contain the queue. The specified queue must be a ANYDATA queue. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>

Usage Notes

You can specify one of the following types of notifications:

- An email address to which message notifications are sent. When a relevant message is enqueued into the queue, an email with the message properties is mailed to the specified email address.
- A PL/SQL procedure to be invoked on a notification. When a relevant message is enqueued into the queue, the specified PL/SQL procedure is invoked with the message properties. This PL/SQL procedure can dequeue the message.
- An HTTP URL to which the notification is posted. When a relevant message is enqueued into the queue, a notification with the message properties is posted to the specified URL specified.

A client does not need to be connected to the database to receive a notification.

If you register for email notifications, then you should use the DBMS_AQELM package to set the host name and port name for the SMTP server that will be used by the database to send email notifications. If required, then you should set the send-from email address, which is set by the database as the sent_from field. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, you might want to use the DBMS_AQELM package to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

Each notification is an AQXm1Notification, which includes the following:

- notification_options, which includes the following:
  - destination - The destination queue from which the message was dequeued
  - consumer_name - The name of the messaging client that dequeued the message
- message_set - The set of message properties
Examples

If you use a message notification procedure, then this PL/SQL procedure must have the following signature:

```sql
PROCEDURE procedure_name(
  context  IN  ANYDATA,
  reginfo  IN  SYS.AQ$_REG_INFO,
  descr    IN  SYS.AQ$_DESCRIPTOR);
```

Here, `procedure_name` stands for the name of the procedure. The procedure is a `PLSQLCALLBACK` data structure that specifies the user-defined PL/SQL procedure to be invoked on message notification.

The following is a simple example of a notification procedure that dequeues a message of type `oe.user_msg` using the message identifier and consumer name sent by the notification.

To complete the example, first create the type:

```sql
CREATE TYPE oe.user_msg AS OBJECT(
  object_name    VARCHAR2(30),
  object_owner   VARCHAR2(30),
  message        VARCHAR2(50));
/
```

Next, create the procedure:

```sql
CREATE OR REPLACE PROCEDURE oe.notification_dequeue(
  context ANYDATA,
  reginfo SYS.AQ$_REG_INFO,
  descr SYS.AQ$_DESCRIPTOR)
AS
  dequeue_options     DBMS_AQ.DEQUEUE_OPTIONS_T;
  message_properties  DBMS_AQ.MESSAGE_PROPERTIES_T;
  message_handle      RAW(16);
  message             ANYDATA;
  oe_message          oe.user_msg;
  rc                  PLS_INTEGER;
BEGIN
  -- Get the message identifier and consumer name from the descriptor
  dequeue_options.msgid := descr.msg_id;
  dequeue_options.consumer_name := descr.consumer_name;
  -- Dequeue the message
  DBMS_AQ.DEQUEUE(
    queue_name => descr.queue_name,
    dequeue_options => dequeue_options,
    message_properties => message_properties,
    message_handle => message_handle,
    message_properties => message_properties,
    return_rc => rc);
  oe_message.object_name := message.object_name;
  oe_message.object_owner := message.object_owner;
  oe_message.message := message.message;
  /* Process the message contents */
END;
```

See Also:

- The documentation for the `DBMS_AQELM` package for more information on email notifications and HTTP notifications
- *Oracle Streams Concepts and Administration* for more information about how rules are used in Oracle Streams
payload = message,
msgid = message_handle);
rc := message.getobject(oe_message);
COMMIT;
END;
/

See Also:
Oracle Database PL/SQL Packages and Types Reference for more information about PLSQLCALLBACK data structures

165.6.39 SET_MESSAGE_TRACKING Procedure

This procedure sets the tracking label for logical change records (LCRs) produced by the current session.

This procedure affects only the current session. Any LCRs produced by the current session are tracked, including captured LCRs and persistent LCRs.

Note:
The tracking label set by this procedure does not track non-LCR messages.

See Also:
GET_MESSAGE_TRACKING Function

Syntax

DBMS_STREAMS_ADM.SET_MESSAGE_TRACKING(
    tracking_label IN VARCHAR2 DEFAULT 'Streams_tracking',
    actions IN NUMBER DEFAULT DBMS_STREAMS_ADM.ACTION_MEMORY);

Parameters

Table 165-38 SET_MESSAGE_TRACKING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracking_label</td>
<td>The label used to track the LCRs produced by the session.</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to NULL to stop message tracking in the current session.</td>
</tr>
<tr>
<td></td>
<td>The size limit for a label is 4,000 bytes.</td>
</tr>
</tbody>
</table>
Table 165-38  (Cont.) SET_MESSAGE_TRACKING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actions</td>
<td>When DBMS_STREAMS_ADM.ACTION_MEMORY is specified, the LCRs are tracked in memory, and the V$STREAMS_MESSAGE_TRACKING dynamic performance view is populated with information about the LCRs. Currently, DBMS_STREAMS_ADM.ACTION_MEMORY is the only valid setting for this parameter. The value specified for this parameter is an enumerated constant. Enumerated constants must be prefixed with the package name.</td>
</tr>
</tbody>
</table>

165.6.40 SET_RULE_TRANSFORM_FUNCTION Procedure

This procedure sets or removes the transformation function name for a custom rule-based transformation.

Syntax

DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
  rule_name IN VARCHAR2,
  transform_function IN VARCHAR2);

Parameters

Table 165-39  SET_RULE_TRANSFORM_FUNCTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule whose rule-based transformation function you are setting or removing, specified as [schema_name.]rule_name. For example, to specify a rule in the hr schema named prop_rule1, enter hr.prop_rule1. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
| transform_function | Either the name of the transformation function to be used in the rule-based transformation for the rule or NULL. If you specify a transformation function name, then specify an existing function in one of the following forms:
  • [schema_name.]function_name
  • [schema_name.]package_name.function_name
  If the function is in a package, then you must specify the package_name. For example, to specify a function in the transform_pkg package in the hr schema named executive_to_management, enter hr.transform_pkg.executive_to_management. An error is returned if the specified procedure does not exist.
  If the schema_name is not specified, then the user who invokes the rule-based transformation function is the default.
  If you specify NULL, then the SET_RULE_TRANSFORM_FUNCTION procedure removes the current custom rule-based transformation from the rule. |
Usage Notes

The following sections contain usage notes for this procedure:

• Transformation Function Signature
• Rule Action Context
• User Who Calls the Transformation Function
• Function Verification

Transformation Function Signature

A custom rule-based transformation function always operates on one message, but it can return one message or many messages. A custom rule-based transformation function that returns one message is a one-to-one transformation function. A one-to-one transformation function must have the following signature:

FUNCTION user_function (parameter_name IN ANYDATA)
RETURN ANYDATA;

Here, user_function stands for the name of the function and parameter_name stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an ANYDATA encapsulation of a message.

A custom rule-based transformation function that can return multiple messages is a one-to-many transformation function. A one-to-many transformation function must have the following signature:

FUNCTION user_function (parameter_name IN ANYDATA)
RETURN STREAMS$_ANYDATA_ARRAY;

Here, user_function stands for the name of the function and parameter_name stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an array that contains zero or more ANYDATA encapsulations of a message. If the array contains zero ANYDATA encapsulations of a message, then the original message is discarded.

The STREAMS$_ANYDATA_ARRAY type is an Oracle-supplied type that has the following definition:

CREATE OR REPLACE TYPE SYS.STREAMS$_ANYDATA_ARRAY
   AS VARRAY(2147483647) of ANYDATA
/

The following restrictions apply to custom rule-based transformations that use one-to-many functions:

• Rules that are associated with one-to-many functions are supported for Oracle Streams capture processes only. These rules must not be added to rule sets used by other Oracle Streams clients, including propagations, apply processes, and messaging clients.

• One-to-many functions only can operate on row logical change records (row LCRs). They cannot operate on DDL LCRs.
• Row LCRs returned by a one-to-many function cannot contain piecewise LOB, LONG, or LONG RAW operations.

• The one-to-many function must return row LCRs in the correct order. The order of row LCRs in the array (starting from index 1) is the order that the row LCRs will be executed in the transaction.

When an apply process dequeues row LCRs that are the result of a transformation by a one-to-many function, the apply process uses the instantiation SCN of the LCR passed to the one-to-many function for all of row LCRs.

**Note:**

• An error is raised if a one-to-one or one-to-many transformation function returns NULL.

• Only one custom rule-based transformation can be specified for a particular rule. You cannot specify both a one-to-one and a one-to-many transformation function for the same rule.

• For any LCR constructed and returned by a custom rule-based transformation, the source_database_name, transaction_id, and scn parameter values must match the values in the original LCR. Oracle automatically specifies the values in the original LCR for these parameters, even if an attempt is made to construct LCRs with different values.

**Rule Action Context**

This procedure modifies the specified rule's action context to specify the transformation. A rule's action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to TRUE for a message. The client of the rules engine can be a user-created application or an internal feature of Oracle, such as Oracle Streams. The Oracle Streams clients include capture processes, synchronous captures, apply processes, and messaging clients. The information in an action context is an object of type SYS.RE$NV_LIST, which consists of a list of name-value pairs.

A custom rule-based transformation in Oracle Streams always consists of the following name-value pair in an action context:

• If the function is a one-to-one transformation function, then the name is STREAMS$_TRANSFORM_FUNCTION. If the function is a one-to-many transformation function, then the name is STREAMS$_ARRAY_TRANS_FUNCTION.

• The value is a ANYDATA instance containing a PL/SQL function name specified as a VARCHAR2. This function performs the transformation.

**User Who Calls the Transformation Function**

The user that calls the transformation function must have EXECUTE privilege on the function. The following list describes which user calls the transformation function:

• If a transformation is specified for a rule used by a capture process, then the user who calls the transformation function is the capture user for the capture process.

• If a transformation is specified for a rule used by a synchronous capture, then the user who calls the transformation function is the capture user for the synchronous capture.
• If a transformation is specified for a rule used by a propagation, then the user who calls the transformation function is the owner of the source queue for the propagation.

• If a transformation is specified on a rule used by an apply process, then the user who calls the transformation function is the apply user for the apply process.

• If a transformation is specified on a rule used by a messaging client, then the user who calls the transformation function is the user who invokes the messaging client.

Function Verification

This procedure does not verify that the specified transformation function exists. If the function does not exist, then an error is raised when an Oracle Streams client tries to invoke the transformation function.

165.6.41 SET_TAG Procedure

This procedure sets the binary tag for all redo entries subsequently generated by the current session.

Each redo entry generated by DML or DDL statements in the current session will have this tag. This procedure affects only the current session.

See Also:

• "GET_TAG Function"
• Oracle Streams Replication Administrator’s Guide for more information about tags

Syntax

```
DBMS_STREAMS_ADM.SET_TAG(
    tag  IN RAW DEFAULT NULL);
```

Parameters

Table 165-40 SET_TAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>The binary tag for all subsequent redo entries generated by the current session. A raw value is a sequence of bytes, and a byte is a sequence of bits. By default, the tag for a session is NULL. The size limit for a tag value is 2000 bytes.</td>
</tr>
</tbody>
</table>

Usage Notes

To set the tag to the hexadecimal value of '17' in the current session, run the following procedure:

```
EXEC DBMS_STREAMS_ADM.SET_TAG(tag => HEXTORAW('17'));
```

The following are considerations for the SET_TAG procedure:
• This procedure is not transactional. That is, the effects of `SET_TAG` cannot be rolled back.

• If the `SET_TAG` procedure is run to set a non-`NULL` session tag before a data dictionary build has been performed on the database, then the redo entries for a transaction that started before the dictionary build might not include the specified tag value for the session. Therefore, perform a data dictionary build before using the `SET_TAG` procedure in a session. A data dictionary build happens when the `DBMS_CAPTURE_ADM.BUILD` procedure is run. The `BUILD` procedure can be run automatically when a capture process is created.

See Also:

BUILD Procedure

165.6.42 SET_UP_QUEUE Procedure

This procedure creates a queue table and a `ANYDATA` queue.

Syntax

```sql
DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table     IN  VARCHAR2  DEFAULT 'streams_queue_table',
    storage_clause  IN  VARCHAR2  DEFAULT NULL,
    queue_name      IN  VARCHAR2  DEFAULT 'streams_queue',
    queue_user      IN  VARCHAR2  DEFAULT NULL,
    comment         IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 165-41  SET_UP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>queue_table</code></td>
<td>The name of the queue table specified as [schema_name.]queue_table_name. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default. If the queue table owner is not specified, then the procedure specifies the user who runs this procedure automatically as the queue table owner. Queue table names can be a maximum of 24 bytes.</td>
</tr>
<tr>
<td><code>storage_clause</code></td>
<td>The storage clause for queue table. The storage parameter is included in the <code>CREATE TABLE</code> statement when the queue table is created. You can specify any valid table storage clause. If a tablespace is not specified here, then the procedure creates the queue table and all its related objects in the default user tablespace of the user who runs this procedure. If a tablespace is specified here, then the procedure creates the queue table and all its related objects in the tablesapce specified in the storage clause. If <code>NULL</code>, then the procedure uses the storage characteristics of the tablespace in which the queue table is created.</td>
</tr>
</tbody>
</table>

See Also: Oracle Database SQL Language Reference for more information about storage clauses.
Table 165-41  (Cont.) SET_UP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the queue that will function as the ANYDATA queue, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue. If the schema is not specified, then the procedure uses the queue table owner. The owner of the queue table must also be the owner of the queue. The queue owner automatically has privileges to perform all queue operations on the queue. If the schema is not specified for this parameter, and the queue table owner is not specified in queue_table, then the current user is the default. Queue names can be a maximum of 24 bytes.</td>
</tr>
<tr>
<td>queue_user</td>
<td>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option. If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</td>
</tr>
<tr>
<td>comment</td>
<td>The comment for the queue</td>
</tr>
</tbody>
</table>

Usage Notes

Set up includes the following actions:

- If the specified queue table does not exist, then this procedure runs the CREATE_QUEUE_TABLE procedure in the DBMS_AQADM package to create the queue table with the specified storage clause. If this procedure creates the queue table, then it creates a multiple consumer ANYDATA queue that is both a secure queue and a transactional queue.

  Also, if the database is Oracle Database 10g release 2 or later, the sort_list setting in CREATE_QUEUE_TABLE is set to commit_time. If the database is a release before Oracle Database 10g release 2, the sort_list setting in CREATE_QUEUE_TABLE is set to enq_time.

- If the specified queue table exists, then the queue uses the properties of the existing queue table.

- If the specified queue name does not exist, then this procedure runs the CREATE_QUEUE procedure in the DBMS_AQADM package to create the queue.

- This procedure starts the queue.

- If a queue user is specified, then this procedure configures this user as a secure queue user of the queue and grants ENQUEUE and DEQUEUE privileges on the queue to the specified queue user.

  To configure the queue user as a secure queue user, this procedure creates an Advanced Queuing agent with the same name as the user name, if one does not exist. If an agent with this name exists and is associated with the queue user only, then it is used. SET_UP_QUEUE then runs the ENABLE_DB_ACCESS procedure in the DBMS_AQADM package, specifying the agent and the user.
**Note:**

- To enqueue messages into and dequeue messages from a queue, a queue user must have `EXECUTE` privilege on the `DBMS_STREAMS_MESSAGING` package or the `DBMS_AQ` package. The `SET_UP_QUEUE` procedure does not grant this privilege.
- If the agent that `SET_UP_QUEUE` tries to create exists and is associated with a user other than the user specified by `queue_user`, then the procedure raises an error. In this case, rename or remove the existing agent, and retry `SET_UP_QUEUE`.

**See Also:**

`Oracle Streams Concepts and Administration` for more information about secure queue users.

### 165.6.43 SPLIT_STREAMS Procedure

This procedure splits one stream flowing from a capture process off from all of the other streams flowing from the capture process.

This procedure is intended for an Oracle Streams replication environment in which a capture process captures changes that are propagated to two or more destination databases. When one destination of a propagation stops accepting the captured changes, the changes remain in the capture process's queue. The queue can grow and begin to spill messages to the hard disk, degrading the performance of the Oracle Streams environment. A destination might stop accepting changes for several reasons. For example, the destination database might be down.

Specifically, this procedure performs the following actions:

1. Creates a new queue at the database running the capture process. The new queue is called the cloned queue because it is a clone of the queue used by the original stream. The new queue will be used by the new, cloned capture process, and it will be the source queue for the new, cloned propagation.

2. Creates a new propagation that propagates messages from the source queue created in Step 1 to the existing destination queue. The new propagation is called the cloned propagation because it is a clone of the propagation used by the original stream. The cloned propagation uses the same rule set as the original propagation.

3. Stops the capture process.

4. Queries the acknowledge SCN for the original propagation. The acknowledged SCN is the last SCN acknowledged by the apply process that applies the changes sent by the propagation.

5. Creates a new capture process. The new capture process is called the cloned capture process because it is a clone of the capture process used by the original stream. The procedure sets the start SCN for the cloned capture process to the value of the
acknowledged SCN queried in Step 4. The cloned capture process uses the same
rule set as the original capture process.

6. Drops the original propagation.

7. Starts the original capture process with the start SCN set to the acknowledged
SCN queried in Step 4.

8. If the auto_merge_threshold parameter is set to a positive number, then creates
an Oracle Scheduler job to run the MERGE_STREAMS_JOB procedure at set intervals
according to its schedule. When the two streams are within the specified merge
threshold, the MERGE_STREAMS_JOB procedure runs the MERGE_STREAMS procedure
to merge the streams automatically.

After the SPLIT_STREAMS procedure has finished running, the cloned capture process is
disabled. When the problem at the destination database is solved, and the destination
queue can accept changes, you should start the cloned capture process using the
START_CAPTURE procedure in the DBMS_CAPTURE_ADM package.

The DBA_STREAMS_SPLIT_MERGE view contains the name of each cloned component
and information about the split and merge operation.

---

**Note:**

If the original capture process is a downstream capture process, then you
must configure the cloned capture process to read the redo log from the
source database before you start the cloned capture process.

---

**See Also:**

- "MERGE_STREAMS Procedure"
- "MERGE_STREAMS_JOB Procedure"
- Oracle Streams Replication Administrator's Guide for instructions on
  using the SPLIT_STREAMS procedure

---

**Syntax**

```sql
DBMS_STREAMS_ADM.SPLIT_STREAMS(
    propagation_name IN VARCHAR2,
    cloned_propagation_name IN VARCHAR2 DEFAULT NULL,
    cloned_queue_name IN VARCHAR2 DEFAULT NULL,
    cloned_capture_name IN VARCHAR2 DEFAULT NULL,
    perform_actions IN BOOLEAN DEFAULT TRUE,
    script_name IN VARCHAR2 DEFAULT NULL,
    script_directory_object IN VARCHAR2 DEFAULT NULL,
    auto_merge_threshold IN NUMBER DEFAULT NULL,
    schedule_name IN OUT VARCHAR2,
    merge_job_name IN OUT VARCHAR2);```

---
Parameters

Table 165-42  SPLIT_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagation that cannot send messages to its destination queue. The specified propagation is the propagation for the stream that is being split off from the other streams. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
<tr>
<td>cloned_propagation_name</td>
<td>The name of the new propagation created by this procedure for the stream that is split off. If NULL, then the system generates a propagation name.</td>
</tr>
<tr>
<td>cloned_queue_name</td>
<td>The name of the new queue created by this procedure for the stream that is split off. If NULL, then the system generates a queue name.</td>
</tr>
<tr>
<td>cloned_capture_name</td>
<td>The name of the new capture process created by this procedure for the stream that is split off. If NULL, then the system generates a capture process name.</td>
</tr>
</tbody>
</table>
| perform_actions            | If TRUE, then the procedure performs the necessary actions to split the stream directly. If FALSE, then the procedure does not perform the necessary actions to split the stream directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:  
  * script_name  
  * script_directory_object |
| script_name                | If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to split the stream. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file. If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to split the stream directly. If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to split the stream directly and does not generate a script. If NULL and the perform_actions parameter is FALSE, then the procedure raises an error. |
| script_directory_object    | The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error. |

Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.
Table 165-42  (Cont.) SPLIT_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_merge_threshold</td>
<td>If a positive number is specified, then the stream that was split off is automatically merged back into all of the other streams flowing from the capture process by an Oracle Scheduler job. The job runs the MERGE_STREAMS_JOB procedure at set intervals according to its schedule. The value of the CAPTURE_MESSAGE&gt;Create_TIME column for each capture process in the GV$STREAMS_CAPTURE dynamic performance view determines when the streams are merged. Specifically, if the difference, in seconds, between CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process and the original capture process is less than or equal to the value specified for the auto_merge_threshold parameter, then the two streams are merged automatically. The cloned capture process must be started before the split stream can be merged back with the original stream. If NULL or 0 (zero) is specified, then the split stream is not merged back with the original stream automatically. To merge the split stream with the original stream, run the MERGE_STREAM procedure manually when the CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process catches up to, or nearly catches up to, the CAPTURE_MESSAGE_CREATE_TIME of the original capture process. The CAPTURE_MESSAGE_CREATE_TIME records the time when a captured change was recorded in the redo log.</td>
</tr>
<tr>
<td>schedule_name</td>
<td>The Oracle Scheduler schedule name, specified as [schema_name.]schedule_name. For example, strmadmin.merge_schedule. If the schema is not specified, then the current user is the default. If auto_merge_threshold is a non-NULL positive number, then the schedule is used by the job that will automatically merge the streams at the appropriate time. You can specify a schedule name to adhere to naming conventions or to track the schedule more easily. If NULL and auto_merge_threshold is a non-NULL positive number, then the system generates a schedule name. If auto_merge_threshold is NULL or 0 (zero), then this parameter must be NULL. If this procedure creates a schedule, the schedule starts when the procedure completes. You can modify the schedule to control how often the merge job is run. If an existing schedule name is specified, an error is raised.</td>
</tr>
</tbody>
</table>
Table 165-42  (Cont.) SPLIT_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge_job_name</td>
<td>The Oracle Scheduler job name, specified as [schema_name.]merge_job_name. For example, strmadmin.merge_job. If the schema is not specified, then the current user is the default. If auto_merge_threshold is a non-NULL positive number, then the job will automatically merge the streams at the appropriate time. Specify a merge job name to adhere to naming conventions or to track the job more easily. If NULL and auto_merge_threshold is a non-NULL positive number, then the system generates a job name. If auto_merge_threshold is NULL or 0 (zero), then this parameter must be NULL. If an existing job name is specified, an error is raised.</td>
</tr>
</tbody>
</table>

See Also:

*Oracle Database Administrator's Guide* for information about Oracle Scheduler
166

DBMS_STREAMS_ADVISOR_ADM

The `DBMS_STREAMS_ADVISOR_ADM` package, one of a set of Oracle Streams packages, provides an interface to gather information about an Oracle Streams environment and advise database administrators based on the information gathered.

This package is part of the Oracle Streams Performance Advisor.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Views
- Summary of `DBMS_STREAMS_ADVISOR_ADM` Subprograms

See Also:

`Oracle Streams Concepts and Administration` for instructions about using this package

166.1 DBMS_STREAMS_ADVISOR_ADM Overview

The `DBMS_STREAMS_ADVISOR_ADM` package enables you to gather and analyze information about an Oracle Streams environment.

You can use this information in the following ways:

- To populate data dictionary views with an Oracle Streams topology that contains information about the Oracle Streams components at one or more databases
- To examine the Oracle Streams components at one or more databases in your environment and the ways in which information flows through streams that include these components
- To analyze the performance of the Oracle Streams components in your environment
- To detect performance problems with Oracle Streams components and correct these problems

See Also:

`Oracle Streams Concepts and Administration` for instructions about using this package
166.2 DBMS_STREAMS_ADVISOR_ADM Security Model

Security on this package can be controlled by either granting EXECUTE on this package to selected users or roles, or granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

See Also:
Oracle Streams Replication Administrator's Guide for information about configuring an Oracle Streams administrator

166.3 DBMS STREAMS_ADVISOR_ADM Constants

The DBMS_STREAMS_ADVISOR_ADM package defines several enumerated constants for specifying parameter values.

Enumerated constants must be prefixed with the package name. For example, DBMS_DBMS_ADVISOR_ADM.CAPTURE_TYPE.
Table 166-1 DBMS_STREAMS_ADVISOR_ADM Parameters With Enumerated Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component_type</td>
<td>CAPTURE_TYPE</td>
<td>NUMBER</td>
<td>CAPTURE_TYPE indicates that the Oracle Streams component is a capture process. The constant number for this option is 1.</td>
</tr>
<tr>
<td></td>
<td>PROPAGATION_SENDER_TYPE</td>
<td></td>
<td>PROPAGATION_SENDER_TYPE indicates that the Oracle Streams component is a propagation sender. The constant number for this option is 2.</td>
</tr>
<tr>
<td></td>
<td>PROPAGATION_RECEIVER_TYPE</td>
<td></td>
<td>PROPAGATION_RECEIVER_TYPE indicates that the Oracle Streams component is a propagation receiver. The constant number for this option is 3.</td>
</tr>
<tr>
<td></td>
<td>APPLY_TYPE</td>
<td></td>
<td>APPLY_TYPE indicates that the Oracle Streams component is an apply process. The constant number for this option is 4.</td>
</tr>
<tr>
<td></td>
<td>QUEUE_TYPE</td>
<td></td>
<td>QUEUE_TYPE indicates that the Oracle Streams component is a queue. The constant number for this option is 5.</td>
</tr>
</tbody>
</table>

166.4 DBMS_STREAMS_ADVISOR_ADM Views

The views in the package contain information about Oracle Streams components and performance.

DBMS_STREAMS_ADVISOR_ADM uses the following views:

- **DBA_STREAMS_TP_COMPONENT** contains information about each Oracle Streams component at each database.
- **DBA_STREAMS_TP_COMPONENT_LINK** contains information about how messages flow between Oracle Streams components.
- **DBA_STREAMS_TP_COMPONENT_STAT** contains temporary performance statistics about each Oracle Streams component.
- **DBA_STREAMS_TP_DATABASE** contains information about each database that contains Oracle Streams components.
- **DBA_STREAMS TP_PATH_BOTTLENECK** contains temporary information about Oracle Streams components that might be slowing down the flow of a stream.
- **DBA STREAMS TP_PATH_STAT** contains temporary performance statistics about each stream path that exists in the Oracle Streams topology.

The topology information is stored permanently in the following data dictionary views:
**DBA_STREAMS_TP_DATABASE, DBA_STREAMS_TP_COMPONENT, and DBA_STREAMS_TP_COMPONENT_LINK.**

However, the following views contain temporary information:
**DBA_STREAMS_TP_COMPONENT_STAT, DBA_STREAMS_TP_PATH_BOTTLENECK, and DBA_STREAMS_TP_PATH_STAT. Some of the data in these views is retained only for the user**
session that runs the `ANALYZE_CURRENT_PERFORMANCE` procedure. When this user session ends, this temporary information is purged.

**See Also:**

- *Oracle Database Reference*
- *Oracle Streams Concepts and Administration* for sample queries that use these views

### 166.5 DBMS_STREAMS_ADVISOR_ADM Operational Notes

This section contains operational notes for the `DBMS_STREAMS_ADVISOR_ADM` package.

**Oracle Streams Components Analyzed by the DBMS_STREAMS_ADVISOR_ADM Package**

The `DBMS_STREAMS_ADVISOR_ADM` analyzes the following Oracle Streams components at the specified databases:

- Capture processes
- Propagations
- Apply processes
- Queues

The `DBMS_STREAMS_ADVISOR_ADM` package does not analyze the following Oracle Streams components:

- Synchronous captures
- Messaging clients

**General Steps for Running the Oracle Streams Performance Advisor and Analyzing the Information**

To use the `DBMS_STREAMS_ADVISOR_ADM` package, complete the following general steps:

1. Identify the database that you will use to gather the information.

   An administrative user at this database must meet the following requirements:

   - The user must have access to a database link to each database that contains Oracle Streams components.
   - The user must have been granted privileges using the `DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE` procedure, and each database link must connect to a user at the remote database that has been granted privileges using the `DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE` procedure.

   In an Oracle Streams environment, the Oracle Streams administrator uses this package.
If no database in your environment meets these requirements, then choose a database, configure the necessary database links, and grant the necessary privileges to the users before proceeding.

2. Connect as the administrative user to the database you identified previously, and remain connected to the session while you complete the remaining steps.

3. Run the `ANALYZE_CURRENT_PERFORMANCE` procedure.

4. Optionally, allow messages to flow in the environment for some time.

5. Optionally, rerun the `ANALYZE_CURRENT_PERFORMANCE` procedure one or more times.

6. Query the data dictionary views listed in "Views" to analyze the Oracle Streams environment.

7. If you want to update the information in the data dictionary views or if you add new Oracle Streams components to any database in the environment, repeat these steps.

---

**Note:**

When you exit the user session, the rate, bandwidth, event, and flow control statistics are purged from the data dictionary views.

---

### 166.6 Summary of DBMS_STREAMS_ADVISOR_ADM Subprograms

This table lists the `DBMS_STREAMS_ADVISOR_ADM` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ANALYZE_CURRENT_PERFORMANCE Procedure</code></td>
<td>Gathers information about the Oracle Streams components at one or more databases in your environment and analyzes Oracle Streams performance based on the information gathered</td>
</tr>
</tbody>
</table>

### 166.6.1 `ANALYZE_CURRENT_PERFORMANCE` Procedure

This procedure gathers information about the Oracle Streams components at one or more databases in your environment and analyzes Oracle Streams performance based on the information gathered.

The performance analyses includes:

- Calculating bottleneck components for each separate stream
- Calculating the throughput of each Oracle Streams component
- Calculating the latency of each Oracle Streams component
- Calculating the top wait event of each Oracle Streams component
- Calculating the message rate of each stream
• Calculating the transaction rate of each stream
The procedure places the gathered information in data dictionary views.

⚠️ **Note:**
The parameters in this procedure must all be either non-NULL or NULL.

⚠️ **See Also:**
• *Oracle Streams Concepts and Administration* for instructions on using this procedure
• "Views"

**Syntax**
```sql
DBMS_STREAMS_ADVISOR_ADM.ANALYZE_CURRENT_PERFORMANCE(
  component_name  IN  VARCHAR2  DEFAULT NULL,
  component_db    IN  VARCHAR2  DEFAULT NULL,
  component_type  IN  NUMBER    DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component_name</td>
<td>The name of the Oracle Streams component to analyze. For example, to analyze a capture process named capture01, then specify capture01. If NULL, then all of the Oracle Streams components are analyzed, and the other two parameters must also be NULL.</td>
</tr>
<tr>
<td>component_db</td>
<td>The global name of the database that contains the component specified in the component_name parameter. For example, if the db.net database contains the component, then specify db.net. If NULL, then all of the Oracle Streams components are analyzed, and the other two parameters must also be NULL.</td>
</tr>
</tbody>
</table>
| component_type  | The type of the component specified in the component_name parameter. If the component_name parameter is non-NULL, then specify one of the following:  
  - DBMS_STREAMS_ADVISOR_ADM.CAPTURE_TYPE  
  - DBMS_STREAMS_ADVISOR_ADM.PROPAGATION_SENDER_TYPE  
  - DBMS_STREAMS_ADVISOR_ADM.PROPAGATION_RECEIVER_TYPE  
  - DBMS_STREAMS_ADVISOR_ADM.APPLY_TYPE  
  - DBMS_STREAMS_ADVISOR_ADM.QUEUE_TYPE  
  See "Constants" for information about these constants. If NULL, then all of the Oracle Streams components are analyzed, and the other two parameters must also be NULL. |
The **DBMS_STREAMS_AUTH** package, one of a set of Oracle Streams packages, provides subprograms for granting privileges to Oracle Streams administrators and revoking privileges from Oracle Streams administrators.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Summary of DBMS_STREAMS_AUTH Subprograms**

### 167.1 DBMS_STREAMS_AUTH Overview

This package provides subprograms for granting privileges to Oracle Streams administrators and revoking privileges from Oracle Streams administrators.

**See Also:**

*Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Oracle Streams administrators.

### 167.2 DBMS_STREAMS_AUTH Security Model

Security on this package can be controlled by either granting `EXECUTE` on this package to selected users or roles, or by granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, connect as an administrative user who can create users, grant privileges, and create tablespaces when using this package.

### 167.3 Summary of DBMS_STREAMS_AUTH Subprograms

This table lists the **DBMS_STREAMS_AUTH** subprograms and briefly describes them.
Table 167-1  DBMS_STREAMS_AUTH Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT_ADMIN_PRIVILEGE Procedure</td>
<td>Either grants the privileges needed by a user to be an Oracle Streams administrator directly, or generates a script that you can use to grant these privileges</td>
</tr>
<tr>
<td>GRANT_REMOTE_ADMIN_ACCESS Procedure</td>
<td>Enables a remote Oracle Streams administrator to perform administrative actions at the local database by connecting to the grantee using a database link</td>
</tr>
<tr>
<td>REVOKE_ADMIN_PRIVILEGE Procedure</td>
<td>Either revokes Oracle Streams administrator privileges from a user directly, or generates a script that you can use to revoke these privileges</td>
</tr>
<tr>
<td>REVOKE_REMOTE_ADMIN_ACCESS Procedure</td>
<td>Disables a remote Oracle Streams administrator from performing administrative actions by connecting to the grantee using a database link</td>
</tr>
</tbody>
</table>

Note:
All subprograms commit unless specified otherwise.

167.3.1 GRANT_ADMIN_PRIVILEGE Procedure

This procedure either grants the privileges needed by a user to be an Oracle Streams administrator directly, or generates a script that you can use to grant these privileges.

Syntax

```sql
DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee           IN  VARCHAR2,
    grant_privileges  IN  BOOLEAN   DEFAULT TRUE,
    file_name         IN  VARCHAR2  DEFAULT NULL,
    directory_name    IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 167-2  GRANT_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user to whom privileges are granted</td>
</tr>
</tbody>
</table>
Table 167-2  (Cont.) GRANT_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant_privileges</td>
<td>If TRUE, then the procedure grants the privileges to the specified grantee directly, and adds the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view with YES for both the LOCAL_PRIVILEGES column and the ACCESS_FROM_REMOTE column. If the user has an entry in this data dictionary view, then the procedure does not make another entry, and no error is raised. If TRUE and any of the grant statements fail, then the procedure raises an error. If FALSE, then the procedure does not grant the privileges to the specified grantee directly, and does not add the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view. You specify FALSE when the procedure is generating a file that you will edit and then run. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file generated by the procedure. The file contains all of the statements that grant the privileges. If a file with the specified file name exists in the specified directory name, then the grant statements are appended to the existing file. If NULL, then the procedure does not generate a file.</td>
</tr>
<tr>
<td>directory_name</td>
<td>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>

Usage Notes

The user who runs the procedure must be an administrative user who can grant privileges to other users.

Specifically, the procedure grants the following privileges to the specified user:

- **The RESTRICTED SESSION** system privilege
- **EXECUTE** on the following packages:
  - DBMS_APPLY_ADM
  - DBMS_AQ
  - DBMS_AQADM
  - DBMS_AQIN
  - DBMS_AQELM
  - DBMS_CAPTURE_ADM
  - DBMS_FLASHBACK
– DBMS_LOCK
– DBMS_PROPAGATION_ADM
– DBMS_RULE_ADM
– DBMS_STREAMS_ADM
– DBMS_STREAMS_ADVISOR_ADM
– DBMS_STREAMS_HANDLER_ADM
– DBMS_STREAMS_MESSAGING
– DBMS_TRANSFORM

• Privileges to enqueue messages into and dequeue messages from any queue
• Privileges to manage any queue
• Privileges to create, alter, and execute any of the following types of objects in the user’s own schema and in other schemas:
  – Evaluation contexts
  – Rule sets
  – Rules

In addition, the grantee can grant these privileges to other users.

• SELECT_CATALOG_ROLE
• SELECT or READ privilege on data dictionary views related to Oracle Streams
• The ability to allow a remote Oracle Streams administrator to perform administrative actions through a database link by connecting to the grantee. This ability is enabled by running the GRANT_REMOTE_ADMIN_ACCESS procedure in this package.

Note:

– To view all of the statements run by the procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.
– This procedure does not grant any roles to the grantee.
– This procedure grants only the privileges necessary to configure and administer an Oracle Streams environment. You can grant more privileges to the grantee if necessary.

See Also:

– GRANTREMOTE_ADMIN_ACCESS Procedure
– Oracle Streams Replication Administrator’s Guide for more information about configuring an Oracle Streams administrator
167.3.2 GRANT_REMOTE_ADMIN_ACCESS Procedure

This procedure enables a remote Oracle Streams administrator to perform administrative actions at the local database by connecting to the grantee using a database link.

Syntax

```
DBMS_STREAMS_AUTH.GRANT_REMOTE_ADMIN_ACCESS(
    grantee IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user who allows remote access. The procedure adds the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view with YES for the ACCESS_FROM_REMOTE column. If the user has an entry in this data dictionary view, then the procedure does not make another entry. Instead, it updates the ACCESS_FROM_REMOTE column to YES.</td>
</tr>
</tbody>
</table>

Usage Notes

Typically, you run the procedure and specify a grantee at a local source database if a downstream capture process captures changes originating at the local source database. The Oracle Streams administrator at a downstream capture database administers the source database using this connection. You can also run the procedure at a database running an apply process so that a remote Oracle Streams administrator can set instantiation SCNs at the local database.

Note:

The GRANT_ADMIN_PRIVILEGE procedure runs this procedure.

See Also:

GRANT_ADMIN_PRIVILEGE Procedure

167.3.3 REVOKE_ADMIN_PRIVILEGE Procedure

This procedure either revokes Oracle Streams administrator privileges from a user directly, or generates a script that you can use to revoke these privileges.

Syntax

```
DBMS_STREAMS_AUTH.REVOKE_ADMIN_PRIVILEGE(
    grantee IN VARCHAR2,
    revoke_privileges IN BOOLEAN DEFAULT TRUE,
```
Parameters

Table 167-4 REVOKE_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user from whom privileges are revoked</td>
</tr>
<tr>
<td>revoke_privileges</td>
<td>If TRUE, then the procedure revokes the privileges from the specified user directly, and removes the user from the <code>DBA_STREAMS_ADMINISTRATOR</code> data dictionary view. If the user does not have a record in this data dictionary view, then the procedure does not remove a record from the view, and no error is raised. If TRUE and any of the revoke statements fail, then the procedure raises an error. A revoke statement will fail if the user is not granted the privilege that is being revoked. If FALSE, then the procedure does not revoke the privileges to the specified user directly, and does not remove the user from the <code>DBA_STREAMS_ADMINISTRATOR</code> data dictionary view. You specify FALSE when the procedure is generating a file that you will edit and then run. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure does not raise an error.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file generated by this procedure. The file contains all of the statements that revoke the privileges. If a file with the specified file name exists in the specified directory name, then the revoke statements are appended to the existing file. If NULL, then the procedure does not generate a file.</td>
</tr>
<tr>
<td>directory_name</td>
<td>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement <code>CREATE DIRECTORY</code>. If you specify a directory, then the user who invokes the procedure must have WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>

Usage Notes

The user who runs this procedure must be an administrative user who can revoke privileges from other users. Specifically, this procedure revokes the privileges granted by running the `GRANT_ADMIN_PRIVILEGE` procedure in this package.

Note:

To view all of the statements run by this procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.
167.3.4 REVOKE_REMOTE_ADMIN_ACCESS Procedure

This procedure disables a remote Oracle Streams administrator from performing administrative actions by connecting to the grantee using a database link.

Note: The REVOKE_ADMIN_PRIVILEGE procedure runs this procedure.

See Also: REVOKE_ADMIN_PRIVILEGE Procedure

Syntax

```
DBMS_STREAMS_AUTH.REVOKE_REMOTE_ADMIN_ACCESS(
    grantee  IN  VARCHAR2);
```

Parameters

Table 167-5  REVOKE_REMOTE_ADMIN_ACCESS Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user for whom access from a remote Oracle Streams administrator is disabled.</td>
</tr>
<tr>
<td></td>
<td>If a row for the grantee exists in the DBA_STREAMS_ADMINISTRATOR data dictionary view, then the procedure updates the ACCESS_FROM_REMOTE column for the grantee to NO. If, after this update, both the LOCAL_PRIVILEGES column and the ACCESS_FROM_REMOTE column are NO for the grantee, then the procedure removes the grantee from the view.</td>
</tr>
<tr>
<td></td>
<td>If no row for the grantee exists in the DBA_STREAMS_ADMINISTRATOR data dictionary view, then the procedure does not update the view and does not raise an error.</td>
</tr>
</tbody>
</table>
The **DBMS_STREAMS_HANDLER_ADM** package, one of a set of Oracle Streams packages, provides interfaces to manage statement DML handlers.

This chapter contains the following topics:

- Overview
- Security Model
- Views
- Operational Notes
- Summary of **DBMS_STREAMS_HANDLER_ADM** Subprograms

### 168.1 DBMS_STREAMS_HANDLER_ADM Overview

A statement DML handler runs one or more data manipulation language (DML) statements on row logical change records (row LCRs) that are dequeued by an apply process.

A single statement DML handler can include multiple statements, and you control the execution order of the statements.

Statement DML handlers are similar to procedure DML handlers for apply processes. Both statement DML handlers and procedure DML handlers provide custom processing of row changes that are encapsulated in row LCRs. Statement DML handlers and procedure DML handlers both run when an apply process dequeues a row LCR. However, statement DML handlers have the following advantages over procedure DML handlers:

- Statement DML handlers typically perform better than procedure DML handlers because statement DML handlers do not require PL/SQL processing.
- The syntax for statement DML handlers is same as DML syntax. Statement DML handlers do not require PL/SQL programming. Procedure DML handlers require PL/SQL programming.
- Statement DML handlers do not require the manipulation of **ANYDATA** values to access the information in row LCRs. Typically, procedure DML handlers must manipulate **ANYDATA** values.
- A statement DML handler can coexist with an error handler for same operation on the same database object. In contrast, you cannot specify both a procedure DML handler and an error handler for the same operation on the same database object.
Note:
You can specify multiple statement DML handlers for the same operation on the database object. In this case, the statement DML handlers can execute in any order, and each statement DML handler receives a copy of the original row LCR that was dequeued by the apply process.

See Also:
Oracle Streams Concepts and Administration

168.2 DBMS_STREAMS_HANDLER_ADM Security Model
Security on this package can be controlled by either granting EXECUTE on this package to selected users or roles, or by granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

See Also:
Oracle Streams Replication Administrator's Guide for information about configuring an Oracle Streams administrator

168.3 DBMS_STREAMS_HANDLER_ADM Views
The DBMS_STREAMS_HANDLER_ADM package uses these views.

- DBA_APPLY_DML_HANDLERS
- DBA_STREAMS_STMTS
- DBA_STREAMS_STMT_Handlers

They are described in detail in the Oracle Database Reference.
168.4 DBMS_STREAMS_HANDLER_ADM Operational Notes

This section contains operational notes about the DBMS_STREAMS_HANDLER_ADM package.

Statement Execution Order

Each statement in a statement DML handler has a unique execution sequence number. When a statement DML handler is invoked, it executes its statements in order from the statement with the lowest execution sequence number to the statement with the highest execution sequence number.

Supported SQL Statements

You can use statement DML handlers for any valid DML operation on a row logical change record (row LCR). For example, a statement DML handler can audit the DML changes made to a table.

The following SQL statements are supported in statement DML handlers:

- INSERT
- UPDATE
- DELETE
- MERGE

In addition, define variables are not supported in the SQL statements in a statement DML handler.

However, the SQL statements in a statement DML handler can include calls to member subprograms for the row LCR type (LCR$_ROW_RECORD), such as ADD_COLUMN, DELETE_COLUMN, KEEP_COLUMNS, and RENAME_COLUMN.

Note:

A statement DML handler cannot modify the value of a column in a row LCR.

See Also:

"LCR$_ROW_RECORD Type"

Supported Row LCR Column Attributes

Statements in statement DML handlers can contain the row LCR column attributes described in the following table.
Table 168-1  Row LCR Column Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new</td>
<td>Returns the new column value in a row LCR. If the new value does not exist, then this attribute returns the old value.</td>
</tr>
<tr>
<td>new_exists</td>
<td>Returns <strong>TRUE</strong> if a new column value exists in a row LCR. Returns <strong>FALSE</strong> if a new column value does not exist in a row LCR.</td>
</tr>
<tr>
<td>new_only</td>
<td>Returns the new column value in a row LCR. If the new value does not exist, then this attribute returns <strong>NULL</strong> and does not return the old column value.</td>
</tr>
<tr>
<td>old</td>
<td>Returns the old column value in a row LCR.</td>
</tr>
<tr>
<td>old_exists</td>
<td>Returns <strong>TRUE</strong> if an old column value exists in a row LCR. Returns <strong>FALSE</strong> if an old column value does not exist in a row LCR.</td>
</tr>
</tbody>
</table>

Specify these attributes in the following way in a statement:

```
:attribute.column_name
```

For example, to specify the **new_only** attribute for the **salary** column, enter the following in a statement:

```
:new_only.salary
```

Supported Row LCR Attributes

Statements in statement DML handlers can contain the row LCR attributes described in the following table.

Table 168-2  Row LCR Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_type</td>
<td>Returns the type of DML statement that produced the change, either INSERT, UPDATE, or DELETE. DBMS_LOB piecewise LOB operations are not supported by statement DML handlers.</td>
</tr>
<tr>
<td>commit_scn</td>
<td>Returns the commit system change number (SCN) of the transaction to which the LCR belongs.</td>
</tr>
<tr>
<td>compatible</td>
<td>Returns the minimal database compatibility required to support the LCR.</td>
</tr>
<tr>
<td>instance_number</td>
<td>Returns the instance number of the database instance that made the change that is encapsulated in the LCR. Typically, the instance number is relevant in an Oracle Real Application Clusters (Oracle RAC) configuration.</td>
</tr>
<tr>
<td>object_owner</td>
<td>Returns the schema name that contains the table with the changed row.</td>
</tr>
<tr>
<td>object_name</td>
<td>Returns the name of the table that contains the changed row.</td>
</tr>
<tr>
<td>scn</td>
<td>Returns the SCN at the time when the change was made.</td>
</tr>
<tr>
<td>source_database_name</td>
<td>Returns the name of the source database where the row change occurred.</td>
</tr>
</tbody>
</table>
Table 168-2  (Cont.) Row LCR Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_time</td>
<td>Returns the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created.</td>
</tr>
<tr>
<td>tag</td>
<td>Returns a raw tag that can be used to track the LCR.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>Returns the identifier of the transaction in which the DML statement was run.</td>
</tr>
</tbody>
</table>

Specify these attributes in the following way in a statement:

`:attribute_name`

For example, to specify the `source_database_name` attribute for a row LCR, enter the following in a statement:

`:source_database_name`

Supported Row LCR Extra Attributes

Statements in statement DML handlers can contain the row LCR extra attributes described in the following table.

Table 168-3  Row LCR Extra Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>Returns the rowid of the row changed in a row LCR.</td>
</tr>
<tr>
<td>serial#</td>
<td>Returns the serial number of the session that performed the change captured in the LCR.</td>
</tr>
<tr>
<td>session#</td>
<td>Returns the identifier of the session that performed the change captured in the LCR.</td>
</tr>
<tr>
<td>thread#</td>
<td>Returns the thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle RAC configuration.</td>
</tr>
<tr>
<td>tx_name</td>
<td>Returns the name of the transaction that includes the LCR.</td>
</tr>
<tr>
<td>username</td>
<td>Returns the name of the current user who performed the change captured in the LCR.</td>
</tr>
</tbody>
</table>

Specify these attributes in the following way in a statement:

`:extra_attribute.attribute_name`

For example, to specify the `row_id` extra attribute for a row LCR, enter the following in a statement:

`:extra_attribute.row_id`

Supported Row LCR Method

A statement in a statement DML handler can include a call to the `EXECUTE` member procedure for row LCRs. The `EXECUTE` member procedure executes the row LCR under the security domain of the current user.
A statement that runs the `EXECUTE` member procedure can be placed anywhere in the execution sequence order of the statement DML handler. It is not necessary to execute a row LCR unless the goal is to apply the changes in the row LCR to a table in addition to performing any other SQL statements in the statement DML handler.

When you call the `EXECUTE` member procedure in a statement, the `conflict_resolution` parameter controls whether any conflict resolution defined for the table using the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package is used to resolve conflicts resulting from the execution of the LCR:

```plaintext
:lcr.execute TRUE|FALSE
```

A `TRUE` argument indicates that conflict resolution is used. A `FALSE` argument indicates that conflict resolution is not used.

For example, to use conflict resolution, enter the following in a statement:

```
:lcr.execute TRUE
```

An error is raised if this parameter is not specified or is set to `NULL`.

---

### See Also:

- "SET_UPDATE_CONFLICT_HANDLER Procedure"
- *Oracle Streams Replication Administrator's Guide*

---

### 168.5 Summary of DBMS_STREAMS_HANDLER_ADM Subprograms

This table lists the `DBMS_STREAMS_HANDLER_ADM` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_STMT_TO_HANDLER Procedure</td>
<td>Adds a statement to a statement DML handler</td>
</tr>
<tr>
<td>CREATE_STMT_HANDLER Procedure</td>
<td>Creates a statement DML handler</td>
</tr>
<tr>
<td>DROP_STMT_HANDLER Procedure</td>
<td>Drops a statement DML handler</td>
</tr>
<tr>
<td>REMOVE_STMT_FROM_HANDLER Procedure</td>
<td>Removes a statement from a statement DML handler</td>
</tr>
</tbody>
</table>

---

**Note:**

The subprograms in this package do not commit.
### 168.5.1 ADD_STMT_TO_HANDLER Procedure

This procedure adds a statement to a statement DML handler.

**Syntax**

```sql
DBMS_STREAMS_HANDLER_ADM.ADD_STMT_TO_HANDLER(
    handler_name       IN VARCHAR2,
    statement          IN CLOB,  
    execution_sequence IN NUMBER  DEFAULT NULL);
```

**Parameters**

**Table 168-5** ADD_STMT_TO_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler_name</td>
<td>The name of the statement DML handler.</td>
</tr>
<tr>
<td>statement</td>
<td>The text of the SQL statement to add to the statement DML handler. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>execution_sequence</td>
<td>The position of the statement in the statement DML handler at which a SQL statement is to be set to execute. Statements are executed in order from the lowest execution sequence number to the highest execution sequence number. You can specify a positive or negative integer or decimal, or you can specify 0 (zero). If you specify an execution sequence number that is used by an existing statement in the statement DML handler, then the statement in the statement parameter replaces the existing statement. If NULL, then the statement is added to the statement DML handler with an execution sequence number that is larger than the execution sequence number for any statement in the statement DML handler.</td>
</tr>
</tbody>
</table>

### 168.5.2 CREATE_STMT_HANDLER Procedure

This procedure creates a statement DML handler.

**Syntax**

```sql
DBMS_STREAMS_HANDLER_ADM.CREATE_STMT_HANDLER(
    handler_name IN VARCHAR2,  
    comment      IN VARCHAR2  DEFAULT NULL);
```

**Parameters**

**Table 168-6** CREATE_STMT_HANDLER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler_name</td>
<td>The name of the statement DML handler.</td>
</tr>
<tr>
<td>comment</td>
<td>A comment for the statement DML handler. If NULL, then no comment is recorded for the statement DML handler.</td>
</tr>
</tbody>
</table>
168.5.3 DROP_STMT_HANDLER Procedure

This procedure drops a statement DML handler.

Syntax

```sql
DBMS_STREAMS_HANDLER_ADM.DROP_STMT_HANDLER(
    handler_name IN VARCHAR2);
```

Parameters

**Table 168-7  DROP_STMT_HANDLER Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler_name</td>
<td>The name of the statement DML handler.</td>
</tr>
</tbody>
</table>

168.5.4 REMOVE_STMT_FROM_HANDLER Procedure

This procedure removes a statement from a statement DML handler.

Syntax

```sql
DBMS_STREAMS_HANDLER_ADM.REMOVE_STMT_FROM_HANDLER(
    handler_name IN VARCHAR2,
    execution_sequence IN NUMBER DEFAULT NULL);
```

Parameters

**Table 168-8  REMOVE_STMT_FROM_HANDLER Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler_name</td>
<td>The name of the statement DML handler.</td>
</tr>
<tr>
<td>execution_sequence</td>
<td>The position of the statement to remove. You can specify a positive or negative integer or decimal, or you can specify 0 (zero). If NULL, the procedure removes the last statement in the statement DML handler. If the specified execution sequence number does not exist for the statement DML handler, then the procedure raises an error.</td>
</tr>
</tbody>
</table>

The DBMS_STREAMS_MESSAGING package, one of a set of Oracle Streams packages, provides interfaces to enqueue messages into and dequeue messages from a ANYDATA queue.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_STREAMS_MESSAGING Subprograms

169.1 DBMS_STREAMS_MESSAGING Overview

This package provides interfaces to enqueue messages into and dequeue messages from a ANYDATA queue.

**Note:**
Currently, messaging clients cannot dequeue buffered messages. In addition, the DBMS_STREAMS_MESSAGING package cannot be used to enqueue messages into or dequeue messages from a buffered queue. However, you can use the DBMS_AQ package to enqueue and dequeue buffered messages.

**See Also:**
Oracle Database Advanced Queuing User's Guide for more information about queues, messaging, and the DBMS_AQ package

169.2 DBMS_STREAMS_MESSAGING Security Model

Security on this package can be controlled by either granting EXECUTE on this package to selected users or roles, or by granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.
169.3 Summary of DBMS_STREAMS_MESSAGING Subprograms

This table lists the DBMS_STREAMS_MESSAGING subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEQUEUE Procedure</td>
<td>Uses the specified Oracle Streams messaging client to dequeue a message from the specified queue</td>
</tr>
<tr>
<td>ENQUEUE Procedure</td>
<td>The current user enqueues a message into the specified queue</td>
</tr>
</tbody>
</table>

**Note:**

The subprograms in this package do not commit.

169.3.1 DEQUEUE Procedure

This procedure uses the specified Oracle Streams messaging client to dequeue a message from the specified queue.

This procedure is overloaded. One version of this procedure contains the `msgid OUT` parameter, and the other does not.

**Syntax**

```sql
DBMS_STREAMS_MESSAGING.DEQUEUE(
    queue_name    IN   VARCHAR2,
    streams_name  IN   VARCHAR2,
    payload       OUT  ANYDATA,
    dequeue_mode  IN   VARCHAR2        DEFAULT 'REMOVE',
    navigation    IN   VARCHAR2        DEFAULT 'NEXT MESSAGE',
    wait          IN   BINARY_INTEGER  DEFAULT FOREVER,
    msgid         OUT  RAW);

DBMS_STREAMS_MESSAGING.DEQUEUE(
    queue_name    IN   VARCHAR2,
    streams_name  IN   VARCHAR2,
    payload       OUT  ANYDATA,
    dequeue_mode  IN   VARCHAR2        DEFAULT 'REMOVE',
    navigation    IN   VARCHAR2        DEFAULT 'NEXT MESSAGE',
    wait          IN   BINARY_INTEGER  DEFAULT FOREVER,
    msgid         OUT  RAW);
```
Parameters

Table 169-2  DEQUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the local queue from which messages will be dequeued, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be a secure queue of ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the Oracle Streams messaging client. For example, if the user strmadmin is the messaging client, then specify strmadmin. If NULL and a relevant messaging client for the queue exists, then the procedure uses the relevant messaging client. If NULL and multiple relevant messaging clients for the queue exist, then the procedure raises an error.</td>
</tr>
<tr>
<td>payload</td>
<td>The payload that is dequeued</td>
</tr>
<tr>
<td>dequeue_mode</td>
<td>Specify one of the following settings:</td>
</tr>
<tr>
<td></td>
<td>REMOVE: Read the message and delete it. This setting is the default. The message can be retained in the queue table based on the retention properties.</td>
</tr>
<tr>
<td></td>
<td>LOCKED: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This setting is equivalent to a select for update statement.</td>
</tr>
<tr>
<td></td>
<td>BROWSE: Read the message without acquiring any lock on the message. This specification is equivalent to a select statement.</td>
</tr>
<tr>
<td>navigation</td>
<td>The position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. Specify one of the following settings:</td>
</tr>
<tr>
<td></td>
<td>NEXT MESSAGE: Retrieve the next message that is available and matches the search criteria. If the previous message belongs to a message group, then retrieve the next available message that matches the search criteria and belongs to the message group. This setting is the default.</td>
</tr>
<tr>
<td></td>
<td>NEXT TRANSACTION: Skip the remainder of the current message group (if any) and retrieve the first message of the next message group. This setting can only be used if message grouping is enabled for the current queue.</td>
</tr>
<tr>
<td></td>
<td>FIRST MESSAGE: Retrieves the first message which is available and matches the search criteria. This setting resets the position to the beginning of the queue.</td>
</tr>
<tr>
<td></td>
<td>Note: Each message group contains the messages in a single transaction. See Also: Oracle Database Advanced Queuing User's Guide for more information about dequeue options</td>
</tr>
<tr>
<td>wait</td>
<td>Either FOREVER or NO_WAIT</td>
</tr>
<tr>
<td></td>
<td>If FOREVER, then the dequeue call is blocked without a time out until a message is available in the queue.</td>
</tr>
<tr>
<td></td>
<td>If NO_WAIT, then a wait time of zero seconds is used. In this case, the dequeue will return immediately even if there are no messages in the queue.</td>
</tr>
</tbody>
</table>
### Table 169-2 (Cont.) DEQUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgid</td>
<td>The message identifier of the message that is dequeued</td>
</tr>
</tbody>
</table>

**Exceptions**

### Table 169-3 DEQUEUE Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENDOFCURTRANS</td>
<td>Dequeue has reached the end of the messages in the current transaction. Specify this exception in the following way: SYS.DBMS_STREAMS_MESSAGING.ENDOFCURTRANS Every dequeue procedure should include an exception handler that handles this exception.</td>
</tr>
<tr>
<td>NOMOREMSGS</td>
<td>There are no more messages in the queue for the dequeue operation. Specify this exception in the following way: SYS.DBMS_STREAMS_MESSAGING.NOMOREMSGS A dequeue procedure that specifies NO_WAIT for the wait parameter should include an exception handler that handles this exception.</td>
</tr>
</tbody>
</table>

### 169.3.2 ENQUEUE Procedure

This procedure enables the current user to enqueue a message into the specified queue.

This procedure is overloaded. One version of this procedure contains the msgid OUT parameter, and the other does not.

**Syntax**

```sql
DBMS_STREAMS_MESSAGING.ENQUEUE(
    queue_name  IN   VARCHAR2,
    payload     IN   ANYDATA,
    msgid       OUT  RAW);

DBMS_STREAMS_MESSAGING.ENQUEUE(
    queue_name  IN   VARCHAR2,
    payload     IN   ANYDATA);
```
Parameters

Table 169-4 ENQUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the local queue into which messages will be enqueued, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be a secure queue of ANYDATA type. For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>payload</td>
<td>The payload that is enqueued</td>
</tr>
<tr>
<td>msgid</td>
<td>The message identifier of the message that is enqueued</td>
</tr>
</tbody>
</table>

Usage Notes

To successfully enqueue messages into a queue, the current user must be mapped to a unique Advanced Queuing agent with the same name as the current user. You can run the DBMS_STREAMS_ADM.SET_UP_QUEUE procedure and specify a user as the queue user to grant the necessary privileges to the user to perform enqueues. The Advanced Queuing agent is created automatically when you run SET_UP_QUEUE and specify a queue user.

See Also:

SET_UP_QUEUE Procedure
The `DBMS_STREAMS_TABLESPACE_ADM` package, one of a set of Oracle Streams packages, provides administrative interfaces for copying tablespaces between databases and moving tablespaces from one database to another.

This package uses transportable tablespaces, Data Pump, the `DBMS_FILE_TRANSFER` package, and the `DBMS_FILE_GROUP` package.

This chapter contains the following topics:

- Overview
- Security Model
- Data Structures
- Summary of `DBMS_STREAMS_TABLESPACE_ADM` Subprograms

**See Also:**

Oracle Streams Concepts and Administration and Oracle Streams Replication Administrator's Guide for more information about this package and Oracle Streams

### 170.1 DBMS_STREAMS_TABLESPACE_ADM Overview

Either a simple tablespace or a self-contained tablespace set must be specified in each procedure in this package.

A **self-contained tablespace** has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. A **simple tablespace** is a self-contained tablespace that uses only one datafile.

A simple tablespace must be specified in the following procedures:

- `ATTACH_SIMPLE_TABLESPACE` Procedure
- `CLONE_SIMPLE_TABLESPACE` Procedure
- `DETACH_SIMPLE_TABLESPACE` Procedure
- `PULL_SIMPLE_TABLESPACE` Procedure

A **self-contained tablespace set** has no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

A self-contained tablespace set must be specified in the following procedures:

- `ATTACH_TABLESPACES` Procedure
• CLONE_TABLESPACES Procedure
•_DETACH_TABLESPACES Procedure
• PULL_TABLESPACES Procedure

To determine whether a set of tablespaces is self-contained, use the TRANSPORT_SET_CHECK procedure in the Oracle supplied package DBMS_TTS.

See Also:
Oracle Database Administrator's Guide for more information about self-contained tablespaces and tablespace sets

170.2 DBMS_STREAMS_TABLESPACE_ADM Security Model

Security on this package can be controlled by either granting EXECUTE on this package to selected users or roles, or by granting EXECUTE_CATALOG_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

170.3 DBMS_STREAMS_TABLESPACE_ADM Data Structures

The DBMS_STREAMS_TABLESPACE_ADM package defines RECORD types and TABLE types.

DIRECTORY_OBJECT_SET Table Type
Contains the names of one or more directory objects. Each name must be a directory object created using the SQL statement CREATE DIRECTORY.

Syntax

TYPE DIRECTORY_OBJECT_SET IS TABLE OF VARCHAR2(32)
  INDEX BY BINARY_INTEGER;

FILE Record Type
Contains the directory object associated with a directory and the name of the file in the directory.

Syntax

TYPE FILE_SET IS TABLE OF FILE
  INDEX BY BINARY_INTEGER;

Fields
Table 170-1  FILE Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory_object</td>
<td>The name of a directory object. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file in the corresponding directory associated with the directory object</td>
</tr>
</tbody>
</table>

FILE_SET Table Type
Contains one or more files.

Syntax

```
TYPE FILE_SET IS TABLE OF FILE
  INDEX BY BINARY_INTEGER;
```

TABLESPACE_SET Table Type
Contains the names of one or more tablespaces.

Syntax

```
TYPE TABLESPACE_SET IS TABLE OF VARCHAR2(32)
  INDEX BY BINARY_INTEGER;
```

170.3.1 DIRECTORY_OBJECT_SET Table Type
Contains the names of one or more directory objects. Each name must be a directory object created using the SQL statement CREATE DIRECTORY.

170.3.2 FILE Record Type
This record type contains the directory object associated with a directory and the name of the file in the directory.

170.3.3 FILE_SET Table Type
This table type contains one or more files.

170.3.4 TABLESPACE_SET Table Type
This type contains the names of one or more tablespaces.

170.4 Summary of DBMS_STREAMS_TABLESPACE_ADM Subprograms
This table lists the DBMS_STREAMS_TABLESPACE_ADM subprograms and briefly describes them.
Table 170-2  DBMS_STREAMS_TABLESPACE_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACH_SIMPLE_TABLESPACE</td>
<td>Uses Data Pump to import a simple tablespace previously exported using the</td>
</tr>
<tr>
<td>Procedure</td>
<td>DBMS_STREAMS_TABLESPACE_ADM package or Data Pump export</td>
</tr>
<tr>
<td>ATTACH_TABLESPACES Procedure</td>
<td>Uses Data Pump to import a self-contained tablespace set previously exported</td>
</tr>
<tr>
<td></td>
<td>using the DBMS_STREAMS_TABLESPACE_ADM package, Data Pump export, or the</td>
</tr>
<tr>
<td></td>
<td>Recovery Manager (RMAN) TRANSPORT TABLESPACE command</td>
</tr>
<tr>
<td>CLONE_SIMPLE_TABLESPACE</td>
<td>Clones a simple tablespace. The tablespace can later be attached to a</td>
</tr>
<tr>
<td>Procedure</td>
<td>database.</td>
</tr>
<tr>
<td>CLONE_TABLESPACES Procedure</td>
<td>Clones a set of self-contained tablespaces. The tablespaces can later be</td>
</tr>
<tr>
<td></td>
<td>attached to a database.</td>
</tr>
<tr>
<td>DETACH_SIMPLE_TABLESPACE</td>
<td>Detaches a simple tablespace. The tablespace can later be attached to a</td>
</tr>
<tr>
<td>Procedure</td>
<td>database.</td>
</tr>
<tr>
<td>DETACH_TABLESPACES Procedure</td>
<td>Detaches a set of self-contained tablespaces. The tablespaces can later be</td>
</tr>
<tr>
<td></td>
<td>attached to a database.</td>
</tr>
<tr>
<td>PULL_SIMPLE_TABLESPACE</td>
<td>Copies a simple tablespace from a remote database and attaches it to the</td>
</tr>
<tr>
<td>Procedure</td>
<td>current database</td>
</tr>
<tr>
<td>PULL_TABLESPACES Procedure</td>
<td>Copies a set of self-contained tablespaces from a remote database and</td>
</tr>
<tr>
<td></td>
<td>attaches the tablespaces to the current database</td>
</tr>
</tbody>
</table>

Note:

All subprograms commit unless specified otherwise.

170.4.1 ATTACH_SIMPLE_TABLESPACE Procedure

This procedure uses Data Pump to import a simple tablespace previously exported using the DBMS_STREAMS_TABLESPACE_ADM package, Data Pump export, or the Recovery Manager (RMAN) TRANSPORT TABLESPACE command.

Syntax

```sql
DBMS_STREAMS_TABLESPACE_ADM.ATTACH_SIMPLE_TABLESPACE (  
directory_object IN VARCHAR2,  
tablespace_file_name IN VARCHAR2,  
converted_file_name IN VARCHAR2 DEFAULT NULL,  
datafile_platform IN VARCHAR2 DEFAULT NULL,  
tablespace_name OUT VARCHAR2);
```
Parameters

Table 170-3 ATTACH_SIMPLE_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory_object</td>
<td>The directory that contains the Data Pump dump file and the datafile for the tablespace. You must specify the name of a directory object created using the SQL statement <code>CREATE DIRECTORY</code>. The name of the Data Pump export dump file must be the same as the data file name for the tablespace, except with a <code>.dmp</code> extension. If the <code>converted_file_name</code> is non-NULL, specify the dump file produced by the export database, not the file name after conversion. The Data Pump import log file is written to this directory. The name of the log file is the same as the data file name for the tablespace, except with an <code>.alg</code> extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>tablespace_file_name</td>
<td>The name of the datafile for the tablespace being imported. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>converted_file_name</td>
<td>If the <code>datafile_platform</code> parameter is non-NULL and is different from the platform of the local import database, then specify a file name for the converted datafile. The datafile is converted to the platform of the local import database and copied to the new file name. The existing datafile is not modified nor deleted. If non-NULL and the <code>datafile_platform</code> parameter is NULL, then the procedure ignores this parameter. If non-NULL and the <code>datafile_platform</code> parameter specifies the same platform as the local import database, then the procedure ignores this parameter. If NULL and the <code>datafile_platform</code> parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>datafile_platform</td>
<td>Specify NULL if the platform is the same for the export database and the current import database. Specify the platform for the export database if the platform is different for the export database and the import database. You can determine the platform of a database by querying the <code>PLATFORM_NAME</code> column in the <code>V$DATABASE</code> dynamic performance view. The <code>V$TRANSPORTABLE_PLATFORM</code> dynamic performance view lists all platforms that support cross-platform transportable tablespaces.</td>
</tr>
<tr>
<td>tablespace_name</td>
<td>Contains the name of the attached tablespace. The attached tablespace is read-only. Use an <code>ALTER TABLESPACE</code> statement to make the tablespace read/write if necessary.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet the following requirements:

- Have `IMP_FULL_DATABASE` role
- Have `READ` and `WRITE` privilege on the directory object that contains the Data Pump export dump file and the datafiles for the tablespaces in the set, specified by the `directory_object` parameter
Automatic Storage Management (ASM) directories cannot be used with this procedure.

See Also:
- Overview

### 170.4.2 ATTACH_TABLESPACES Procedure

This procedure uses Data Pump to import a self-contained tablespace set previously exported using the `DBMS_STREAMS_TABLESPACE_ADM` package, Data Pump export, or the Recovery Manager (RMAN) `TRANSPORT TABLESPACE` command.

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the `datapump_job_name` parameter. This job performs the Data Pump import to complete the attach operation. In addition, if the platform at the export database is different than the local database platform, then this procedure optionally can create datafiles for the tablespace set that can be used with the local platform.

- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump import is performed. This version of the procedure uses the files in a file group version and can copy the export dump file, export log file, and the datafiles that comprise the tablespace set into the specified directories. The file group and version are specified using the `file_group_name` and `version_name` parameters, respectively. This version of the procedure does not require a datafiles platform specification if the platform at the export database is different than the local database platform. Instead, the tablespace set is migrated automatically to the correct platform when it is attached.

#### Syntax

```sql
DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
    datapump_job_name          IN OUT VARCHAR2,
    dump_file                  IN     FILE,
    tablespace_files           IN     FILE_SET,
    converted_files            IN     FILE_SET,
    datafiles_platform         IN     VARCHAR2  DEFAULT NULL,
    log_file                   IN     FILE      DEFAULT NULL,
    tablespace_names           OUT    TABLESPACE_SET);

DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
    file_group_name            IN   VARCHAR2,
    version_name               IN   VARCHAR2 DEFAULT NULL,
    datafiles_directory_object IN   VARCHAR2 DEFAULT NULL,
    logfile_directory_object   IN   VARCHAR2 DEFAULT NULL,
    repository_db_link         IN   VARCHAR2 DEFAULT NULL,
    tablespace_names           OUT  TABLESPACE_SET);
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pump_job_name</td>
<td>The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily. If NULL, then the system generates a Data Pump job name.</td>
</tr>
<tr>
<td>dump_file</td>
<td>The file name of the Data Pump dump file to import.</td>
</tr>
<tr>
<td></td>
<td>If NULL or if a file attribute is NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>tablespace_files</td>
<td>The file set that contains the datafiles for the tablespace set being imported.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>converted_files</td>
<td>If the <code>datafiles_platform</code> parameter is non-NULL and is different from the platform for the local import database, then specify a file set with the names of the converted datafiles. The datafiles are converted to the platform of the local import database and copied to the new file names. In this case, the number of files in the specified file set must match the number of files in the file set specified for the <code>tablespace_files</code> parameter. The existing datafiles are not modified nor deleted. If non-NULL and the <code>datafiles_platform</code> parameter is NULL, then the procedure ignores this parameter. If non-NULL and the <code>datafiles_platform</code> parameter specifies the same platform as the local import database, then the procedure ignores this parameter. If NULL and the <code>datafiles_platform</code> parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>datafiles_platform</td>
<td>Specify NULL if the platform is the same for the export database and the current import database. Specify the platform for the export database if the platform is different for the export database and the import database. You can determine the platform of a database by querying the <code>PLATFORM_NAME</code> column in the <code>V$DATABASE</code> dynamic performance view. The <code>V$TRANSPORTABLE_PLATFORM</code> dynamic performance view lists all platforms that support cross-platform transportable tablespaces.</td>
</tr>
<tr>
<td>log_file</td>
<td>Specify the log file name for the Data Pump import.</td>
</tr>
<tr>
<td></td>
<td>If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension <code>.alg</code> and places it in the Data Pump export dump file directory. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</td>
</tr>
<tr>
<td>file_group_name</td>
<td>The name of the file group, specified as <code>[schema_name.]file_group_name</code>. For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales</code>, then specify <code>hq_dba.sales</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
Table 170-4  (Cont.) ATTACH_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version_name</td>
<td>The name of the file group version to attach. If NULL, then the procedure uses the version with the latest creation time for the file group.</td>
</tr>
<tr>
<td>datafiles_directory_object</td>
<td>The directory object into which the datafiles and Data Pump export dump file are copied. The files are copied from the tablespace repository directories to this directory. If non-NULL, the attached tablespaces use the files in specified directory. However, the file group version specified in the version_name parameter consists of the files in the original directory, not in the directory specified by this datafiles_directory_object parameter. If NULL, then the procedure does not copy the datafiles and dump file.</td>
</tr>
<tr>
<td>logfile_directory_object</td>
<td>The directory object into which the Data Pump import log file is placed. The system generates a log file name with the extension .alg. If NULL, then the procedure places the import log file in the same directory as the dump file.</td>
</tr>
<tr>
<td>repository_db_link</td>
<td>If the file group is in a different database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure. If this parameter is non-NULL, then meet the following requirements: Each directory object that contains files in the version being attached must exist on both databases. The corresponding directory objects must have the same names on both databases. If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.</td>
</tr>
<tr>
<td>tablespace_names</td>
<td>Contains the names of the attached tablespaces. The attached tablespaces are read-only. Use ALTER TABLESPACE statements to make the tablespaces read/write if necessary.</td>
</tr>
</tbody>
</table>

Usage Notes

The following sections contain usage notes for this procedure:

- User Requirements
- Procedures Used to Clone or Detach a Tablespace Set
- When the Attach Database Is Different Than the Clone or Detach Database
- Automatic Storage Management Directories
User Requirements

To run either version of this procedure, a user must meet the following requirements:

- Have `IMP_FULL_DATABASE` role
- Have `READ` and `WRITE` privilege on the directory objects that contain the Data Pump export dump file and the datafiles for the tablespaces in the set, specified by the `dump_file` and `tablespace_files` parameters, or by the `datafiles_directory_object` parameter
- Have `WRITE` privilege on the directory object that will hold the Data Pump import log file, specified by the `log_file` parameter or `logfile_directory_object` parameter if it is non-NULL

If the Data Pump job version of the procedure is run, then the user must have `WRITE` privilege on the directory objects that will hold the converted datafiles for the tablespaces in the set if platform conversion is necessary. These directory objects are specified by the `converted_files` parameter if it is non-NULL.

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.

Procedures Used to Clone or Detach a Tablespace Set

After a tablespace set is cloned or detached using the `CLONE_TABLESPACES` or `DETACH_TABLESPACES` procedure, respectively, the tablespace set can be attached to a database using the `ATTACH_TABLESPACES` procedure. If the Data Pump job version of the `CLONE_TABLESPACES` or `DETACH_TABLESPACES` procedure was used, then use the Data Pump job version of the `ATTACH_TABLESPACES` procedure. If the file group version of the `CLONE_TABLESPACES` or `DETACH_TABLESPACES` procedure was used, then use the file group version of the `ATTACH_TABLESPACES` procedure.

When the Attach Database Is Different Than the Clone or Detach Database

You can attach a tablespace set to a different database than the database from which the tablespace set was cloned or detached. The two databases might or might not share a file system. If the two databases do not share a file system, then you must transfer the dump file and datafiles to the remote system using the `DBMS_FILE_TRANSFER` package, FTP, or some other method. You can attach the tablespace set in one of the following ways depending on the version of the `ATTACH_TABLESPACES` procedure you use:
• If you use the Data Pump job version of the procedure, then specify the relevant files on the file system. The directory object names can be different in the databases.

• If you use the file group version of the procedure, then you can use the repository_db_link parameter to specify the database where tablespace repository resides. The directory objects for the files must exist and must match in the databases.

See Also:
• CLONE_TABLESPACES Procedure
• DETACH_TABLESPACES Procedure
• DBMS_FILE_GROUP for more information about file groups

Automatic Storage Management Directories

Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

See Also:
Oracle Database Utilities for information about specifying ASM directories for directory objects

170.4.3 CLONE_SIMPLE_TABLESPACE Procedure

This procedure clones a simple tablespace. The specified tablespace must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace and places the dump file in the specified directory
3. Places the datafile for the specified tablespace in the specified directory
4. If this procedure made the tablespace read-only, then makes the tablespace read/write

In addition, this procedure optionally can create a datafile for the tablespace that can be used with a platform that is different than the local database platform.

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.CLONE_SIMPLE_TABLESPACE(
    tablespace_name IN VARCHAR2,
    directory_object IN VARCHAR2,
    destination_platform IN VARCHAR2 DEFAULT NULL,
    tablespace_file_name OUT VARCHAR2);
```
Parameters

Table 170-5  CLONE_SIMPLE_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablespace_name</td>
<td>The tablespace to be cloned. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>directory_object</td>
<td>The directory where the Data Pump export dump file, the Data Pump export log file, and the datafile for the tablespace are placed. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY.</td>
</tr>
<tr>
<td></td>
<td>The name of the Data Pump export dump file is the same as the data file name for the tablespace, except with a .dmp extension. If a file exists with the same name as the dump file in the directory, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>The name of the log file is the same as the data file name for the tablespace, except with a .clg extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>destination_platform</td>
<td>Specify NULL if the platform is the same for the current export database and the intended import database. Specify the platform for the intended import database if the platform is different for the export database and the import database.</td>
</tr>
<tr>
<td></td>
<td>You can determine the platform of a database by querying the PLATFORM_NAME column in the V$DATABASE dynamic performance view. The V$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.</td>
</tr>
<tr>
<td>tablespace_file_name</td>
<td>Contains the name of the cloned tablespace datafile. This datafile is placed in the directory specified by the parameter directory_object.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet the following requirements:

- Have **EXP_FULL_DATABASE role**
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include **DBA_TABLESPACES** and **USER_TABLESPACES**.
- Have **MANAGE TABLESPACE** or **ALTER TABLESPACE** on a tablescape if the tablescape must be made read-only
- Have **READ** privilege on the directory object for the directory that contains the datafile for the tablespace. The name of this tablespace is specified by the **tablespace_name** parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure.
- Have **READ** and **WRITE** privilege on the directory object that will contain the Data Pump export dump file, specified by the **directory_object** parameter
- If the file group version of the procedure is run, then the user must have the necessary privileges to manage file group.
After cloning a tablespace using this procedure, you can add the tablespace to a different database using the ATTACH_SIMPLE_TABLESPACE procedure. If the database is a remote database and you want to use the ATTACH_SIMPLE_TABLESPACE procedure, then you can transfer the dump file and datafile to the remote system using the DBMS_FILE_TRANSFER package, FTP, or some other method.

Automatic Storage Management (ASM) directories cannot be used with this procedure.

### See Also:

- Overview
- ATTACH_SIMPLE_TABLESPACE Procedure and PULL_SIMPLE_TABLESPACE Procedure

### 170.4.4 CLONE_TABLESPACES Procedure

This procedure clones a set of self-contained tablespaces. All of the tablespaces in the specified tablespace set must be online.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set read-only
2. Uses Data Pump to export the metadata for the tablespaces in the tablespace set and places the dump file in the specified directory
3. Places the datafiles that comprise the specified tablespace set in the specified directory
4. If this procedure made a tablespace read-only, then makes the tablespace read/write

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the datapump_job_name parameter. This job performs the Data Pump export. This version of the procedure completes the clone operation by placing the export dump file, export log file, and the datafiles that comprise the tablespace set in the specified directories, but the files are not added to a file group version. In addition, this version of the procedure optionally can create datafiles for the tablespace set that can be used with a platform that is different than the local database platform.

- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump export is performed, and this version of the procedure completes the clone operation by placing the export dump file, export log file, and the datafiles that comprise the tablespace set in the appropriate file group version. The file group and version are specified using the file_group_name and version_name parameters, respectively. This version of the procedure does not require a destination platform specification if the destination platform is different. Instead, the tablespace set is migrated automatically to the correct
platform when it is attached at the destination database using the file group version of the ATTACH_TABLESPACES procedure.

Syntax

DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES(
  datapump_job_name            IN OUT VARCHAR2,
  tablespace_names             IN     TABLESPACE_SET,
  dump_file                    IN     FILE,
  tablespace_directory_objects IN     DIRECTORY_OBJECT_SET,
  destination_platform         IN     VARCHAR2  DEFAULT NULL,
  log_file                     IN     FILE      DEFAULT NULL,
  tablespace_files             OUT    FILE_SET);

DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES(
  tablespace_names             IN  TABLESPACE_SET,
  tablespace_directory_object  IN  VARCHAR2  DEFAULT NULL,
  log_file_directory_object    IN  VARCHAR2  DEFAULT NULL,
  file_group_name              IN  VARCHAR2,
  version_name                 IN  VARCHAR2  DEFAULT NULL,
  repository_db_link           IN  VARCHAR2  DEFAULT NULL);

Parameters

Table 170-6  CLONE_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pump_job_name</td>
<td>The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the system generates a Data Pump job name.</td>
</tr>
<tr>
<td>tablespace_names</td>
<td>The tablespace set to be cloned.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>dump_file</td>
<td>The file name of the Data Pump dump file that is exported.</td>
</tr>
<tr>
<td></td>
<td>If NULL or if a file attribute is NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If the specified file exists, then the procedure raises an error.</td>
</tr>
<tr>
<td>tablespace_directory_objects</td>
<td>The set of directory objects into which the datafiles for the tablespaces are copied. If multiple directory objects are in the set, then the procedure copies a datafile to each directory object in the set in sequence. In this case, if the end of the directory object set is reached, then datafile copying starts again with the first directory object in the set.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure copies datafiles for the tablespace set to the dump file directory.</td>
</tr>
</tbody>
</table>
### Table 170-6  (Cont.) CLONE_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_platform</td>
<td>Specify NULL if the platform is the same for the current export database and the intended import database. Specify the platform for the intended import database if the platform is different for the export database and the import database. You can determine the platform of a database by querying the PLATFORM_NAME column in the V$DATABASE dynamic performance view. The V$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.</td>
</tr>
<tr>
<td>log_file</td>
<td>Specify the log file name for the Data Pump export. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .clg and places it in the dump file directory. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</td>
</tr>
<tr>
<td>tablespace_directory_object</td>
<td>The directory object into which the data files are copied and Data Pump export dump file is placed. The system generates a dump file name with the extension .dmp. If NULL, then the procedure copies the datafiles to and places the dump file in the default directory object for the version. If the version does not have a default directory object, then the procedure uses the default directory object for the file group. If NULL and no default directory object exists for the version or file group, then the procedure raises an error.</td>
</tr>
<tr>
<td>log_file_directory_object</td>
<td>The directory object into which the Data Pump export log file is placed. The system generates a log file name with the extension .clg. If NULL, then the procedure uses the directory object specified in tablespace_directory_object.</td>
</tr>
<tr>
<td>file_group_name</td>
<td>The name of the file group, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales, then specify hq_dba.sales. If the schema is not specified, then the current user is the default. If the specified file group does not exist, then the procedure creates it.</td>
</tr>
<tr>
<td>version_name</td>
<td>The name of the version into which the cloned tablespace set is placed. The specified version name cannot be a positive integer. If the specified version does not exist, then the procedure creates it. If the specified version exists, then the procedure adds the tablespace set to the version. Only one Data Pump export dump file can exist in a version. If NULL, then the procedure creates a new version, and the version number can be used to manage the version.</td>
</tr>
</tbody>
</table>
Table 170-6  (Cont.) CLONE_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repository_db_link</td>
<td>If the file group is in a remote database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure. If this parameter is non-NULL, then the directory object specified in tablespace_directory_object must exist on the local database and on the remote database. If tablespace_directory_object is NULL, then the default directory object must exist on both databases. The directory object must have the same name on each database and must correspond to the same directory on a shared file system. If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.</td>
</tr>
<tr>
<td>tablespace_files</td>
<td>Contains the datafiles for the cloned tablespace set. These datafiles are placed in the directories specified by the directory objects in the parameter tablespace_directory_objects.</td>
</tr>
</tbody>
</table>

Usage Notes

To run either version of this procedure, a user must meet the following requirements:

- Have **EXP_FULL_DATABASE** role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include **DBA_TABLESPACES** and **USER_TABLESPACES**.
- Have **MANAGE TABLESPACE** or **ALTER TABLESPACE** on a tablespace if the tablespace must be made read-only
- Have **READ** privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the **tablespace_names** parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.
- Have **READ** and **WRITE** privilege on the directory object that will contain the Data Pump export dump file, specified by the **dump_file** parameter or the **tablespace_directory_object** parameter
- Have **WRITE** privilege on the directory objects that will contain the copied datafiles for the tablespaces in the set, specified by the **tablespace_directory_objects** parameter if non-NULL or the **tablespace_directory_object** parameter
- Have **WRITE** privilege on the directory object that will contain the Data Pump export log file, specified by the **log_file** parameter if non-NULL or the **log_file_directory_object** parameter if non-NULL

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.
Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

After cloning a tablespace set using this procedure, you can attach the tablespaces to a different database using the ATTACH_TABLESPACES procedure.

See Also:

- Overview
- ATTACH_TABLESPACES Procedure
- DBMS_FILE_GROUP for more information about file groups
- Oracle Streams Concepts and Administration

170.4.5 DETACH_SIMPLE_TABLESPACE Procedure

This procedure detaches a simple tablespace. The specified tablespace must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace and places the dump file in the directory that contains the tablespace datafile
3. Drops the tablespace and its contents from the database

Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.DETACH_SIMPLE_TABLESPACE(
    tablespace_name       IN  VARCHAR2,
    directory_object      OUT VARCHAR2,
    tablespace_file_name  OUT VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pump_job_name</td>
<td>The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily. If NULL, then the system generates a Data Pump job name.</td>
</tr>
</tbody>
</table>
Table 170-7  (Cont.) DETACH_SIMPLE_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory_object</td>
<td>Contains the directory where the Data Pump export dump file and the Data Pump export log file are placed. The procedure uses the directory of the datafile for the tablespace. Therefore, make sure a directory object created using the SQL statement CREATE DIRECTORY exists for this directory. The name of the Data Pump export dump file is the same as the data file name for the tablespace, except with a .dmp extension. If a file exists with the same name as the dump file in the directory, then the procedure raises an error. The name of the log file is the same as the data file name for the tablespace, except with a .dlg extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</td>
</tr>
<tr>
<td>tablespace_file_name</td>
<td>Contains the name of the detached tablespace datafile.</td>
</tr>
</tbody>
</table>

Usage Notes
To run this procedure, a user must meet the following requirements:

- Have EXP_FULL_DATABASE role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have DROP TABLESPACE privilege
- Have MANAGE TABLESPACE or ALTER TABLESPACE on a tablespace if the tablespace must be made read-only
- Have READ and WRITE privilege on the directory object for the directory that contains the tablespace datafile. The name of this tablespace is specified by the tablespace_name parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure. This directory also will contain the Data Pump export dump file generated by this procedure.

After detaching a tablespace using this procedure, you can add the tablespace to a different database using the ATTACH_SIMPLE_TABLESPACE procedure. If the database is a remote database and you want to use the ATTACH_SIMPLE_TABLESPACE procedure, then you can transfer the dump file and datafile to the remote system using the DBMS_FILE_TRANSFER package, FTP, or some other method. You can use the two OUT parameters in this procedure to accomplish the attach or pull operation.

Automatic Storage Management (ASM) directories cannot be used with this procedure.

Note:
Do not use the DETACH_SIMPLE_TABLESPACE procedure on a tablespace if the tablespace is using the Oracle-managed files feature. If you do, then the datafile for the tablespace is dropped automatically when the tablespace is dropped.
170.4.6 DETACH_TABLESPACES Procedure

This procedure detaches a set of self-contained tablespaces. All of the tablespaces in the specified tablespace set must be online and any table partitions must not span tablespaces in the tablespace set.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set read-only
2. Uses Data Pump to export the metadata for the tablespace set and places the dump file in the specified directory
3. Drops the tablespaces in the specified tablespace set and their contents from the database

This procedure does not move or copy the datafiles that comprise the specified tablespace set.

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the datapump_job_name parameter. This job performs the Data Pump export. This version of the procedure completes the detach operation by placing the export dump file and export log file in the specified directories, but the files are not added to a file group version.

- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump export is performed, and this version of the procedure completes the detach operation by placing the export dump file and export log file in the appropriate file group version. The datafiles that comprise the tablespace set are not moved or copied, but they are referenced in the version that is detached. The file group and version are specified using the file_group_name and version_name parameters, respectively. Also, if the destination platform is different, then the tablespace set is migrated automatically to the correct platform when it is attached at the destination database using the file group version of the ATTACH_TABLESPACES procedure.
Note:

Do not use the DETACH_TABLESPACES procedure if any of the tablespaces in the
tablespace set are using the Oracle-managed files feature. If you do, then the
datafiles for these tablespaces are dropped automatically when the tablespaces are
dropped.

Syntax

```sql
DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES(
    datapump_job_name         IN OUT VARCHAR2,
    tablespace_names          IN     TABLESPACE_SET,
    dump_file                 IN     FILE,
    log_file                  IN     FILE  DEFAULT NULL,
    tablespace_files          OUT    FILE_SET);

DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES(
    tablespace_names          IN TABLESPACE_SET,
    export_directory_object   IN VARCHAR2  DEFAULT NULL,
    log_file_directory_object IN VARCHAR2  DEFAULT NULL,
    file_group_name           IN VARCHAR2,
    version_name              IN VARCHAR2  DEFAULT NULL,
    repository_db_link        IN VARCHAR2  DEFAULT NULL);
```

Parameters

**Table 170-8  DETACH_TABLESPACES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pump_job_name</td>
<td>The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily. If NULL, then the system generates a Data Pump job name.</td>
</tr>
<tr>
<td>tablespace_names</td>
<td>The tablespace set to be detached. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>dump_file</td>
<td>The file name of the Data Pump dump file that is exported. If NULL or if a file attribute is NULL, then the procedure raises an error. If the specified file exists, then the procedure raises an error.</td>
</tr>
<tr>
<td>log_file</td>
<td>Specify the log file name for the Data Pump export. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .dlg and places it in the dump file directory. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</td>
</tr>
<tr>
<td>tablespace_files</td>
<td>Contains the names of the datafiles for the detached tablespace set.</td>
</tr>
</tbody>
</table>
Table 170-8  (Cont.) DETACH_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>export_directory_object</td>
<td>The directory object into which the Data Pump export dump file is placed. The system generates a dump file name with the extension <code>.dmp</code>.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then procedure places the dump file in the default directory object for the version.</td>
</tr>
<tr>
<td></td>
<td>If the version does not have a default directory object, then the procedure uses the default directory object for the file group.</td>
</tr>
<tr>
<td>log_file_directory_object</td>
<td>The directory object into which the Data Pump export log file is placed. The system generates a log file name with the extension <code>.dlg</code>.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure places the export log file in the same directory as the export dump file.</td>
</tr>
<tr>
<td>file_group_name</td>
<td>The name of the file group, specified as [schema_name.]file_group_name. For example, if the schema is hq_dba and the file group name is sales, then</td>
</tr>
<tr>
<td></td>
<td>specify hq_dba.sales. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>If the specified file group does not exist, then the procedure creates it.</td>
</tr>
<tr>
<td>version_name</td>
<td>The name of the version into which the detached tablespace set is placed. The specified version name cannot be a positive integer.</td>
</tr>
<tr>
<td></td>
<td>If the specified version does not exist, then the procedure creates it.</td>
</tr>
<tr>
<td></td>
<td>If the specified version exists, then procedure adds the tablespace set to the version. Only one Data Pump export dump file can exist in a version.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure creates a new version, and the version number can be used to manage the version.</td>
</tr>
<tr>
<td>repository_db_link</td>
<td>If the file group is in a remote database, then specify the name of the database link to the database that contains the file group. The database</td>
</tr>
<tr>
<td></td>
<td>link must be accessible to the user who runs the procedure.</td>
</tr>
<tr>
<td></td>
<td>If this parameter is non-NULL, then the directory object specified in export_directory_object must exist on the local database and on the remote</td>
</tr>
<tr>
<td></td>
<td>database. If export_directory_object is NULL, then the default directory object must exist on both databases. The directory object must have the</td>
</tr>
<tr>
<td></td>
<td>same name on each database and must correspond to the same directory on a shared file system.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this either version of this procedure, a user must meet the following requirements:
• Have **EXP_FULL_DATABASE** role
• Have access to at least one data dictionary view that contains information about the tablespaces. These views include **DBA_TABLESPACES** and **USER_TABLESPACES**.
• Have **DROP TABLESPACE** privilege
• Have **MANAGE TABLESPACE** or **ALTER TABLESPACE** on a tablespace if the tablespace must be made read-only
• Have **READ** privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the **tablespace_names** parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.
• Have **READ** and **WRITE** privilege on the directory object that will contain the Data Pump export dump file, specified by the **dump_file** parameter or the **export_directory_object** parameter
• Have **WRITE** privilege on the directory object that will contain the Data Pump export the log file, specified by the **log_file** parameter if non-NULL or by the **log_file_directory_object** parameter if non-NULL

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.

Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

After detaching a tablespace set using this procedure, you can attach the tablespaces to a different database using the **ATTACH_TABLESPACES** procedure.

---

### See Also:

- **Overview**
- **ATTACH_TABLESPACES Procedure**
- **DBMS_FILE_GROUP** for more information about file groups
- **Oracle Streams Concepts and Administration**
- **Oracle Database Administrator's Guide** for more information about the Oracle-managed files feature

---

### 170.4.7 PULL_SIMPLE_TABLESPACE Procedure

This procedure copies a simple tablespace from a remote database and attaches it to the current database.

The specified tablespace at the remote database must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only at the remote database if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace

3. Uses a database link and the `DBMS_FILE_TRANSFER` package to transfer the datafile for the tablespace and the log file for the Data Pump export to the current database

4. Places the datafile for the specified tablespace and the log file for the Data Pump export in the specified directory at the local database

5. If this procedure made the tablespace read-only, then makes the tablespace read/write

6. Uses Data Pump to import the metadata for the tablespace in the local database

In addition, this procedure optionally can create a datafile for the tablespace that can be used with the local platform, if the platform at the remote database is different than the local database platform.

**Syntax**

```sql
DBMS_STREAMS_TABLESPACE_ADM.PULL_SIMPLE_TABLESPACE(
  tablespace_name    IN VARCHAR2,
  database_link      IN VARCHAR2,
  directory_object   IN VARCHAR2  DEFAULT NULL,
  conversion_extension IN VARCHAR2  DEFAULT NULL,
  convert_directory_object IN VARCHAR2  DEFAULT NULL);
```

**Parameters**

Table 170-9  PULL_SIMPLE_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tablespace_name</code></td>
<td>The tablespace to be pulled.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td><code>database_link</code></td>
<td>The name of the database link to the database that contains the tablespace to pull. The database link must be accessible to the user who runs the procedure.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td><code>directory_object</code></td>
<td>The directory object to which the datafile for the tablespace is copied on the local database. You must specify the name of a directory object created using the SQL statement <code>CREATE DIRECTORY</code>.</td>
</tr>
<tr>
<td></td>
<td>The Data Pump import log file is written to this directory. The name of the log file is the same as the data file name for the tablespace, except with a <code>.plg</code> extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>
Table 170-9  (Cont.) PULL_SIMPLE_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conversion_extension</td>
<td>Specify NULL if the platform is the same for the remote export database and the current import database. If the platform is different for the export database and the import database, then specify an extension for the tablespace datafile that is different than the extension for the tablespace datafile at the remote database. In this case, the procedure transfers the datafile to the import database and converts it to be compatible with the current import database platform automatically. After conversion is complete, the original datafile is deleted at the import database.</td>
</tr>
<tr>
<td>convert_directory_object</td>
<td>Specify NULL if the platform is the same for the remote export database and the current import database. If the platform is different for the export database and the import database, then specify a directory object in the local export database. The procedure uses the directory object for platform conversion before it transfers the files to the remote database. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY.</td>
</tr>
</tbody>
</table>

Usage Notes

To run this procedure, a user must meet the following requirements on the remote database:

- Have the EXP_FULL_DATABASE role
- Have EXECUTE privilege on the DBMS_STREAMS_TABLESPACE_ADM package
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.
- Have MANAGE TABLESPACE or ALTER TABLESPACE privilege on a tablespace if the tablespace must be made read-only
- Have READ privilege on the directory object for the directory that contains the datafile for the tablespace. The name of this tablespace is specified by the tablespace_name parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure.

To run this procedure, a user must meet the following requirements on the local database:

- Have the roles IMP_FULL_DATABASE and EXECUTE_CATALOG_ROLE
- Have WRITE privilege on the directory object that will contain the Data Pump export the log file, specified by the log_file parameter if non-NULL
- Have WRITE privilege on the directory object that will hold the datafile for the tablespace, specified by the directory_object parameter

Automatic Storage Management (ASM) directories cannot be used with this procedure.
**170.4.8 PULL_TABLESPACES Procedure**

This procedure copies a set of self-contained tablespaces from a remote database and attaches the tablespaces to the current database.

All of the tablespaces in the specified tablespace set at the remote database must be online.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set at the remote database read-only
2. Uses Data Pump to export the metadata for the tablespaces in the tablespace set
3. Uses a database link and the `DBMS_FILE_TRANSFER` package to transfer the datafiles for the tablespace set and the log file for the Data Pump export to the current database
4. Places the datafiles that comprise the specified tablespace set in the specified directories at the local database
5. Places the log file for the Data Pump export in the specified directory at the local database
6. If this procedure made a tablespace read-only, then makes the tablespace read/write
7. Uses Data Pump to import the metadata for the tablespaces in the tablespace set at the local database

In addition, this procedure optionally can create datafiles for the tablespace set that can be used with the local platform, if the platform at the remote database is different than the local database platform.

**Syntax**

```sql
DBMS_STREAMS_TABLESPACE_ADM.PULL_TABLESPACES(
    datapump_job_name            IN OUT VARCHAR2,
    database_link                IN     VARCHAR2,
    tablespace_names             IN     TABLESPACE_SET,
    tablespace_directory_objects IN     DIRECTORY_OBJECT_SET,
    log_file                     IN     FILE,
    conversion_extension         IN     VARCHAR2  DEFAULT NULL,
    convert_directory_object     IN     VARCHAR2  DEFAULT NULL);
```
Parameters

Table 170-10  PULL_TABLESPACES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| data_pump_job_name   | The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily.  
                       | If NULL, then the system generates a Data Pump job name.                                                                                     |
| database_link        | The name of the database link to the database that contains the tablespace set to pull. The database link must be accessible to the user who  
                       | runs the procedure. If NULL, then the procedure raises an error.                                                                               |
| tablespace_names     | The tablespace set to be pulled. If NULL, then the procedure raises an error.                                                                    |
| tablespace_directory_objects | The set of directory objects to which the datafiles for the tablespaces are copied. If multiple directory objects are in the set, then the  
                                      procedure copies a datafile to each directory object in the set in sequence. In this case, if the  
                                      end of the directory object set is reached, then datafile copying starts again with the first  
                                      directory object in the set. If NULL, then the procedure raises an error.            |
| log_file             | Specify the log file name for the Data Pump export. If NULL or if at least one file parameter is NULL, then the system generates a log file  
                       | name with the extension .plg and places it in one of the data file directories. If a file exists with the same name as the log file in the  
                       | directory, then the procedure overwrites the file.                                                                                        |
| conversion_extension | Specify NULL if the platform is the same for the remote export database and the current import database.  
                       | If the platform is different for the export database and the import database, then specify an extension for the tablespace datafiles that is  
                       | different than the extension for the tablespace datafiles at the remote database. In this case, the procedure transfers the datafiles to the  
                       | import database and converts them to be compatible with the current import database platform automatically. After conversion is complete, the  
                       | original datafiles are deleted at the import database.                                                                                    |
| convert_directory_object | Specify NULL if the platform is the same for the remote export database and the current import database.  
                                | If the platform is different for the export database and the import database, then specify a directory object in the local export database.  
                                | The procedure uses the directory object for platform conversion before it transfers the files to the remote database. You must specify the  
                                | name of a directory object created using the SQL statement CREATE DIRECTORY.                                                               |

Usage Notes

To run this procedure, a user must meet the following requirements on the remote database:
• Have the EXP_FULL_DATABASE role

• Have EXECUTE privilege on the DBMS_STREAMS_TABLESPACE_ADM package

• Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA_TABLESPACES and USER_TABLESPACES.

• Have MANAGE TABLESPACE or ALTER TABLESPACE privilege on a tablespace if the tablespace must be made read-only

• Have READ privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the tablespace_names parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.

To run this procedure, a user must meet the following requirements on the local database:

• Have the roles IMP_FULL_DATABASE and EXECUTE_CATALOG_ROLE

• Have WRITE privilege on the directory object that will contain the Data Pump export the log file, specified by the log_file parameter if non-NULL

• Have WRITE privilege on the directory objects that will hold the datafiles for the tablespaces in the set, specified by the tablespace_directory_objects parameter

Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

See Also:

Overview
The DBMS_SYNC_REFRESH package provides an interface to perform a synchronous refresh of materialized views.

**See Also:**

*Oracle Database Data Warehousing Guide* for more information on using DBMS_SYNC_REFRESH

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Summary of DBMS_SYNC_REFRESH Subprograms**

### 171.1 DBMS_SYNC_REFRESH Overview

Synchronous refresh is a refresh method introduced in Oracle Database Release 12c, which enables you to keep a set of tables and the materialized views defined on them to be always in sync.

**See Also:**

*Oracle Database Data Warehousing Guide* for more information about using synchronous refresh

### 171.2 DBMS_SYNC_REFRESH Security Model

The execute privilege for this package is granted to PUBLIC, so all users can execute the procedures in this package to perform synchronous refresh on objects owned by them. The database administrator can perform synchronous refresh operations on all tables and materialized views in the database.

In general, if a user without the DBA privilege wants to use synchronous refresh on another user's table, he must complete privileges to read from and write to that table, that it, the user must have the SELECT or READ, INSERT, UPDATE, and DELETE privileges on that table or materialized view. A couple of exceptions are:

- **PURGE_REFRESH_STATS** and **ALTER_REFRESH_STATS_RETENTION** Functions

These two functions implement the purge policy and can be used to change the default retention period. These functions can only be executed by the database administrator.
• **The CAN_SYNCREF_TABLE Function**

This is an advisory function which examines the eligibility for sync refresh of all the materialized views associated with a specified table. Hence, this function requires the `SELECT` or `READ` privilege on all materialized views associated with the specified table.

### 171.3 Summary of DBMS_SYNC_REFRESH Subprograms

This table lists and briefly describes the `DBMS_SYNC_REFRESH` package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT_REFRESH Procedure</td>
<td>Aborts a refresh.</td>
</tr>
<tr>
<td>ALTER_REFRESH_STATS_RETENTION</td>
<td>Alters the refresh history retention value, specified in days.</td>
</tr>
<tr>
<td>CAN_SYNCREF_TABLE Procedure</td>
<td>Advises on whether a table and its dependent materialized views are eligible for synchronous refresh.</td>
</tr>
<tr>
<td>EXECUTE_REFRESH Procedure</td>
<td>Executes synchronous refresh on the synchronous refresh groups.</td>
</tr>
<tr>
<td>GET_ALL_GROUP_IDS Function</td>
<td>Returns the group IDs of all the synchronous refresh groups in the database.</td>
</tr>
<tr>
<td>GET_GROUP_ID Function</td>
<td>Returns the group ID of a table or materialized view.</td>
</tr>
<tr>
<td>GET_GROUP_ID_LIST Function</td>
<td>Returns the group IDs of the tables in a given list of objects (tables or materialized views).</td>
</tr>
<tr>
<td>PREPARE_REFRESH Procedure</td>
<td>Prepares the sync refresh groups for refresh.</td>
</tr>
<tr>
<td>PREPARE_STAGING_LOG Procedure</td>
<td>Validates and collects statistics on the data in the staging log.</td>
</tr>
<tr>
<td>PURGE_REFRESH_STATS Procedure</td>
<td>Purges the refresh history of sync refreshing that took place within a time specified by a timestamp parameter.</td>
</tr>
<tr>
<td>REGISTER_MVIEWS</td>
<td>Registers materialized views for synchronous refresh.</td>
</tr>
<tr>
<td>REGISTER_PARTITION_OPERATION</td>
<td>Registers a partition maintenance operation on a partition of a base table.</td>
</tr>
<tr>
<td>UNREGISTER_MVIEWS</td>
<td>Unregisters materialized views from synchronous refresh.</td>
</tr>
<tr>
<td>UNREGISTER_PARTITION_OPERATION</td>
<td>Unregisters a partition maintenance operation on a partition of a base table.</td>
</tr>
</tbody>
</table>

### 171.3.1 ABORT_REFRESH Procedure

This procedure undoes all the changes made by `PREPARE_REFRESH` or `EXECUTE_REFRESH` for the specified sync refresh groups. It helps you to recover to a state where the tables and materialized views are usable and consistent in case they encounter unexpected errors.

This procedure is overloaded.
Syntax

DBMS_SYNC_REFRESH.ABORT_REFRESH (group_id IN NUMBER);

DBMS_SYNC_REFRESH.ABORT_REFRESH (group_id_list IN DBMS_UTILITY.NUMBER_ARRAY);

Parameters

Table 171-2  ABORT_REFRESH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>The group ID of a sync refresh group.</td>
</tr>
<tr>
<td>group_id_list</td>
<td>An array of group IDs of the sync refresh groups to be aborted for sync refresh.</td>
</tr>
</tbody>
</table>

Usage Notes

If called after PREPARE_REFRESH, this procedure drops the outside tables created by it and unlocks the tables and materialized views in the sync refresh group.

If called after EXECUTE_REFRESH fails, this procedure restores the state of tables to before EXECUTE_REFRESH by undoing any partition exchanges which successfully finished.

This procedure releases the locks placed on the tables in the sync refresh group which were placed on them by the PREPARE_REFRESH procedure. See "PREPARE_REFRESH Procedure" for a description of these locks.

ABORT_REFRESH will work only if a PREPARE_REFRESH or EXECUTE_REFRESH statement has failed. It cannot be used after successful runs of those commands, and throws an error in such cases.

171.3.2 ALTER_REFRESH_STATS_RETENTION Procedure

This procedure alters the refresh history retention value, specified in days. It is intended for use in conjunction with PURGE_REFRESH_HISTORY. It also requires the SYSDBA privilege in addition to the privilege to execute it.

Syntax

DBMS_SYNC_REFRESH.ALTER_REFRESH_STATS_RETENTION (retention IN NUMBER);
### Parameters

**Table 171-3  ALTER_REFRESH_STATS_RETENTION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| retention | The retention time in days. The refresh history will be retained for at least these many number of days. The valid range is 1 to 365,000. You can use the following values for special purposes:  
  - -1 - Refresh history is never purged by PREPARE_REFRESH.  
  - 0 - Old refresh history is never saved. PREPARE_REFRESH will delete all refresh history.  
  - NULL - Change refresh history retention to default value. |

#### 171.3.3 CAN_SYNCREF_TABLE Procedure

This procedure advises on whether a table and its dependent materialized views are eligible for sync refresh. It provides an explanation of its analysis. If not eligible, you can examine the reasons and take appropriate action if possible.

This procedure lists all of the table's dependent materialized views and whether they qualify for sync refresh. Note that a materialized view may qualify for sync refresh even though the base table may not.

The eligibility rules for materialized views for synchronous refresh are discussed in detail in *Oracle Database Data Warehousing Guide*.

You can invoke CAN_SYNCREF_TABLE in two ways. The first is to use a table, while the second is to create a VARRAY.

**Syntax**

```sql
DBMS_SYNC_REFRESH.CAN_SYNCREF_TABLE (  
    schema_name    IN VARCHAR2,  
    table_name     IN VARCHAR2,  
    statement_id   IN VARCHAR2);
```

```sql
DBMS_SYNC_REFRESH.CAN_SYNCREF_TABLE (  
    schema_name    IN VARCHAR2,  
    table_name     IN VARCHAR2,  
    output_array   IN OUT Sys.CanSyncRefTypeArray);
```

Note that only one of statement_id or output_array need be provided to CAN_SYNCREF_TABLE.

**Parameters**

**Table 171-4  CAN_SYNCREF_TABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema of the base table.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the base table.</td>
</tr>
</tbody>
</table>
Table 171-4  (Cont.) CAN_SYNCREF_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement_id</td>
<td>A string (VARCHAR2(30)) to identify the rows pertaining to an invocation of CAN_SYNCREF_TABLE when the output is directed to a table named SYNCREF_TABLE in the user's schema.</td>
</tr>
<tr>
<td>output_array</td>
<td>The output array into which CAN_SYNCREF_TABLE records the information on the eligibility of the base table and its dependent materialized views for synchronous refresh.</td>
</tr>
</tbody>
</table>

Using SYNCREF_TABLE

The output of CAN_SYNCREF_TABLE can be directed to a table named SYNCREF_TABLE. The user is responsible for creating the SYNCREF_TABLE; it can be dropped when it is no longer needed. Its structure is as follows:

```sql
CREATE TABLE SYNCREF_TABLE (
    statement_id    VARCHAR2(30),
    schema_name     VARCHAR2(30),
    table_name      VARCHAR2(30),
    mv_schema_name  VARCHAR2(30),
    mv_name         VARCHAR2(30),
    eligible        VARCHAR2(1),
    seq_num         NUMBER,
    msg_number      NUMBER,
    message         VARCHAR2(4000));
```

Using a VARRAY

You can save the output of CAN_SYNCREF_TABLE in a PL/SQL VARRAY. The elements of this array are of type CanSyncRefMessage, which is predefined in the SYS schema, as shown in the following:

```plsql
TYPE CanSyncRefMessage IS OBJECT (
    schema_name     VARCHAR2(30),
    table_name      VARCHAR2(30),
    mv_schema_name  VARCHAR2(30),
    mv_name         VARCHAR2(30),
    eligible        VARCHAR2(1),
    seq_num         NUMBER,
    msg_number      NUMBER,
    message         VARCHAR2(4000));
```

The array type CanSyncRefArrayType, which is a varray of CanSyncRefMessage objects, is predefined in the SYS schema as follows:

```plsql
TYPE CanSyncRefArrayType AS VARRAY(256) OF CanSyncRefMessage;
```

Each CanSyncRefMessage record provides a message concerning the eligibility of the base table or a dependent materialized view for synchronous refresh. The semantics of the fields is the same as that of the corresponding fields in the SYNCREF_TABLE. However, the SYNCREF_TABLE has a statement_id field which is absent in CanSyncRefMessage because no statement_id is supplied (because it is not required) when CAN_SYNCREF_TABLE is called with a VARRAY parameter.
171.3.4 EXECUTE_REFRESH Procedure

This procedure executes sync refresh on the sync refresh groups prepared by DBMS_SYNC_REFRESH.PREPARE_REFRESH. These groups are identified by their group IDs.

Note this procedure will only perform the refresh on those materialized views that have been registered for synch refresh; any other materialized views will become stale once this procedure completes.

For more information on how to monitor the status of the two synchronous refresh operations, PREPARE_REFRESH and EXECUTE_REFRESH and how to troubleshoot errors that might occur using the information in the catalog views, refer to "Trouble-Shooting Synchronous Refresh Operations" in Oracle Database Data Warehousing Guide.

This procedure is overloaded.

Syntax

DBMS_SYNC_REFRESH.EXECUTE_REFRESH (group_id IN NUMBER);

DBMS_SYNC_REFRESH.EXECUTE_REFRESH (group_id_list IN DBMS_UTILITY.NUMBER_ARRAY);

Parameters

Table 171-5 EXECUTE_REFRESH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>The group ID of a sync refresh group.</td>
</tr>
<tr>
<td>group_id_list</td>
<td>An array of group IDs of the sync refresh groups to be executed for sync refresh.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure also releases the locks placed on the tables in the sync refresh group that were placed on them by the PREPARE_REFRESH procedure. See "PREPARE_REFRESH Procedure" for a description of these locks and Oracle Database Reference for information regarding the status of the refresh operation after DBMS_SYNC_REFRESH.EXECUTE_REFRESH.

171.3.5 GET_ALL_GROUP_IDs Function

This function returns the group IDs of all the sync refresh groups in the database.

Syntax

FUNCTION DBMS_SYNC_REFRESH.GET_ALL_GROUP_IDs
    RETURN DBMS_UTILITY.NUMBER_ARRAY;
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_all_group_ids</td>
<td>Returns the group IDs of all the sync refresh groups in the database.</td>
</tr>
</tbody>
</table>

171.3.6 GET_GROUP_ID Function

This function returns the group ID of a materialized view. The group ID identifies the sync refresh group the table belongs to. A sync refresh group is a group of related tables and their dependent materialized views which must be all refreshed together jointly to ensure consistency and correctness.

Syntax

```sql
DBMS_SYNC_REFRESH.GET_GROUP_ID (
    object_name_list   IN VARCHAR2
) RETURN DBMS_UTILITY.NUMBER_ARRAY;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name_list</td>
<td>The name of the materialized view. The name can be schema-qualified.</td>
</tr>
</tbody>
</table>

171.3.7 GET_GROUP_ID_LIST Function

This function returns the group IDs of the tables in a given list of objects (materialized views).

Syntax

```sql
DBMS_SYNC_REFRESH.GET_GROUP_ID_LIST (
    object_name_list   IN VARCHAR2
) RETURN DBMS_UTILITY.NUMBER_ARRAY;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name_list</td>
<td>A comma-separated list of object names (materialized views). Each name can be schema-qualified.</td>
</tr>
</tbody>
</table>
171.3.8 PREPARE_REFRESH Procedure

This procedure prepares for refresh the sync refresh groups identified by the group ID in the input.

A sync refresh group consists of a set of related tables and all materialized views dependent on those base tables. Note this procedure will only prepare for refresh those dependent materialized views that have been registered for synchronous refresh.

For more information on how to monitor the status of the two synchronous refresh operations, PREPARE_REFRESH and EXECUTE_REFRESH and how to troubleshoot errors that might occur using the information in the catalog views, refer to "Trouble-Shooting Synchronous Refresh Operations" in Oracle Database Data Warehousing Guide.

Syntax

```sql
DBMS_SYNC_REFRESH.PREPARE_REFRESH (
    group_id   IN NUMBER
) RETURN DBMS_UTILITY.NUMBER_ARRAY;
```

Parameters

Table 171-9  PREPARE_REFRESH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>The group ID of the sync refresh group to be prepared for sync refresh.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure plans the three phases of the sync refresh operation and executes the steps associated with the prepare phase itself. These steps include identifying the partitions of the fact tables and materialized views that have been changed, and computing their new values as a result of the changes. The new values of the partitions are stored in tables called outside tables that are exchanged into their corresponding partitions at the time of the EXECUTE_REFRESH.

Before running this procedure, the user must run PREPARE_STAGING_LOG on all tables in the group. This is required even for staging logs that do not have changes in them. The user must also register any partition operations on the tables in the group using the REGISTER_PARTITION_OPERATION.

One of the side effects of this procedure is that the tables being prepared are locked in this sense: the staging logs of the tables will be locked to prevent any DMLs from occurring and the registration of partition operations will be disabled. These locks will be in effect until you issue an EXECUTE_REFRESH statement. Alternatively, you can issue an ABORT_REFRESH operation to release these locks. Another side effect of this procedure is that it purges from the catalog records of earlier sync refresh operations; if they are older than the retention period, they are purged.

The degree of parallelism of the prepare refresh job is inherited from the session parameters which you can control with an ALTER SESSION statement.
The group ID of a table can be found using \texttt{GET\_GROUP\_ID}(table\_name). The group IDs of a list of tables can be found with \texttt{GET\_GROUP\_ID\_LIST}(table\_name\_list). The group IDs of all the lists of tables can be retrieved with \texttt{GET\_ALL\_GROUP\_IDS}.

By default, synchronous refresh does not maintain global indexes belonging to the tables and materialized views in the sync refresh group. If you wish to do so, you can set event 31904, level 64 before executing \texttt{PREPARE\_REFRESH}. This will cause the partition exchange DDL statements generated by \texttt{PREPARE\_REFRESH} to have the \texttt{UPDATE\_INDEXES} clause appended to them, and when they are executed by \texttt{EXECUTE\_REFRESH}, the global indexes will be maintained.

171.3.9 \texttt{PREPARE\_STAGING\_LOG} Procedure

This procedure collects statistics on the data in the staging log of the base table and validates the data in the log.

It can be run in several different modes ranging from the enforced mode in which strict checking of the data is done to trusted mode in which no checking is done. You should run this procedure after loading the staging log and before running \texttt{PREPARE\_REFRESH}.

In the enforced mode, which is the default, this procedure will fill in the missing values of the columns of the rows being deleted or updated. An error is thrown if any violations of the staging-log rules are found. You can query the view \texttt{USER\_SR\_STLOG\_EXCEPTIONS} to get details on the exceptions.

The notion of the staging log key is described in \emph{Oracle Database Data Warehousing Guide}.

In the enforced mode, this procedure processes each delete/update row in the staging log as follows:

- It verifies the existence of the row in the base table using the key.
- For the rows being deleted (\texttt{DMLTYPE} is 'D'), it verifies a row with this key exists in the base table; if non-null non-key values are supplied in the staging log, it verifies the values match the corresponding columns in the base table; else an exception is logged in the exceptions table. If the values of any of the non-key columns are missing, it fills in those values from the row in the base table.
- For the rows being updated (\texttt{DMLTYPE} is 'UO' or 'UN'), it verifies a row with this key exists in the base table. In the old values row (\texttt{DMLTYPE} is 'UO'), it checks that at least the value of one the columns differs from its old value; else an exception is logged.
- In the new values row (\texttt{DMLTYPE} is 'UN'), a null value in a column is interpreted as having the same value as the old value of the column except if the old value is non-null and the new value is null in which case, the new value of the column is interpreted as being null. This requires that the user must provide the old value of columns which are being updated to \texttt{NULL}.

In the default enforced mode, this procedure verifies that each key is specified for at most once for a delete or update operation. This means that the user, when doing the change consolidation, must consolidate delete-insert of the same row into an update operation with rows 'UO' and 'UN'; multiple updates must be consolidated into a single update; and null changes such as an insert-update-delete of the same row must not appear in the staging log.

The checking done in the enforced mode can be time-consuming. If you are confident in the integrity of the data, you can choose a lower level of checking. You can choose to:
• trust all the insert rows (DMLTYPE$ is 'I') by choosing the psl_mode of DBMS_SYNC_REFRESH.INSERT_TRUSTED
• trust all the delete rows (DMLTYPE$ is 'D') by choosing the psl_mode of DBMS_SYNC_REFRESH.DELETE_TRUSTED
• trust all the update rows (DMLTYPE$ is 'UO' or 'UN') by choosing the psl_mode of DBMS_SYNC_REFRESH.UPDATE_TRUSTED
• trust all three types of DMLs by choosing the psl_mode of DBMS_SYNC_REFRESH.TRUSTED.

In addition, you can specify the psl_mode as a bitmask of the flags described above. For example, DBMS_SYNC_REFRESH.INSERT_TRUSTED + DBMS_SYNC_REFRESH.DELETE_TRUSTED will treat inserts and deletes to be trusted but not updates.

Syntax

DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG {
    schema_name IN VARCHAR2,
    base_table_name IN VARCHAR2,
    psl_mode IN NUMBER DEFAULT DBMS_SYNC_REFRESH.ENFORCED);

Parameters

Table 171-10  PREPARE_STAGING_LOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema of the base table.</td>
</tr>
<tr>
<td>base_table_name</td>
<td>The name of the base table.</td>
</tr>
<tr>
<td>psl_mode</td>
<td>The mode in which staging log preparation should be done. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>• DBMS_SYNC_REFRESH.ENFORCED (the default)</td>
</tr>
<tr>
<td></td>
<td>• DBMS_SYNC_REFRESH.INSERT_TRUSTED</td>
</tr>
<tr>
<td></td>
<td>• DBMS_SYNC_REFRESH.DELETE_TRUSTED</td>
</tr>
<tr>
<td></td>
<td>• DBMS_SYNC_REFRESH.UPDATE_TRUSTED</td>
</tr>
<tr>
<td></td>
<td>• DBMS_SYNC_REFRESH.TRUSTED</td>
</tr>
</tbody>
</table>

171.3.10 PURGE_REFRESH_STATS Procedure

This procedure purges the refresh history of sync refreshes that took place before the value specified by the BEFORE_TIMESTAMP parameter.

This procedure requires the SYSDBA privilege in addition to the privilege to execute it.

Syntax

DBMS_SYNC_REFRESH.PURGE_REFRESH_STATS {
    before_timestamp IN TIMESTAMP WITH TIME ZONE);
171.3.11 REGISTER_MVIEWS

This procedure registers a list of materialized views for synchronous refresh.

It checks each materialized view in the list for eligibility and places it in the sync refresh group it belongs to. A sync refresh group is a set of related tables and materialized views defined on top of them. Two tables are considered related if there is a referential constraint between them.

The eligibility rules of materialized views for synchronous refresh are described in detail in Oracle Database Data Warehousing Guide. The principal requirements are that the materialized view must be partitioned and its partition key must be derivable from the partition key of its fact table. The materialized view definition must specify the USING TRUSTED CONSTRAINTS clause because sync refresh trusts the foreign key and primary key relationships to perform various refresh optimizations. The materialized view's refresh policy must be specified as ON DEMAND.

You have an option to register only some of the materialized views associated with a table, and leave some unregistered. Oracle Corporation does not recommend this, and in such a case, the user has to maintain the unregistered ones using the PCT or complete refresh methods.

A staging table must have been created for each base table of each materialized view in the materialized view list (mv_list), or else an error is thrown.

If any of the materialized views are not eligible for sync refresh, an error is thrown and the registration of all materialized views in the materialized view list fails.

Syntax

```
DBMS_SYNC_REFRESH.REGISTER_MVIEWS (  
    mv_list   IN VARCHAR2);
```

Parameter

Table 171-12  REGISTER_MVIEWS Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv_list</td>
<td>A comma-delimited list of materialized views to register. These names are optionally schema-qualified.</td>
</tr>
</tbody>
</table>
171.3.12 REGISTER_PARTITION_OPERATION Procedure

This procedure registers a partition-maintenance operation (PMOP) on a partition of a base table.

Syntax

```
DBMS_SYNC_REFRESH.REGISTER_PARTITION_OPERATION (  
  partition_op                IN VARCHAR2,  
  schema_name                 IN VARCHAR2,  
  base_table_name             IN VARCHAR2,  
  partition_name              IN VARCHAR2,  
  outside_partn_table_schema  IN VARCHAR2,  
  outside_partn_table_name    IN VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition_op</td>
<td>The name of the partition operation (DROP, EXCHANGE, or TRUNCATE).</td>
</tr>
<tr>
<td>schema_name</td>
<td>The name of the schema of the base table.</td>
</tr>
<tr>
<td>base_table_name</td>
<td>The name of the base table.</td>
</tr>
<tr>
<td>partition_name</td>
<td>The name of the partition to be changed; either exchanged with the outside partition table or dropped or truncated.</td>
</tr>
<tr>
<td>outside_partn_table_schema</td>
<td>The name of the schema of the outside partition table (required for EXCHANGE only).</td>
</tr>
<tr>
<td>outside_partn_table_name</td>
<td>The name of the outside partition table (required for EXCHANGE only).</td>
</tr>
</tbody>
</table>

Usage Notes

The three kinds of change operations that may be specified on partitions are DROP, TRUNCATE, and EXCHANGE.

If DROP is specified, then the partition will be dropped from the base table at the time of EXECUTE_REFRESH. If TRUNCATE is specified, then the data from the partition will be deleted but the partition itself will not be dropped. These operations provide a more efficient way of specifying the deletes of all the rows in a partition than specifying them individually in the staging log.

If EXCHANGE is specified, then the contents of the outside table is exchanged with contents of the specified partition of EXECUTE_REFRESH. This provides an alternative method to the user of providing the changes to the base tables instead of populating the staging log.
171.3.13 UNREGISTER_MVIEWS

This procedure unregisters a list of materialized views from synchronous refresh. Once a materialized view is unregistered, it can be maintained by the user with any of the traditional refresh methods, such as complete or PCT, refresh.

Syntax

```sql
DBMS_SYNC_REFRESH.UNREGISTER_MVIEWS (  
    mv_list   IN VARCHAR20;
)
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv_list</td>
<td>A comma-delimited list of materialized views to unregister. These names are optionally schema-qualified.</td>
</tr>
</tbody>
</table>

171.3.14 UNREGISTER_PARTITION_OPERATION Procedure

This procedure unregisters a partition-maintenance operation (PMOP) that had been previously registered with `REGISTER_PARTITION_OPERATION` on a base table. The three kinds of change operations that can be specified on partitions are `DROP`, `TRUNCATE`, and `EXCHANGE`.

Syntax

```sql
DBMS_SYNC_REFRESH.UNREGISTER_PARTITION_OPERATION (  
    partition_op     IN VARCHAR2,  
    schema_name      IN VARCHAR2,  
    base_table_name  IN VARCHAR2,  
    partition_name   IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition_op</td>
<td>The name of the partition operation (DROP, EXCHANGE, or TRUNCATE).</td>
</tr>
<tr>
<td>schema_name</td>
<td>The name of the schema of the base table.</td>
</tr>
<tr>
<td>base_table_name</td>
<td>The name of the base table.</td>
</tr>
<tr>
<td>partition_name</td>
<td>The name of the partition to be changed; either exchanged with the outside partition table or dropped or truncated.</td>
</tr>
</tbody>
</table>
The DBMS_TDB package reports whether a database can be transported between platforms using the RMAN CONVERT DATABASE command.

The package verifies that databases on the current host platform are of the same endian format as the destination platform, and that the state of the current database does not prevent transport of the database.

See Also:
Oracle Database Backup and Recovery User's Guide regarding database transport using CONVERT DATABASE

This chapter contains the following topics:

• Overview
• DBMS_TDB Security Model
• Constants
• Views
• Operational Notes
• Summary of DBMS_TDB Subprograms

172.1 DBMS_TDB Overview

In many cases, Oracle supports transporting databases between platforms which have the same endian format. However, even when the endian formats are the same, a database must undergo a conversion process to move from one platform to another. There are also preconditions required for the process of transporting a database, such as having the database to be transported open read-only.

The DBMS_TDB package serves two purposes:

• Confirming that Oracle supports transporting a database from a given source platform to a given target platform
• Determining whether a database to be transported has been properly prepared for transport, and if not, identifying the condition that prevents database transport

The actual conversion is performed using the Recovery Manager CONVERT DATABASE command. For a complete discussion of the requirements for transporting a database, the process of converting a database for transport across platforms, and examples of the use of the DBMS_TDB subprograms in the conversion process, see Oracle Database Backup and Recovery User's Guide.
172.2 DBMS_TDB Security Model

Use of this package requires the DBA privilege.

172.3 DBMS_TDB Constants

The DBMS_TDB package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, DBMS_TDB.SKIP_NONE.

The DBMS_TDB package uses the constants shown in Table 172-1.

Table 172-1 DBMS_TDB Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKIP_NONE</td>
<td>NUMBER</td>
<td>0</td>
<td>Check all files when checking whether a database is ready for transport.</td>
</tr>
<tr>
<td>SKIP_OFFLINE</td>
<td>NUMBER</td>
<td>2</td>
<td>Skip files in offline tablespaces when checking whether a database is ready for transport.</td>
</tr>
<tr>
<td>SKIP_READONLY</td>
<td>NUMBER</td>
<td>3</td>
<td>Skip files in read-only tablespaces when checking whether a database is ready for transport.</td>
</tr>
</tbody>
</table>

172.4 DBMS_TDB Views

The DBMS_TDB package uses the V$DB_TRANSPORTABLE_PLATFORM view.

This view is described in Oracle Database Reference.

• V$DB_TRANSPORTABLE_PLATFORM, which specifies which combinations of source and target platforms support database transport

172.5 DBMS_TDB Operational Notes

The following notes apply to DBMS_TDB.

• The subprograms in this package are useful both in determining whether the desired cross-platform database conversion is possible, and in checking whether the database is ready for conversion. See Oracle Database Backup and Recovery User's Guide for details on the different uses of these subprograms are used in the conversion process.

• The subprograms in this package return simple TRUE or FALSE results to indicate whether database transport is possible. Use the subprograms with SERVEROUTPUT ON for informative messages about why transport is not possible.

172.6 Summary of DBMS_TDB Subprograms

This table lists the DBMS_TDB subprograms and briefly describes them.
Table 172-2  DBMS_TDB Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK_DB Function</td>
<td>Checks whether a database can be transported to a target platform</td>
</tr>
<tr>
<td>CHECK_EXTERNAL Function</td>
<td>Checks whether a database has external tables, directory or BFILEs</td>
</tr>
</tbody>
</table>

172.6.1 CHECK_DB Function

This function checks whether a database can be transported to a target platform. It tests whether transport is supported at all for a given source and destination platform, and whether the database is currently in the correct state for transport.

You can specify whether to skip checking parts of the database that are read-only or offline, if you do not plan to transport them.

The function is overloaded. The different functionality of each form of syntax is presented along with the definition.

Syntax

```
DBMS_TDB.CHECK_DB (  
    target_platform_name   IN VARCHAR2,  
    skip_option            IN  NUMBER)  
RETURN BOOLEAN;
```

```
DBMS_TDB.CHECK_DB (  
    target_platform_name   IN VARCHAR2)  
RETURN BOOLEAN;
```

```
DBMS_TDB.CHECK_DB  
RETURN BOOLEAN;
```

Parameters

Table 172-3  CHECK_DB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_platform_name</td>
<td>The name of the destination platform, as it appears in V$DB_TRANSPORTABLE_PLATFORM.</td>
</tr>
<tr>
<td>skip_option</td>
<td>Specifies which, if any, parts of the database to skip when checking whether the database can be transported. Supported values are listed in Table 172-1.</td>
</tr>
</tbody>
</table>

Return Values

If the database cannot be transported to the target platform or is not ready to be transported, returns FALSE. If the database is ready for transport, returns TRUE.
Usage Notes

- If `SERVEROUTPUT` is on, then the output will contain the reasons why the database cannot be transported and how to fix the problems. For details on possible reasons and fixes, see Table 172-4.

### Table 172-4  Reasons for CHECK_DB Function to Return FALSE

<table>
<thead>
<tr>
<th>Cause</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrecognized target platform name.</td>
<td>Check <code>V$DB_TRANSPORTABLE_PLATFORM</code> for recognized platform names.</td>
</tr>
<tr>
<td>Target platform has a different endian format.</td>
<td>Conversion is not supported.</td>
</tr>
<tr>
<td>Database is not open read-only.</td>
<td>Open database read-only and retry.</td>
</tr>
<tr>
<td>There are active or in-doubt transactions in the database.</td>
<td>Open the database read-write. After the active transactions are rolled back, open the database read-only and retry the operation. This situation can occur if users flash back the database and open it read only. The active transactions will be rolled back when the database is opened read-write.</td>
</tr>
<tr>
<td>Deferred transaction rollback needs to be done.</td>
<td>Open the database read-write and bring online the necessary tablespaces. Once the deferred transaction rollback is complete, open the database read-only and retry the operation.</td>
</tr>
<tr>
<td>Database compatibility version is below 10.0.0.</td>
<td>Change the <code>COMPATIBLE</code> initialization parameter to 10.0.0 or higher, open the database read-only, and retry the operation.</td>
</tr>
<tr>
<td>Some tablespaces have not been open read-write with compatibility version is 10.0.0 or higher.</td>
<td>Change the <code>COMPATIBLE</code> initialization parameter to 10.0.0 or higher, then open the affected tablespaces read-write. Shut down the database, open it read-only, and retry the operation.</td>
</tr>
</tbody>
</table>

Examples

This example illustrates the use of `CHECK_DB` with a database that is open read-write:

```sql
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     db_ready BOOLEAN;
  3  BEGIN
  4     db_ready := DBMS_TDB.CHECK_DB('Microsoft Windows IA (32-bit)');
  5  END;
  6 /

Database is not open READ ONLY. Please open database READ ONLY and retry.

PL/SQL procedure successfully completed.
```
172.6.2 CHECKEXTERNAL Function

This function determines whether a database has external tables, directories, or BFILEs.

Syntax

```sql
DBMS_TDB.CHECK_EXTERNAL
    RETURN BOOLEAN;
```

Return Values

If the database has external tables, directories, or BFILEs, return `TRUE`. Otherwise, return `FALSE`.

Usage Notes

- If `SERVEROUTPUT` is `ON`, then the function will output the names of the external tables, directories, and BFILEs in the database.
- The database must be open read-write.

Examples

This example illustrates the use of `CHECK_EXTERNAL` with a database that has several external tables, directories, and BFILEs:

```sql
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
       external BOOLEAN;
    BEGIN
       external := DBMS_TDB.CHECK_EXTERNAL;
    END;
/

The following external tables exist in the database:

SH.SALES_TRANSACTIONS_EXT

The following directories exist in the database:

SYS.MEDIA_DIR, SYS.DATA_FILE_DIR, SYS.LOG_FILE_DIR, SYS.DATA_PUMP_DIR

The following BFILEs exist in the database:

PM.PRINT_MEDIA

PL/SQL procedure successfully completed.
The DBMS_TF package contains utilities for Polymorphic Table Functions (PTF) implementation. You can use DBMS_TF subprograms to consume and produce data, and get information about its execution environment.

You must be familiar with the Polymorphic Table Function (PTF) concepts, syntax and semantics.

**See Also:**

- *Oracle Database PL/SQL Language Reference* for an overview of Polymorphic Table Function (PTF) concepts
- *Oracle Database PL/SQL Language Reference* for more information about `CREATE FUNCTION PIPELINED` clause syntax and semantics

This chapter contains the following topics:

- DBMS_TF Overview
- DBMS_TF Security Model
- DBMS_TF Constants
- DBMS_TF Operational Notes
- DBMS_TF Execution Flow
- DBMS_TF Restrictions
- DBMS_TF Examples
- DBMS_TF Data Structures
- Summary of DBMS_TF Subprograms

### 173.1 DBMS_TF Overview

The DBMS_TF package contains types, constants, and subprograms that can be used by Polymorphic Table Functions (PTFs).

Polymorphic Table Functions (PTFs) need various services from the database to implement their functionality. PTFs need a mechanism to get rows from the database and send back new rows, for instance. The DBMS_TF package provides these server and client interfaces utilities.
173.2 DBMS_TF Security Model

PUBLIC is granted the EXECUTE privilege on package DBMS_TF. Its subprograms execute with invoker’s rights privileges.

173.3 DBMS_TF Constants

This topic describes useful constants defined in the DBMS_TF package.

The DBMS_TF package defines several enumerated constants that should be used for specifying parameter values or types. Enumerated constants must be prefixed with the package name, for example, DBMS_TF.TYPE_DATE.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_BINARY_DOUBLE</td>
<td>Type code for BINARY_DOUBLE</td>
</tr>
<tr>
<td>TYPE_BINARY_FLOAT</td>
<td>Type code for BINARY_FLOAT</td>
</tr>
<tr>
<td>TYPE_BLOB</td>
<td>Type code for BLOB</td>
</tr>
<tr>
<td>TYPE_CHAR</td>
<td>Type code for CHAR</td>
</tr>
<tr>
<td>TYPE_CLOB</td>
<td>Type code for CLOB</td>
</tr>
<tr>
<td>TYPE_DATE</td>
<td>Type code for DATE</td>
</tr>
<tr>
<td>TYPE_INTERVAL_DS</td>
<td>Type code for INTERVAL_DS</td>
</tr>
<tr>
<td>TYPE_INTERVAL_YM</td>
<td>Type code for INTERVAL_YM</td>
</tr>
<tr>
<td>TYPE_NUMBER</td>
<td>Type code for NUMBER</td>
</tr>
<tr>
<td>TYPE_ROWID</td>
<td>Type code for ROWID</td>
</tr>
<tr>
<td>TYPE_RAW</td>
<td>Type code for RAW</td>
</tr>
<tr>
<td>TYPE_TIMESTAMP</td>
<td>Type code for TIMESTAMP</td>
</tr>
<tr>
<td>TYPE_TIMESTAMP_TZ</td>
<td>Type code for TIMESTAMP_TZ</td>
</tr>
<tr>
<td>TYPE_VARCHAR2</td>
<td>Type code for VARCHAR2</td>
</tr>
</tbody>
</table>

Additional constants are defined for use with specific subprograms.

See Also:

- Table 173-3 for more information about CSTORE related constants
- Table 173-4 for more information about predefined PTF method names
- Table 173-6 for more information about XSTORE related constants
- Supported Types Collections for more information about predefined collections of supported types
173.4 DBMS_TF Operational Notes

These operational notes describe the client and the server-side interfaces, and detail the compilation and execution statement management of Polymorphic Table Functions (PTF).

173.4.1 PTF Client Interface

The Polymorphic Table Function (PTF) implementation client interface is a set of subprograms with fixed names that every PTF must provide.

The PTF client interface can have up to four subprograms as follow:

- **DESCRIBE** function (Required)
- **OPEN** procedure (Optional)
- **FETCH_ROWS** procedure (Optional)
- **CLOSE** procedure (Optional)

The function **DESCRIBE** is invoked during SQL cursor compilation.

The procedures **OPEN**, **FETCH_ROWS**, and **CLOSE** are invoked during query execution.

The arguments to the implementation functions must match the PTF function with the following modifications:

1. Arguments of the type `TABLE` and `COLUMNS` are skipped for the execution procedures **OPEN**, **FETCH_ROWS**, and **CLOSE**.
2. The `TABLE` and `COLUMNS` arguments have descriptor types for the **DESCRIBE** function.
3. Scalar arguments that are not available during compilation are passed as NULL values (when using bind variables for instance). During execution, the actual values are passed in.

**DESCRIBE Function**

The **DESCRIBE** function is invoked to determine the type of rows (row shape) produced by the Polymorphic Table Function (PTF). It returns a `DBMS_TF.DESCRIBE_T` table.

The function **DESCRIBE** is invoked during SQL cursor compilation when a SQL query references a PTF. The SQL compiler locates the **DESCRIBE** function defined in the PTF implementation package. All the argument values from the query calling the PTF are passed to the **DESCRIBE** function. Like any PLSQL function, the **DESCRIBE** function can be overloaded and can have arguments default values.

The arguments of the PTF function and **DESCRIBE** function must match, but with the type of any **TABLE** argument replaced with the `DBMS_TF.TABLE_T` descriptor type, and the type of any **COLUMNS** argument replaced with `DBMS_TF.COLUMN_T` descriptor.

The **DESCRIBE** function indicates which columns must be kept by the database and passed unchanged as the PTF output (Pass-Through columns). In addition, the **DESCRIBE** function indicates any input columns that the PTF will use for its computation (Read columns).

Finally, the **DESCRIBE** function returns the list of any new columns that the PTF will create (or **NULL** if no new columns are being produced) using the `DBMS_TF.DESCRIBE_T` descriptor.
OPEN Procedure

The OPEN procedure purpose is to initialize and allocate any execution specific state. The OPEN procedure is most useful when you implement a Table Semantics PTF. The function typically calls the GET_XID function to get a unique ID for managing the execution state.

OPEN procedure is generally invoked before calling the FETCH_ROWS procedure.

FETCH_ROWS Procedure

The FETCH_ROWS procedure produces an output rowset that it sends to the database. The number of invocations of this function and the size of each rowset are data dependent and determined during query execution.

CLOSE Procedure

The CLOSE procedure is called at the end of the PTF execution. The procedure releases resources associated with the PTF execution state.

Example 173-1 Noop Polymorphic Table Function Example

This example creates a PTF called noop. This PTF returns the input rows as the output rows without any modification or filtering. Noop is one of the smallest PTF you can write.

```
CREATE PACKAGE noop_package AS
    FUNCTION describe(t IN OUT DBMS_TF.TABLE_T)
        RETURN DBMS_TF.DESCRIBE_T;

    PROCEDURE fetch_rows;
END noop_package;
```

The DESCRIBE function does not produce any new columns and hence, returns NULL. Executing FETCH_ROWS also results in NULL.

```
CREATE PACKAGE BODY noop_package AS
    FUNCTION describe(t IN OUT DBMS_TF.TABLE_T)
        RETURN DBMS_TF.DESCRIBE_T AS
        BEGIN
            RETURN NULL;
        END;

    PROCEDURE fetch_rows AS
```
BEGIN
  RETURN;
END;
END noop_package;

The noop PTF is defined to execute the noop_package when it is invoked.

CREATE FUNCTION noop (t TABLE)
  RETURN TABLE PIPELINED ROW POLYMORPHIC USING noop_package;

The PTF can be invoked in queries. For example:

SELECT *
FROM   NOOP(emp)
WHERE deptno = 10;

7782 CLARK      MANAGER         7839 09-JUN-81       2450        10
7839 KING       PRESIDENT            17-NOV-81       5000        10
7934 MILLER     CLERK           7782 23-JAN-82       1300        10

WITH e
  AS (SELECT *
       FROM   emp
       NATURAL JOIN dept
       WHERE dname = 'SALES')
SELECT t.*
FROM   NOOP(e) t;

30       7499 ALLEN      SALESMAN        7698 20-FEB-81       1600       300
SALES          CHICAGO
30       7521 WARD       SALESMAN        7698 22-FEB-81       1250       500
SALES          CHICAGO
30       7654 MARTIN     SALESMAN        7698 28-SEP-81       1250      1400
SALES          CHICAGO
30       7698 BLAKE      MANAGER         7839 01-MAY-81       2850
SALES          CHICAGO
30       7844 TURNER     SALESMAN        7698 08-SEP-81       1500         0
SALES          CHICAGO
30       7900 JAMES      CLERK           7698 03-DEC-81        950
SALES          CHICAGO

173.4.1.1 DESCRIBE Only Polymorphic Table Function

A Polymorphic Table Function (PTF) can have a DESCRIBE function only.

A PTF which does not have any runtime methods (Open/Fetch_Rows/Close) is used only at
cursor compilation time with no runtime row source allocated. The explain plan output of a
Describe-Only PTF will not show any rows for the PTF.

173.4.2 PTF Server Side Interface

The DBMS_TF package provides the server side interface needed for Polymorphic Table
Functions (PTFs) implementation to read and write information in the database.
This topic contains a partial list of types and subprograms used for the PTF server side implementation.

Table 173-2  Summary of Commonly Used Types and Subprograms in PTF Server Side Interface

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_METADATA_T</td>
<td>Column metadata record</td>
</tr>
<tr>
<td>COLUMN_T</td>
<td>Column descriptor record</td>
</tr>
<tr>
<td>TABLE_T</td>
<td>Table descriptor record</td>
</tr>
<tr>
<td>COLUMNS_T</td>
<td>Collection containing column names</td>
</tr>
<tr>
<td>COLUMNS_NEW_T</td>
<td>Collection for new columns</td>
</tr>
<tr>
<td>TAB_&lt;typ&gt;_T</td>
<td>Collection for each supported types, where &lt;typ&gt; is described in &quot;Supported Types Collections&quot;</td>
</tr>
<tr>
<td>ROW_SET_T</td>
<td>Data for a rowset record</td>
</tr>
<tr>
<td>GET_COL Procedure</td>
<td>Fetches data for a specified (input) column</td>
</tr>
<tr>
<td>PUT_COL Procedure</td>
<td>Returns data for a specified (new) column</td>
</tr>
<tr>
<td>GET_ROW_SET Procedure</td>
<td>Fetches the input rowset of column values</td>
</tr>
<tr>
<td>PUT_ROW_SET Procedure</td>
<td>Returns data for ALL (new) columns</td>
</tr>
<tr>
<td>SUPPORTED_TYPE Function</td>
<td>Verifies if a type is supported by DBMS_TF subprograms</td>
</tr>
<tr>
<td>GET_XID Function</td>
<td>Returns a unique execution ID to index PTF state in a session</td>
</tr>
</tbody>
</table>

See Also:
- DBMS_TF Data Structures for the complete list of types
- Summary of DBMS_TF Subprograms for the complete list of subprograms

173.4.3 Read Columns

Read columns are a set of table columns that the Polymorphic Table Function (PTF) processes when executing the FETCH_ROWS procedure.

The PTF indicates the read columns inside DESCRIBE by annotating them in the input table descriptor, TABLE_T. Only the indicated read columns will be fetched and thus available for processing during FETCH_ROWS.

The PTF invocation in a query will typically use the COLUMNS operator to indicate which columns the query wants the PTF to read, and this information is passed to the DESCRIBE function which then in turn sets the COLUMN_T.FOR_READ boolean flag.

Only scalar SQL data types are allowed for the read columns.

The Echo Polymorphic Table Function Example takes a table and a list of columns and produces new columns with the same values.
173.4.4 Pass-Through Columns

Pass-through columns are passed from the input table of the Polymorphic Table Function (PTF) to the output, without any modifications.

The **DESCRIBE** function indicates the pass-through columns by setting the **COLUMN_T.PASS_THROUGH** boolean flag on the input table descriptor, **DBMS_TF.TABLE_T**.

All columns in the **Row Semantics** PTF are marked as pass-through by default. For **Table Semantics** PTF, the default value for pass-through is set to false. For the **Table Semantics** PTF, the partitioning columns are always pass-through and this cannot be changed by the **DESCRIBE** function.

Note, the notions of Pass-Through and Read are orthogonal, and indicating a column as one has no implication for the other.

173.4.5 State Management

The database manages the compilation and execution states of the polymorphic table functions (PTF).

The database fulfills the PTF conductor role. As such, it is responsible for the PTF compilation state and execution state.

1. **Compilation State**: This is the immutable state that is generated by **DESCRIBE** which is needed before execution.
2. **Execution State**: This is the state used by the execution time procedures (**OPEN**, **FETCH_ROWS**, and **CLOSE**) of a **Table semantics** PTF.

The most common use of compilation state is to keep track of the columns to be read and the new columns that are to be produced. The PTF Server interface provides functions that can be used to achieve this: **GET_ENV**, and **GET_ROW_SET**. The PTF author who defines, documents, and implements the PTF can rely on the database to manage the PTF states. The PTF author should not attempt to use the session state (such as PL/SQL package global variables) to store any compilation state. Problems can arise because in a given session all cursors using the PTF will share that state, and other sessions executing the PTF cursor will not see the original compilation state.

Since the execution state is session and cursor private, a **Table Semantics** PTF can use package globals for storing execution state, but with the provision that the PTF uses the database provided unique execution ID to identify that state. The **GET_XID** function guarantees to provide an execution unique ID for the PTF’s execution procedures, where this ID remains constant for all the execution functions of a PTF.

173.4.5.1 CSTORE Compilation State Management

The CSTORE is the PTF compilation state management interface.

The CSTORE enables Polymorphic Table Functions (PTF) to store the compilation state in the SQL cursor.

The CSTORE interface is used to store key-value pairs during cursor compilation through the **DBMS_TF.DESCRIBE_T** record.
The compilation state information is retrieved during execution procedures such as OPEN, FETCH_ROWS and CLOSE.

**CSTORE Subprograms**

The CSTORE interface consists of the following subprograms.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSTORE_GET procedure</td>
<td>Fetches item of specified type. If not found, the OUT value remains unchanged.</td>
</tr>
<tr>
<td>CSTORE_EXISTS function</td>
<td>If an item with the given key exists in the CSTORE, this function returns TRUE.</td>
</tr>
</tbody>
</table>

**CSTORE Supported Types**

The DBMS_TF.DESCRIBE_T supports specifying key-value pairs for these scalar types: VARCHAR2, NUMBER, DATE, BOOLEAN.

**Table 173-3 DBMS_TF CSTORE Scalar Supported Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSTORE_TYPE_VARCHAR2</td>
<td>CSTORE VARCHAR2 type code</td>
</tr>
<tr>
<td>CSTORE_TYPE_NUMBER</td>
<td>CSTORE NUMBER type code</td>
</tr>
<tr>
<td>CSTORE_TYPE_DATE</td>
<td>CSTORE DATE type code</td>
</tr>
<tr>
<td>CSTORE_TYPE_BOOLEAN</td>
<td>CSTORE BOOLEAN type code</td>
</tr>
</tbody>
</table>

**Collections For Compilation Storage**

These predefined collection types are used for compilation state management.

```plaintext
TYPE CSTORE_CHR_T IS TABLE OF VARCHAR2(32767) INDEX BY VARCHAR2(32767);
TYPE CSTORE_NUM_T IS TABLE OF NUMBER INDEX BY VARCHAR2(32767);
TYPE CSTORE_BOL_T IS TABLE OF BOOLEAN INDEX BY VARCHAR2(32767);
TYPE CSTORE_DAT_T IS TABLE OF DATE INDEX BY VARCHAR2(32767);
```

**DBMS_TF Method Names**

The method names are also stored in the DBMS_TF.DESCRIBE_T record. These predefined values for the method names can be customized by the PTF author.

See [Method Name Overrides](#) for more information about changing the default method names.

**Table 173-4 DBMS_TF Method Names Constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE</td>
<td>DBMS_QUOTED_ID</td>
<td>'CLOSE'</td>
<td>Predefined index value for the method named CLOSE</td>
</tr>
</tbody>
</table>
Table 173-4  (Cont.) DBMS_TF Method Names Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FETCH_ROWS</td>
<td>DBMS_QUOTED_ID</td>
<td>'FETCH_ROWS'</td>
<td>Predefined index value for the method named FETCH_ROWS</td>
</tr>
<tr>
<td>OPEN</td>
<td>DBMS_QUOTED_ID</td>
<td>'OPEN'</td>
<td>Predefined index value for the method named OPEN</td>
</tr>
</tbody>
</table>

173.4.5.2 XSTORE Execution State Management

XSTORE is the PTF execution state management interface.

The XSTORE key-value interface simplifies the implementation of Table Semantics PTFs by providing automatic state management capabilities when the keys are strings and values are of commonly used scalar types.

The database automatically manages the deletion of all execution states allocated using this interface.

XSTORE Subprograms

The execution state management interface consists of the following subprograms.

Table 173-5  DBMS_TF XSTORE Subprograms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSTORE_CLEAR</td>
<td>procedure</td>
</tr>
<tr>
<td>XSTORE_EXISTS</td>
<td>function</td>
</tr>
<tr>
<td>XSTORE_GET</td>
<td>procedure</td>
</tr>
<tr>
<td>XSTORE_REMOVE</td>
<td>procedure</td>
</tr>
<tr>
<td>XSTORE_SET</td>
<td>procedure</td>
</tr>
</tbody>
</table>

XSTORE Predefined Types

The XSTORE supports specifying key-value pairs for these scalar types: VARCHAR2, NUMBER, DATE, and BOOLEAN.

Table 173-6  DBMS_TF XSTORE Scalar Supported Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSTORE_TYPE_VARCHAR2</td>
<td>XSTORE VARCHAR2 type code</td>
</tr>
<tr>
<td>XSTORE_TYPE_NUMBER</td>
<td>XSTORE NUMBER type code</td>
</tr>
</tbody>
</table>
### 173.4.6 Method Name Overrides

When multiple polymorphic table function (PTF) implementations are in the same package, you can override the default runtime method names (OPEN, FETCH_ROWS, and CLOSE) with your PTF specific names.

To override a method name, the application can specify the new method names using DBMS_TF.METHOD_NAMES collection (see DESCRIBE_T Record Type).

#### See Also:
Table 173-4

#### Example 173-2 DBMS_TF Method Name Overrides

This example shows how to change the default method name of the noop_p PTF fetch_rows method to noop_fetch.

#### Live SQL:
You can view and run this example on Oracle Live SQL at [DBMS_TF Method Name Overrides](https://example.com).

Create the PTF implementation package noop_p.

```sql
CREATE PACKAGE noop_p AS
    FUNCTION describe(tab IN OUT DBMS_TF.table_t) RETURN DBMS_TF.describe_t;
    PROCEDURE noop_fetch;
END noop_p;
```

To provide a method name override, you can specify the new method names using DBMS_TF.Method_Names collection. The FETCH_ROWS method name is changed to 'Noop_Fetch'. The procedure noop_fetch to implement this method is defined in the package.

```sql
CREATE OR replace PACKAGE BODY noop_p AS
    FUNCTION describe(tab IN OUT DBMS_TF.table_t) RETURN DBMS_TF.describe_t AS
```

---

**Table 173-6 (Cont.) DBMS_TF XSTORE Scalar Supported Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSTORE_TYPE_DATE</td>
<td>XSTORE DATE type code</td>
</tr>
<tr>
<td>XSTORE_TYPEBOOLEAN</td>
<td>XSTORE BOOLEAN type code</td>
</tr>
</tbody>
</table>
methods DBMS_TF.methods_t := DBMS_TF.methods_t(DBMS_TF.fetch_rows => 'Noop_Fetch');
BEGIN
  RETURN DBMS_TF.describe_t(method_names => methods);
END;
PROCEDURE noop_fetch AS
BEGIN
  RETURN;
END;
END noop_p;

The noop PTF is defined to execute the noop_p when it is invoked.

CREATE FUNCTION noop (t TABLE) RETURN TABLE PIPELINED ROW POLYMORPHIC USING noop_p;

The PTF is invoked in the FROM clause of a query block.

SELECT *
FROM noop(scott.emp)
WHERE deptno =10;

173.4.7 Using the COLUMNS Pseudo-Operator

The COLUMNS pseudo-operator is an addition to the SQL expression language.

Use the COLUMNS pseudo-operator to specify the arguments when invoking a Polymorphic Table Function (PTF) in the FROM clause. The COLUMNS pseudo-operator arguments specify the list of column names, or the list of column names with associated types.

See Also:
Oracle Database PL/SQL Language Reference for more information about the COLUMNS pseudo-operator syntax and semantics

173.4.8 Query Transformations
About predicate, projection and partitioning.

The pass-through columns of a Row Semantics PTF, and the PARTITION BY key columns of a Table Semantics PTF can be used for projection and predicate pushdown.

Example 173-3 Query Transformations
This example illustrates the predicate and projection pushdown for a Row Semantics PTF.

This query calls the echo PTF created in Echo Polymorphic Table Function Example.

SELECT empno, ename, sal, comm, echo_sal
FROM echo(emp, COLUMNS(sal,comm))
WHERE deptno = 30
    AND echo_sal > 1000;

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>SAL</th>
<th>COMM</th>
<th>ECHO_SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>1600</td>
<td>300</td>
<td>1600</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>1250</td>
<td>500</td>
<td>1250</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>1250</td>
<td>1400</td>
<td>1250</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>2850</td>
<td>0</td>
<td>2850</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>1500</td>
<td>0</td>
<td>1500</td>
</tr>
</tbody>
</table>

Conceptually, this query will get rewritten as:

WITH t AS (SELECT empno, ename, sal, comm
FROM emp
WHERE deptno=30)
SELECT empno, ename, sal, comm, echo_sal
FROM echo(t, COLUMNS(sal, comm))
WHERE echo_sal > 1000;

173.4.9 Parallel Execution

A key benefit of Polymorphic Table Functions (PTFs) is that their execution can be parallelized.

Row and table semantic PTFs execute in parallel differently.

Row Semantics PTF

Under Row Semantics PTF, the parallel query executes with the same degree of parallelism (DOP) as it would if the PTF were not present. The DOP is driven by the child row source.

Provided that the DOP on table emp has been set to 5, the following is an example that shows this parallelization:

EXPLAIN PLAN FOR
SELECT * FROM echo(emp, COLUMNS(ename, job))
WHERE deptno != 20;

Predicate Information (identified by operation id):
5 - filter("EMP"."DEPTNO"<>20)
Table Semantics PTF

Table Semantics PTF requires its input table rows to be redistributed using the `PARTITION BY` key. The parallel execution is determined by the `PARTITION BY` clause specified in the query.

173.5 DBMS_TF Execution Flow

Query executions invoking Polymorphic Table Functions (PTF) follow this execution model and data transfers flow.

The PTF execution procedures (`OPEN`, `FETCH_ROWS` and `CLOSE`) are called by the database during query execution.

The PTF execution follows this flow:

1. `OPEN` (if present)
2. `FETCH_ROWS` (can be invoked multiple times)
3. `CLOSE` (if present)
/* OPEN */

Open()

/* FETCH_ROWS (N calls) */

Fetch_Rows() #1

Get_Row_Set(Read_Rowset: OUT)

Read_Rowset

Computation/Algorithm

Put_Row_Set(Write_Rowset: IN)

Repeated calls to Fetch_Rows()

Fetch_Rows() #N

Get_Row_Set(Read_Rowset: OUT)

Read_Rowset

Computation/Algorithm

Put_Row_Set(Write_Rowset: IN)

/* CLOSE */

Close()
The `FETCH_ROWS` procedure reads the data for a rowset (collection of rows), and produces an output rowset.

Each call to `FETCH_ROWS` is associated with a rowset which is a data collection of input rows that is expected to be processed by the PTF.

The `GET_ROW_SET` or `GET_COL` is used to read the input rowset.

The `PUT_ROW_SET` or `PUT_COL` is used to produce an output rowset, that is written back to the database.

`PUT_ROW_SET` is used to set all the new columns in a single call.

The `ROWSET_T` record holds data for multiple columns. When the PTF algorithm is more suited toward producing a single output column at a time, you can use `PUT_COL` to produce a single column. A given column can only be produced once within a call to `FETCH_ROWS`.

For a Row Semantics PTF, the `FETCH_ROWS` procedure will return the new rows using the PTF Server interface before returning back to the database.

### 173.6 DBMS_TF Restrictions

These restrictions apply to Polymorphic Table Functions (PTFs) and using the DBMS_TF package.

**Type Restrictions**

A Polymorphic Table Function (PTF) can operate on a table with columns of any SQL types. However, read and new columns are restricted to scalar types. The read and new columns are used in the `PUT_ROW_SET`, `PUT_COL`, `GET_ROW_SET` and `GET_COL` procedures. All SQL types can be used with pass-through columns. The `DESCRIBE` function can determine the supported types using the `DBMS_TF.SUPPORTED_TYPE` function.

**PTF Invocation and Execution Restrictions**

Polymorphic table functions cannot be nested in the `FROM` clause of a query. Nesting PTF is only allowed using `WITH` clause.

Nesting table function with polymorphic table function is only allowed using `CURSOR` expressions. A PTF cannot be specified as an argument of a table function.

You cannot select a rowid from a Polymorphic Table Function (PTF).

The `PARTITION BY` and the `ORDER BY` clause can only be specified on an argument of a Table Semantics PTF.

The PTF execution methods `OPEN`, `FETCH_ROWS`, and `CLOSE` must be invoked in the polymorphic table function execution context only.

You cannot invoke the `DESCRIBE` method directly.

This example shows ten PTF nested invocation.

```sql
WITH t0
AS (SELECT /*+ parallel */ *
    FROM   noop(dept)),
```
173.7 DBMS_TF Examples

These examples use DBMS_TF subprograms.

Summary of DBMS_TF Examples

These examples are incomplete and for demonstration purpose only.

- Example 173-1, "Noop Polymorphic Table Function"
- Echo Polymorphic Table Function Example
- Example 173-2, "DBMS_TF Method Name Overrides"
- Example 173-3, "Query Transformations"
- Example 173-5, "DBMS_TF.COLUMN_TYPE_NAME Example"
- Example 173-6, "DBMS_TF.COL_TO_CHAR Example"
- Example 173-7, "DBMS_TF.CSTORE_EXISTS Example"
- Example 173-8, "DBMS_TF.GET_COL Example"
- Example 173-9, "DBMS_TF.GET_ENV Example"
173.7.1 Echo Polymorphic Table Function Example

The echo PTF takes in a table and a list of columns and produces new columns with same values.

This PTF returns all the columns in the input table tab, and adds to it the columns listed in cols but with the column names prefixed with "ECHO_".

Live SQL:

You can view and run this example on Oracle Live SQL at Echo Polymorphic Table Function

The echo PTF can appear in the FROM clause of the query. The COLUMNS operator is used to specify columns, for example:

```
SELECT *
FROM echo(scott.dept, COLUMNS(dname, loc));
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
<th>ECHO_DNAME</th>
<th>ECHO_LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
<td>RESEARCH</td>
<td>DALLAS</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
<td>SALES</td>
<td>CHICAGO</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
</tr>
</tbody>
</table>

A PTF consists of the following:

- PTF implementation package specification: The specification must have the DESCRIBE method. The OPEN, FETCH_ROWS and CLOSE methods are optional.
• **PTF implementation package body:** The DESCRIBE method may have a new-columns parameter (the additional columns created by this PTF), which is followed by the PTF functions parameters.

• **PTF Function:** The PTF function has a reference to the implementation package. The `echo_package` package specification defines the DESCRIBE and FETCH_ROWS methods.

```sql
CREATE PACKAGE echo_package
AS
  prefix DBMS_ID := 'ECHO_';
  FUNCTION describe(
    tab IN OUT DBMS_TF.TABLE_T,
    cols IN DBMS_TF.COLUMNS_T)
  RETURN DBMS_TF.DESCRIBE_T;
  PROCEDURE fetch_rows;
END echo_package;
```

The `echo_package` package body contains the PTF implementation.

```sql
CREATE PACKAGE BODY echo_package
AS
  FUNCTION describe(tab IN OUT DBMS_TF.TABLE_T,
                    cols IN DBMS_TF.COLUMNS_T)
  RETURN DBMS_TF.DESCRIBE_T
  AS
    new_cols DBMS_TF.COLUMNS_NEW_T;
    col_id   PLS_INTEGER := 1;
    BEGIN
      FOR I IN 1 .. tab.COLUMN.COUNT LOOP
        FOR J IN 1 .. cols.COUNT LOOP
          IF ( tab.COLUMN(i).description.name = cols(j) ) THEN
            IF ( NOT DBMS_TF.SUPPORTED_TYPE(tab.COLUMN(i).description.TYPE) ) THEN
              RAISE_APPLICATION_ERROR(-20102, 'Unsupported column type [' ||
                tab.COLUMN(i).description.TYPE || ']');
            END IF;
            TAB.COLUMN(i).for_read := TRUE;
            NEW_COLS(col_id) := TAB.COLUMN(i).description;
            NEW_COLS(col_id).name := prefix || TAB.COLUMN(i).description.name;
            col_id := col_id + 1;
            EXIT;
          END IF;
        END LOOP;
      END LOOP;

      /* Verify all columns were found */
      IF ( col_id - 1 != cols.COUNT ) THEN
        RAISE_APPLICATION_ERROR(-20101, 'Column mismatch ['||col_id -
l[['', ['|cols.COUNT|''']]);
  END IF;

  RETURN DBMS_TF.DESCRIBE_T(new_columns => new_cols);
END;
PROCEDURE FETCH_ROWS
AS
  ROWSET DBMS_TF.ROW_SET_T;
BEGIN
  DBMS_TF.GET_ROW_SET(rowset);
  DBMS_TF.PUT_ROW_SET(rowset);
END;
END echo_package;

The PTF echo references the implementation package echo_package.

CREATE FUNCTION echo(tab TABLE,
  cols COLUMNS)
RETURN TABLE
PIPELINED ROW POLYMORPHIC USING echo_package;

Example 173-4  Using the Echo PTF in Queries

This example selects all employees in department 20. The resulting rows have three new
columns ECHO_ENAME, ECHO_HIREDATE, and ECHO_SAL.

SELECT *
FROM   echo(scott.emp, COLUMNS(ename, sal, hiredate))
WHERE  deptno = 20;

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
<th>ECHO_ENAME</th>
<th>ECHO_HIRE</th>
<th>ECHO_SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>CLERK</td>
<td>7902</td>
<td>17-DEC-80</td>
<td>800</td>
<td></td>
<td>20</td>
<td>SMITH</td>
<td>17-DEC-80</td>
<td>800</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
<td>7839</td>
<td>02-APR-81</td>
<td>2975</td>
<td></td>
<td>20</td>
<td>JONES</td>
<td>02-APR-81</td>
<td>2975</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>7566</td>
<td>19-APR-87</td>
<td>3000</td>
<td></td>
<td>20</td>
<td>SCOTT</td>
<td>19-APR-87</td>
<td>3000</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>CLERK</td>
<td>7788</td>
<td>23-MAY-87</td>
<td>1100</td>
<td></td>
<td>20</td>
<td>ADAMS</td>
<td>23-MAY-87</td>
<td>1100</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
<td>7566</td>
<td>03-DEC-81</td>
<td>3000</td>
<td></td>
<td>20</td>
<td>FORD</td>
<td>03-DEC-81</td>
<td>3000</td>
</tr>
</tbody>
</table>

Using subquery W, display ENAME, ECHO_LOC and DNAME columns for all employees in
department 30 with a salary greater than 1000.

WITH w
  AS (SELECT e.*,
       dname,
       loc
       FROM scott.emp e,
            scott.dept d
       WHERE e.deptno = d.deptno)
SELECT ename,
       echo_loc,
       dname
       FROM echo(w, COLUMNS(sal, dname, loc, hiredate))
WHERE deptno = 30
    AND echo_sal > 1000;

<table>
<thead>
<tr>
<th>ENAME</th>
<th>ECHO_LOC</th>
<th>DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLEN</td>
<td>CHICAGO</td>
<td>SALES</td>
</tr>
<tr>
<td>WARD</td>
<td>CHICAGO</td>
<td>SALES</td>
</tr>
<tr>
<td>MARTIN</td>
<td>CHICAGO</td>
<td>SALES</td>
</tr>
<tr>
<td>BLAKE</td>
<td>CHICAGO</td>
<td>SALES</td>
</tr>
<tr>
<td>TURNER</td>
<td>CHICAGO</td>
<td>SALES</td>
</tr>
</tbody>
</table>

Using subquery \( w \), display ENAME and DNAME columns for all employees with a salary greater than 1000.

WITH \( w \) AS (SELECT e.*,
    dname,
    loc
    FROM scott.emp e,
    scott.dept d
    WHERE e.deptno = d.deptno)

SELECT echo_ename,
    dname
FROM echo(w, COLUMNS(loc, deptno, dname, ename)) e
WHERE ename IN (SELECT echo_ename
    FROM echo(scott.emp, COLUMNS(sal, deptno, ename, hiredate))
    WHERE deptno = e.echo_deptno
    AND sal > 1000);

<table>
<thead>
<tr>
<th>ECHO_ENAME</th>
<th>DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLEN</td>
<td>SALES</td>
</tr>
<tr>
<td>MILLER</td>
<td>ACCOUNTING</td>
</tr>
<tr>
<td>CLARK</td>
<td>ACCOUNTING</td>
</tr>
<tr>
<td>WARD</td>
<td>SALES</td>
</tr>
<tr>
<td>ADAMS</td>
<td>RESEARCH</td>
</tr>
<tr>
<td>TURNER</td>
<td>SALES</td>
</tr>
<tr>
<td>SCOTT</td>
<td>RESEARCH</td>
</tr>
<tr>
<td>BLAKE</td>
<td>SALES</td>
</tr>
<tr>
<td>JONES</td>
<td>RESEARCH</td>
</tr>
<tr>
<td>KING</td>
<td>ACCOUNTING</td>
</tr>
<tr>
<td>FORD</td>
<td>RESEARCH</td>
</tr>
<tr>
<td>MARTIN</td>
<td>SALES</td>
</tr>
</tbody>
</table>

173.8 DBMS_TF Data Structures

The DBMS_TF package defines these RECORD types, TABLE types and subtype.

**RECORD Types**

- COLUMN_DATA_T Record Type
- COLUMN_METADATA_T Record Type
- COLUMN_T Record Type
• DESCRIBE_T Record Type
• ENV_T Record Type
• PARALLEL_ENV_T Record Type
• TABLE_T Record Type

**TABLE Types**
- Supported Types Collections (TAB_<typ>_T)
- COLUMNS_NEW_T Table Type
- COLUMNS_T Table Type
- COLUMNS_WITH_TYPE_T Table Type
- TABLE_COLUMNS_T Table Type
- ROW_SET_T Table Type

**Types**
- XID_T Subtype

**CSTORE and XSTORE Data Structures**
The compilation and execution state management interfaces use data structures internally.
See Collections For Compilation Storage for more information.

### 173.8.1 Supported Types Collections
Each supported type has a corresponding predefined collection defined.

**Syntax**

```plaintext
TYPE TAB_BOOLEAN_T       IS TABLE OF BOOLEAN       INDEX BY PLS_INTEGER;
TYPE TAB_BINARY_FLOAT_T  IS TABLE OF BINARY_FLOAT  INDEX BY PLS_INTEGER;
TYPE TAB_BINARY_DOUBLE_T IS TABLE OF BINARY_DOUBLE INDEX BY PLS_INTEGER;
TYPE TAB_BLOB_T          IS TABLE OF BLOB          INDEX BY PLS_INTEGER;
TYPE TAB_CHAR_T          IS TABLE OF CHAR(32767)   INDEX BY PLS_INTEGER;
TYPE TAB_CLOB_T          IS TABLE OF CLOB          INDEX BY PLS_INTEGER;
TYPE TAB_DATE_T          IS TABLE OF DATE          INDEX BY PLS_INTEGER;
TYPE TAB_INTERVAL_YM_T   IS TABLE OF YMINTERVAL_UNCONSTRAINED INDEX BY PLS_INTEGER;
TYPE TAB_INTERVAL_DS_T   IS TABLE OF DSINTERVAL_UNCONSTRAINED INDEX BY PLS_INTEGER;
TYPE TAB_NATURALN_T      IS TABLE OF NATURALN      INDEX BY PLS_INTEGER;
TYPE TAB_NUMBER_T        IS TABLE OF NUMBER        INDEX BY PLS_INTEGER;
TYPE TAB_RAW_T           IS TABLE OF RAW(32767)    INDEX BY PLS_INTEGER;
TYPE TAB_ROWID_T         IS TABLE OF ROWID         INDEX BY PLS_INTEGER;
```
TYPE TAB_VARCHAR2_T IS TABLE OF VARCHAR2(32767) INDEX BY PLS_INTEGER;
TYPE TAB_TIMESTAMP_T IS TABLE OF TIMESTAMP_UNCONSTRAINED INDEX BY PLS_INTEGER;
TYPE TAB_TIMESTAMP_TZ_T IS TABLE OF TIMESTAMP_TZ_UNCONSTRAINED INDEX BY PLS_INTEGER;
TYPE TAB_TIMESTAMP_LTZ_T IS TABLE OF TIMESTAMP_LTZ_UNCONSTRAINED INDEX BY PLS_INTEGER;

See Also: Table 173-1 for more information about the DBMS_TF supported types

173.8.2 COLUMN_DATA_T Record Type

Data for a single column (variant record).

Exactly one variant field is active in the record. The description includes information about the column type that is active.

See Table 173-1 for the list of supported types.

Syntax

```plsql
TYPE COLUMN_DATA_T IS RECORD
  ( description            COLUMN_METADATA_T,
    tab_varchar2           TAB_VARCHAR2_T,
    tab_number             TAB_NUMBER_T,
    tab_date               TAB_DATE_T,
    tab_binary_float       TAB_BINARY_FLOAT_T,
    tab_binary_double      TAB_BINARY_DOUBLE_T,
    tab_raw                TAB_RAW_T,
    tab_char               TAB_CHAR_T,
    tab_clob               TAB_CLOB_T,
    tab_blob               TAB_BLOB_T,
    tab_timestamp          TAB_TIMESTAMP_T,
    tab_timestamp_tz       TAB_TIMESTAMP_TZ_T,
    tab_interval_ym        TAB_INTERVAL_YM_T,
    tab_interval_ds        TAB_INTERVAL_DS_T,
    tab_timestamp_ltz      TAB_TIMESTAMP_LTZ_T,
    tab_rowid              TAB_ROWID_T);
```

Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>The tag defines the metadata for the column indicating which variant field is active.</td>
</tr>
<tr>
<td>tab_varchar2</td>
<td>Variant field</td>
</tr>
</tbody>
</table>

Table 173-7 COLUMN_DATA_T Fields
### 173.8.3 COLUMN_METADATA_T Record Type

This type contains metadata about an existing table column or a new column produced by PTF.

#### Syntax

```sql
TYPE COLUMN_METADATA_T IS RECORD
  ( type               PLS_INTEGER,
    max_len            PLS_INTEGER DEFAULT -1,
    name               VARCHAR2(32767),
    name_len           PLS_INTEGER,
    precision          PLS_INTEGER,
    scale              PLS_INTEGER,
    charsetid          PLS_INTEGER,
    charsetform        PLS_INTEGER,
    collation          PLS_INTEGER );
```

#### Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Internal Oracle typecode for the column's type</td>
</tr>
<tr>
<td>max_len</td>
<td>Maximum length of a column. If it is less than the maximum allowed length then that value will be used, if it is NULL or zero, zero will be used. If it is less than zero, then maximum allowed length will be used. If types (like date, float), does not care about length, then this value will be ignored.</td>
</tr>
</tbody>
</table>
Table 173-8  (Cont.) COLUMN_METADATA_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the column</td>
</tr>
<tr>
<td>name_len</td>
<td>Length of the name</td>
</tr>
<tr>
<td>precision</td>
<td>The precision, or the maximum number of significant decimal digits (for numeric data types)</td>
</tr>
<tr>
<td>scale</td>
<td>Scale, or the number of digits from the decimal point to the least significant digit (for numeric data types)</td>
</tr>
<tr>
<td>charsetid</td>
<td>Character set id (internal Oracle code, applies to string types)</td>
</tr>
<tr>
<td>charsetform</td>
<td>Character set form (internal Oracle code, applies to string types)</td>
</tr>
<tr>
<td>collation</td>
<td>Collation id (internal Oracle code, applies to string types)</td>
</tr>
</tbody>
</table>

173.8.4 COLUMN_T Record Type

The column descriptor record for the type COLUMN_METADATA_T that contains PTF specific attributes.

Syntax

```plaintext
TYPE column_t IS RECORD (
  description            COLUMN_METADATA_T,
  pass_through           BOOLEAN,
  for_read               BOOLEAN);
```

Fields

Table 173-9  COLUMN_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Column metadata</td>
</tr>
<tr>
<td>pass_through</td>
<td>Is this a pass through column</td>
</tr>
<tr>
<td>for_read</td>
<td>Is this column read by the PTF</td>
</tr>
</tbody>
</table>

173.8.5 DESCRIBE_T Record Type

The return type from the DESCRIBE method of PTF.

Syntax

```plaintext
TYPE DESCRIBE_T IS RECORD
  ( NEW_COLUMNS        COLUMNS_NEW_T DEFAULT COLUMNS_NEW_T(),
    CSTORE_CHR         CSTORE_CHR_T  DEFAULT CSTORE_CHR_T(),
    CSTORE_NUM         CSTORE_NUM_T  DEFAULT CSTORE_NUM_T(),
    CSTORE_BOL         CSTORE_BOL_T  DEFAULT CSTORE_BOL_T(),
    CSTORE_DAT         CSTORE_DAT_T  DEFAULT CSTORE_DAT_T(),
    METHOD_NAMES       METHODS_T     DEFAULT METHODS_T());
```
Fields

Table 173-10  DESCRIBE_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_COLUMNS</td>
<td>New columns description that will be produced by the PTF</td>
</tr>
<tr>
<td>CSTORE_CHR</td>
<td>CStore array key type : VARCHAR2 (optional)</td>
</tr>
<tr>
<td>CSTORE_NUM</td>
<td>CStore array key type : NUMBER (optional)</td>
</tr>
<tr>
<td>CSTORE_BOL</td>
<td>CStore array key type : BOOLEAN (optional)</td>
</tr>
<tr>
<td>CSTORE_DAT</td>
<td>CStore array key type : DATE (optional)</td>
</tr>
<tr>
<td>METHOD_NAMES</td>
<td>Method names, if user wants to override OPEN, FETCH_ROWS, CLOSE methods</td>
</tr>
</tbody>
</table>

173.8.6 ENV_T Record Type

This record contains metadata about the polymorphic table function execution state.

Syntax

```plaintext
TYPE ENV_T IS RECORD (  
  get_columns      TABLE_METADATA_T,  
  put_columns      TABLE_METADATA_T,  
  ref_put_col      REFERENCED_COLS_T,  
  parallel_env     PARALLEL_ENV_T,  
  query_optim      BOOLEAN,  
  row_count        PLS_INTEGER,  
  row_replication  BOOLEAN,  
  row_insertion    BOOLEAN);  
```

Fields

Table 173-11  ENV_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_columns</td>
<td>Metadata about the columns read by PTF GET_COL procedure</td>
</tr>
<tr>
<td>put_columns</td>
<td>Metadata about columns sent back to database by PUT_COL procedure</td>
</tr>
<tr>
<td>ref_put_col</td>
<td>TRUE if the put column was referenced in the query</td>
</tr>
<tr>
<td>parallel_env</td>
<td>Parallel execution information (when a query runs in parallel)</td>
</tr>
<tr>
<td>query_optim</td>
<td>Is this execution for query optimization? TRUE, if the query was running on behalf of optimizer</td>
</tr>
<tr>
<td>row_count</td>
<td>Number of rows in current row set</td>
</tr>
<tr>
<td>row_replication</td>
<td>Is Row Replication Enabled?</td>
</tr>
<tr>
<td>row_insertion</td>
<td>Is Row Insertion Enabled?</td>
</tr>
</tbody>
</table>

173.8.7 PARALLEL_ENV_T Record Type

The record contains metadata specific to polymorphic table functions parallel execution.
Syntax

TYPE PARALLEL_ENV_T IS RECORD
  ( instance_id      PLS_INTEGER,
    session_id       PLS_INTEGER,
    slave_svr grp   PLS_INTEGER,
    slave_set_no     PLS_INTEGER,
    no_slocal_slaves PLS_INTEGER,
    global_slave_no PLS_INTEGER,
    no_local_slaves  PLS_INTEGER,
    local_slave_no  PLS_INTEGER );

Fields

Table 173-12 PARALLEL_ENV_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance_id</td>
<td>QC instance ID</td>
</tr>
<tr>
<td>session_id</td>
<td>QC session ID</td>
</tr>
<tr>
<td>slave_svr grp</td>
<td>Slave server group</td>
</tr>
<tr>
<td>slave_set_no</td>
<td>Slave server set number</td>
</tr>
<tr>
<td>no_slocal_slaves</td>
<td>Number of sibling slaves (including self)</td>
</tr>
<tr>
<td>global_slave_no</td>
<td>Global slave number (base 0)</td>
</tr>
<tr>
<td>no_local_slaves</td>
<td>Number of sibling slaves running on instance</td>
</tr>
<tr>
<td>local_slave_no</td>
<td>Local slave number (base 0)</td>
</tr>
</tbody>
</table>

173.8.8 TABLE_T Record Type

The DESCRIBE function input table descriptor argument is of TABLE_T record type.

Syntax

TYPE TABLE_T IS RECORD(
  column   TABLE_COLUMNS_T,
  schema_name   DBMS_id,
  package_name DBMS_id,
  ptf_name DBMS_id);

Fields

Table 173-13 TABLE_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>Column information</td>
</tr>
<tr>
<td>schema_name</td>
<td>The PTF schema name</td>
</tr>
<tr>
<td>package_name</td>
<td>The PTF implementation package name</td>
</tr>
<tr>
<td>ptf_name</td>
<td>The PTF name invoked</td>
</tr>
</tbody>
</table>
173.8.9 COLUMNS_NEW_T Table Type

Collection for new columns

Syntax

```
TYPE COLUMNS_NEW_T IS TABLE OF COLUMN_METADATA_T INDEX BY PLS_INTEGER;
```

173.8.10 COLUMNS_T Table Type

Collection containing column names

Syntax

```
TYPE COLUMNS_T IS TABLE OF DBMS_QUOTED_ID;
```

173.8.11 COLUMNS_WITH_TYPE_T Table Type

Collection containing columns metadata

Syntax

```
TYPE COLUMNS_WITH_TYPE_T IS TABLE OF COLUMN_METADATA_T;
```

173.8.12 TABLE_COLUMNS_T Table Type

A collection of columns(COLUMN_T)

Syntax

```
TYPE TABLE_COLUMNS_T IS TABLE OF COLUMN_T;
```

173.8.13 ROW_SET_T Table Type

Data for a rowset

Syntax

```
TYPE ROW_SET_T IS TABLE OF COLUMN_DATA_T INDEX BY PLS_INTEGER;
```

173.8.14 XID_T Subtype

The XID_T subtype is defined to store the execution unique ID returned by function GET_XID.

Syntax

```
SUBTYPE XID_T IS VARCHAR2(1024);
```
173.9 Summary of DBMS_TF Subprograms

This summary briefly describes the DBMS_TF package subprograms.

Table 173-14   DBMS_TF Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_TYPE_NAME</td>
<td>Returns the type name of the specified column type</td>
</tr>
<tr>
<td>COL_TO_CHAR</td>
<td>Returns the string representation of the specified column</td>
</tr>
<tr>
<td>CSTORE_EXISTS</td>
<td>Returns TRUE if an item with a given key exists in the PTF Compilation State management Store</td>
</tr>
<tr>
<td>CSTORE_GET</td>
<td>Gets item(s) of specified type from the PTF Compilation State management Store</td>
</tr>
<tr>
<td>GET_COL</td>
<td>Gets read column values</td>
</tr>
<tr>
<td>GET_ENV</td>
<td>Returns information about the PTF runtime environment</td>
</tr>
<tr>
<td>GET_ROW_SET</td>
<td>Gets read set of column values in the collection</td>
</tr>
<tr>
<td>GET_XID</td>
<td>Returns a unique execution id that can be used by the PTF to index any cursor execution specific runtime state</td>
</tr>
<tr>
<td>PUT_COL</td>
<td>Puts column values in the database</td>
</tr>
<tr>
<td>PUT_ROW_SET</td>
<td>Puts the collection read set of column values in the database</td>
</tr>
<tr>
<td>ROW_REPLICATION</td>
<td>Sets the row replication factor</td>
</tr>
<tr>
<td>ROW_TO_CHAR</td>
<td>Returns the string representation of a row in a rowset</td>
</tr>
<tr>
<td>SUPPORTED_TYPE</td>
<td>Returns TRUE if a specified type is supported by PTF infrastructure</td>
</tr>
<tr>
<td>TRACE</td>
<td>Prints data structures to help development and problem diagnosis</td>
</tr>
<tr>
<td>XSTORE_CLEAR</td>
<td>Removes all key-value pairs from XStore</td>
</tr>
<tr>
<td>XSTORE_EXISTS</td>
<td>Returns TRUE if the key has an associated value</td>
</tr>
<tr>
<td>XSTORE_GET</td>
<td>Gets a key-value store for PTF Execution State Management</td>
</tr>
<tr>
<td>XSTORE_REMOVE</td>
<td>Removes any value associated with the given key</td>
</tr>
<tr>
<td>XSTORE_SET</td>
<td>Sets the value for the given key store for PTF Execution State Management</td>
</tr>
</tbody>
</table>

173.9.1 COLUMN_TYPE_NAME Function

Returns the type name for the specified column type.
Syntax

FUNCTION COLUMN_TYPE_NAME(
    col COLUMN_METADATA_T)
RETURN VARCHAR2;

Parameters

Table 173-15  DBMS_TF.COLUMN_TYPE_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>The column metadata. See COLUMN_METADATA_T Record Type</td>
</tr>
</tbody>
</table>

Return Values

Returns the column type converted as text.

Example 173-5  DBMS_TF.COLUMN_TYPE_NAME Example

This example shows an application type check that invokes COLUMN_TYPE_NAME to compare the column type and raise an application error if the column type is not VARCHAR2.

FUNCTION describe(
    tab   IN OUT DBMS_TF.table_t,
    cols  IN     DBMS_TF.columns_t)
RETURN DBMS_TF.describe_t
AS
    new_cols DBMS_TF.columns_new_t;
    col_id    PLS_INTEGER := 1;
BEGIN
    FOR i IN 1 .. tab.count LOOP
        FOR j IN 1 .. cols.count LOOP
            IF (tab(i).description.name = cols(j)) THEN
                IF (DBMS_TF.column_type_name(tab(i).description.type) != 'VARCHAR2') THEN
                    raise_application_error(-20102, 'Unsupported column type ['||tab(i).description.type||']');
                END IF;
                tab(i).for_read := true;
                new_cols(col_id) := tab(i).description;
                new_cols(col_id).name := 'ECHO_'|| tab(i).description.name;
                col_id := col_id + 1;
                EXIT;
            END IF;
        END LOOP;
    END LOOP;

    -- Verify all columns were found
    IF (col_id - 1 != cols.count) THEN
        raise_application_error(-20101, 'Column mismatch ['||col_id-1||'], ['||cols.count||']);
    END IF;

    RETURN DBMS_TF.describe_t(new_columns => new_cols);
END;
173.9.2 COL_TO_CHAR Function

Returns the string representation of the specified column.

Syntax

FUNCTION COL_TO_CHAR(
    col   COLUMN_DATA_T,
    rid   PLS_INTEGER,
    quote VARCHAR2 DEFAULT '"')
RETURN VARCHAR2;

Parameters

Table 173-16  DBMS_TF.COL_TO_CHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>The column whose value is to be converted</td>
</tr>
<tr>
<td>rid</td>
<td>Row number</td>
</tr>
<tr>
<td>quote</td>
<td>Quotation mark to use for non-numeric values</td>
</tr>
</tbody>
</table>

Return Values

The string representation of a column data value.

Example 173-6  DBMS_TF.COL_TO_CHAR Example

PROCEDURE Fetch_Rows AS
    rowset DBMS_TF.rROW_SET_T;
    str VARCHAR2(32000);
BEGIN
    DBMS_TF.GET_ROW_SET(rowset);
    str := DBMS_TF.COL_TO_CHAR(rowset(1), 1)
END;

173.9.3 CSTORE_EXISTS Function

Returns TRUE if an item with a given key exists in the Store PTF Compilation State.

Syntax

FUNCTION CSTORE_EXISTS
    (key IN VARCHAR2,
     key_type IN PLS_INTEGER default NULL)
return BOOLEAN;
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>A unique character key</td>
</tr>
<tr>
<td>key_type</td>
<td>The type of key (optional)</td>
</tr>
</tbody>
</table>

Parameter key_type is optional. The default value of key_type is NULL.

Return Values

Returns TRUE if the key has an associated value. When the key_type is NULL (default), it returns TRUE if the key has an associated value of any of the supported type.

When a key_type parameter value is passed, it returns TRUE if the key and specified type of key has an associated value. Otherwise, it returns FALSE.

Example 173-7  DBMS_TF.CSTORE_EXISTS Example

This code excerpt checks if an item with the key exists before reading it from the compilation store.

```
IF (DBMS_TF.CSTORE_EXISTS('min'||j)) THEN
    DBMS_TF.CSTORE_GET('min'||j, min_col);
END IF;
```

173.9.4 CSTORE_GET Procedure

You can use the CSTORE_GET procedure to get the associated value for a given key stored for PTF Compilation State.

CSTORE is the PTF compilation state management interface. The CSTORE interface is used to set and store key-value pairs during cursor compilation through the DBMS_TF.DESCRIBE function.

You can get the PTF compilation state during runtime procedures such as OPEN, FETCH_ROWS and CLOSE.

This procedure is overloaded. The DESCRIBE_T supports specifying key-value pairs for these scalar types: VARCHAR2, NUMBER, DATE, BOOLEAN.

See Table 173-3 for more information.

Syntax

Get the value associated with the key in the value out variable. The value type returned is one of the supported scalar types.

```
PROCEDURE CSTORE_GET(
    key   IN     VARCHAR2,
    value IN OUT VARCHAR2);
```

PROCEDURE CSTORE_GET;
PROCEDURE CSTORE_GET(
    key   IN     VARCHAR2,
    value IN OUT NUMBER);

PROCEDURE CSTORE_GET(
    key   IN     VARCHAR2,
    value IN OUT DATE);

PROCEDURE CSTORE_GET(
    key   IN     VARCHAR2,
    value IN OUT BOOLEAN);

When no specific key is passed as an input parameter, the entire collection of key
values for that type that exist in the CSTORE is returned.

PROCEDURE CSTORE_GET(key_value OUT CSTORE_CHR_T);
PROCEDURE CSTORE_GET(key_value OUT CSTORE_NUM_T);
PROCEDURE CSTORE_GET(key_value OUT CSTORE_BOL_T);
PROCEDURE CSTORE_GET(key_value OUT CSTORE_DAT_T);

Parameters

Table 173-18  DBMS_TF.CSTORE_GET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>A unique character key</td>
</tr>
<tr>
<td>value</td>
<td>Value corresponding to the key for supported types</td>
</tr>
<tr>
<td>key_value</td>
<td>Key value</td>
</tr>
</tbody>
</table>

173.9.5 GET_COL Procedure

Get Read Column Values

Syntax

PROCEDURE GET_COL(
    columnId NUMBER,
    collection IN OUT NOCOPY <datatype>);

Where `<datatype>` can be any one of the supported types.

See Table 173-1 for the list of supported types.

Parameters

Table 173-19  GET_COL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnid</td>
<td>The id for the column</td>
</tr>
</tbody>
</table>
Table 173-19  (Cont.) GET_COL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>The data for the column</td>
</tr>
</tbody>
</table>

Usage Notes
This procedure is used to get the read column values in the collection of scalar type.
The column numbers are in the get column order as created in DESCRIBE method of PTF.
For the same ColumnId, GET_COL and PUT_COL may correspond to different column.

Example 173-8  DBMS_TF.GET_COL Example
This example is an excerpt of a fetch_rows procedure defined in the PTF implementation package.

PROCEDURE fetch_rows
IS
  col1 DBMS_TF.TAB_CLOB_T;
  col2 DBMS_TF.TAB_CLOB_T;
  out1 DBMS_TF.TAB_CLOB_T;
  out2 DBMS_TF.TAB_CLOB_T;
BEGIN
  DBMS_TF.GET_COL(1, col1);
  DBMS_TF.GET_COL(2, col2);
  FOR I IN 1 .. col1.COUNT LOOP
    out1(i) := 'ECHO-' || col1(i);
  END LOOP;
  FOR I IN 1 .. col2.COUNT LOOP
    out2(i) := 'ECHO-' || col2(i);
  END LOOP;
  DBMS_TF.PUT_COL(1, out1);
  DBMS_TF.PUT_COL(2, out2);
END;

Note, invoking the DBMS_TF APIs directly is not allowed. An error is raised if an attempt is made to execute these procedures out of context.

exec fetch_rows

ERROR at line 1:
ORA-62562: The API Get_Col can be called only during execution time of a polymorphic table function.

173.9.6 GET_ENV Function
Returns information about the PTF runtime environment
Syntax

FUNCTION GET_ENV
    RETURN ENV_T;

Return Values

Returns information about the PTF runtime environment.

Example 173-9 DBMS_TF.GET_ENV Example

This line shows how you could initialize a local variable env of type ENV_T with the PTF execution information in a FETCH_ROWS implementation procedure.

    env         DBMS_TF.ENV_T  := DBMS_TF.GET_ENV();

173.9.7 GET_ROW_SET Procedure

Get Read Column Values

The FETCH_ROW procedure can call the GET_ROW_SET procedure to read the input rowset set of column values in the collection of supported scalar type. This procedure is overloaded.

Syntax

PROCEDURE GET_ROW_SET(
    rowset    OUT NOCOPY ROW_SET_T);

PROCEDURE GET_ROW_SET(
    rowset    OUT NOCOPY ROW_SET_T,
    row_count OUT        PLS_INTEGER);

PROCEDURE GET_ROW_SET(
    rowset    OUT NOCOPY ROW_SET_T,
    row_count OUT        PLS_INTEGER,
    col_count OUT        PLS_INTEGER);

Parameters

Table 173-20 GET_ROW_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowset</td>
<td>The collection of data and metadata</td>
</tr>
<tr>
<td>row_count</td>
<td>The number of rows in the columns</td>
</tr>
<tr>
<td>col_count</td>
<td>The number of columns</td>
</tr>
</tbody>
</table>

Example 173-10 DBMS_TF.GET_ROW_SET Example

This example is an excerpt from a PTF implementation package for demonstration purpose.

    PROCEDURE fetch_rows(new_name IN VARCHAR2 DEFAULT 'PTF_CONCATENATE')
    AS
rowset DBMS_TF.ROW_SET_T;
accumulator DBMS_TF.TAB_VARCHAR2_T;
row_count PLS_INTEGER;

FUNCTION get_value(col PLS_INTEGER, 
                    ROW PLS_INTEGER) 
RETURN VARCHAR2 
AS 
  col_type PLS_INTEGER := rowset(col).description.TYPE;
BEGIN 
  CASE col_type 
    WHEN DBMS_TF.TYPE_VARCHAR2 THEN 
      RETURN NVL(rowset(col).TAB_VARCHAR2 (ROW), 'empty'); 
    ELSE 
      RAISE_APPLICATION_ERROR(-20201, 'Non-Varchar Type='||col_type);
    END CASE;
  END CASE;
END;
BEGIN 
  DBMS_TF.GET_ROW_SET(rowset, row_count);
  IF ( rowset.count = 0 ) THEN 
    RETURN;
  END IF;
  FOR row_num IN 1 .. row_count LOOP 
    accumulator(row_num) := 'empty';
  END LOOP;
  FOR col_num IN 1 .. rowset.count LOOP 
    FOR row_num IN 1 .. row_count LOOP 
      accumulator(row_num) := accumulator(row_num) || 
        get_value(col_num, row_num);
    END LOOP;
  END LOOP;
  DBMS_TF.PUT_COL(1, accumulator);
END;

173.9.7.1 Stack Polymorphic Table Function Example

The stack PTF example unpivots the non-null values of the specified numeric columns by converting each column value into a new row.

Example 173-11  Stack Polymorphic Table Function Example

Live SQL:

You can view and run this example on Oracle Live SQL at Stack Polymorphic Table Function

Create the PTF implementation package stack_p.
The parameters are:

- **tab** - Input table
- **col** - The names of numeric (input) table columns to stack

CREATE PACKAGE stack_p AS

FUNCTION  describe(tab  IN OUT dbms_tf.table_t,
                   col         dbms_tf.columns_t)
           RETURN dbms_tf.describe_t;

PROCEDURE fetch_rows;

END stack_p;

Create the PTF implementation package body stack_p.

This PTF produces two new columns, **COLUMN_NAME** and **COLUMN_VALUE**, where the former contains the name of the unpivoted column and the latter contains the numeric value of that column. Additionally, the unpivoted columns are removed from the PTF’s output.

CREATE PACKAGE BODY stack_p AS

FUNCTION  describe(tab  IN OUT dbms_tf.table_t,
                   col         dbms_tf.columns_t)
           RETURN dbms_tf.describe_t  AS

BEGIN
  FOR i IN 1 .. tab.column.count LOOP
    FOR j IN 1 .. col.count LOOP
      IF (tab.column(i).description.name = col(j) AND
          tab.column(i).description.TYPE = dbms_tf.type_number) THEN
        tab.column(i).pass_through := false;
        tab.column(i).for_read     := true;
      END IF;
    END LOOP;
  END LOOP;

  RETURN dbms_tf.describe_t(
    new_columns => dbms_tf.columns_new_t(
      1 => dbms_tf.column_metadata_t(name => 'COLUMN_NAME',
          TYPE => dbms_tf.type_varchar2),
      2 => dbms_tf.column_metadata_t(name => 'COLUMN_VALUE',
          TYPE => dbms_tf.type_number)),
    row_replication => true);

END;

PROCEDURE fetch_rows  AS
  env    dbms_tf.env_t := dbms_tf.get_env();
  rowset dbms_tf.row_set_t;
  colcnt PLS_INTEGER;
  rowcnt PLS_INTEGER;

  ...
repfac dbms_tf.tabnaturaln_t;
namcol dbms_tf.tabvarchar2_t;
valcol dbms_tf.tabnumber_t;
BEGIN
  dbms_tf.get_row_set(rowset, rowcnt, colcnt);
  FOR i IN 1 .. rowcnt LOOP
    repfac(i) := 0;
  END LOOP;
  FOR r IN 1 .. rowcnt LOOP
    FOR c IN 1 .. colcnt LOOP
      IF rowset(c).tab_number(r) IS NOT NULL THEN
        repfac(r)                    := repfac(r) + 1;
        namcol(nvl(namcol.last+1,1)) :=
          INITCAP(regexp_replace(env.get_columns(c).name, '^"|"$'));
        valcol(NVL(valcol.last+1,1)) := rowset(c).tab_number(r);
      END IF;
    END LOOP;
  END LOOP;
  dbms_tf.row_replication(replication_factor => repfac);
  dbms_tf.put_col(1, namcol);
  dbms_tf.put_col(2, valcol);
END;
END stack_p;

Create the standalone PTF named stack. Specify exactly one formal argument of type TABLE,
specify the return type of the PTF as TABLE, specify a Row Semantics PTF type, and indicate
the PTF implementation package to use is stack_p.

CREATE FUNCTION stack(tab TABLE,
                        col columns)
    RETURN TABLE
               PIPELINED ROW POLYMORPHIC USING stack_p;

For all employees in departments 10 and 30, report values of columns MGR, SAL, and COMM
ordered by department number and employee name.

SELECT   deptno, ename, column_name, column_value
FROM     stack(scott.emp, COLUMNS(mgr, sal, comm))
WHERE    deptno IN (10, 30)
ORDER BY deptno, ename;

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>ENAME</th>
<th>COLUMN_NAME</th>
<th>COLUMN_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>CLARK</td>
<td>Mgr</td>
<td>7839</td>
</tr>
<tr>
<td>10</td>
<td>CLARK</td>
<td>Sal</td>
<td>2450</td>
</tr>
<tr>
<td>10</td>
<td>KING</td>
<td>Sal</td>
<td>5000</td>
</tr>
<tr>
<td>10</td>
<td>MILLER</td>
<td>Sal</td>
<td>1300</td>
</tr>
<tr>
<td>10</td>
<td>MILLER</td>
<td>Mgr</td>
<td>7782</td>
</tr>
<tr>
<td>30</td>
<td>ALLEN</td>
<td>Comm</td>
<td>300</td>
</tr>
</tbody>
</table>
### 173.9.8 GET_XID Function

Returns a unique execution id that can be used by the PTF to index any cursor-execution specific runtime state.

#### Syntax

```
FUNCTION GET_XID
    RETURN XID_T;
```

#### Return Values

A unique execution id that can be used by the PTF to index any cursor-execution specific runtime state.

#### Example 173-12 DBMS_TF.GET_XID Example

This is an excerpt of code showing an invocation of GET_XID to initialize a local variable indexed using the execution id to a zero value.

```sql
PROCEDURE open IS
BEGIN
    xst(DBMS_TF.GET_XID()) := 0;
END;
```

### 173.9.9 PUT_COL Procedure

Put Column Values

#### Syntax

```
PROCEDURE PUT_COL(
    columnid NUMBER,
    collection IN <datatype>);
```

Where `<datatype>` can be any one of the supported types.

See Table 173-1 for the list of supported types.
Parameters

Table 173-21 PUT_COL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnid</td>
<td>The id for the column</td>
</tr>
<tr>
<td>collection</td>
<td>The data for the column</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is used to put the read column values in the collection of scalar type. The collection of scalar type should be of supported type only. The column numbers are in the get column order as created in `DESCRIBE` method of PTF. For the same columnid, `GET_COL` and `PUT_COL` may correspond to different column.

173.9.9.1 Rand_col Polymorphic Table Function Example

The `rand_col` PTF appends specified number of random-valued columns to the output.

Example 173-13 Rand_col Polymorphic Table Function Example

```
Live SQL:

You can view and run this example on Oracle Live SQL at Rand_col Polymorphic Table Function

This `rand_col` PTF example appends `col_count` number of random-valued columns to the output. Optionally, the caller can restrict the random values to a numeric range by specifying `[low, high]`. The new columns are named "RAND_<n>"

Create the PTF implementation package `rand_col_p`.

The parameters are:

- `tab`: Input table
- `col_count` (optional): Number of random-valued columns to generate [Default = 1]
- `low` (optional): Lower bound for the random numbers [Default = Null]
- `high` (optional): Upper bound for the random numbers [Default = Null]

CREATE PACKAGE rand_col_p AS

FUNCTION describe(tab IN OUT DBMS_TF.table_t, 
                    col_count NATURALN DEFAULT 1, 
                    low NUMBER DEFAULT NULL, 
                    high NUMBER DEFAULT NULL) 
    RETURN DBMS_TF.describe_t;
PROCEDURE fetch_rows(col_count NATURALN DEFAULT 1,  
   low       NUMBER   DEFAULT NULL,  
   high      NUMBER   DEFAULT NULL);  
END rand_col_p;

Create the PTF implementation package body rand_col_p.

The parameter col_count is a 'shape-determining' parameter and thus must be a constant (no binds, correlations, or expressions). By defining the type of col_count to be NATURALN, which has an implicit NOT NULL constraint, we guarantee that a cursor with non-constant value for this parameter will get a compilation error.

CREATE PACKAGE BODY rand_col_p AS  
  col_name_prefix CONSTANT dbms_id := 'RAND_';  
  
  FUNCTION describe(tab IN OUT DBMS_TF.table_t,  
    col_count        NATURALN DEFAULT 1,  
    low              NUMBER   DEFAULT NULL,  
    high             NUMBER   DEFAULT NULL)  
  RETURN DBMS_TF.describe_t  
  AS  
    cols DBMS_TF.columns_new_t;  
    BEGIN  
      FOR i IN 1 .. col_count LOOP  
        cols(i):= DBMS_TF.column_metadata_t(name=>col_name_prefix||i,  
          TYPE=>DBMS_TF.type_number);  
        END LOOP;  
      RETURN DBMS_TF.describe_t(new_columns => cols);  
    END;  
  
  PROCEDURE fetch_rows(col_count NATURALN DEFAULT 1,  
    low       NUMBER   DEFAULT NULL,  
    high      NUMBER   DEFAULT NULL)  
  AS  
    row_count CONSTANT PLS_INTEGER := DBMS_TF.get_env().row_count;  
    col       DBMS_TF.tab_number_t;  
    BEGIN  
      FOR c IN 1 .. col_count LOOP  
        FOR i IN 1 .. row_count LOOP  
          col(i) := CASE WHEN (low IS NULL OR high IS NULL)  
            THEN dbms_random.VALUE  
            ELSE dbms_random.VALUE(low,  
            high)  
            END;  
        END LOOP;  
        DBMS_TF.put_col(c, col);  
      END LOOP;  
    END;  
  END rand_col_p;
Create the standalone rand_col PTF. Specify exactly one formal argument of type TABLE, specify the return type of the PTF as TABLE, specify a Row Semantics PTF type, and indicate the PTF implementation package to use is rand_col_p.

```
CREATE FUNCTION rand_col(tab TABLE,
    col_count NATURALN DEFAULT 1,
    low       NUMBER   DEFAULT NULL,
    high      NUMBER   DEFAULT NULL)
RETURN TABLE
PIPELINED ROW POLYMORPHIC USING rand_col_p;
```

Invoke the rand_col PTF to display all columns of table SCOTT.DEPT with one produced RAND_1 column.

```
SELECT *
FROM rand_col(scott.dept);
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
<th>RAND_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>0.738666262</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
<td>0.093256312</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
<td>0.992944835</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
<td>0.397948124</td>
</tr>
</tbody>
</table>

Invoke the rand_col PTF to display all columns of table SCOTT.DEPT with two produced RAND_1 and RAND_2 columns.

```
SELECT *
FROM rand_col(scott.dept, col_count => 2);
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
<th>RAND_1</th>
<th>RAND_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>0.976521361</td>
<td>0.209802028</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
<td>0.899577891</td>
<td>0.10050334</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
<td>0.277238362</td>
<td>0.110736583</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
<td>0.989839995</td>
<td>0.164822363</td>
</tr>
</tbody>
</table>

For all employees for which their job is not being a SALESMAN, display the employee name, job, and produce three RAND columns generating random values between —10 and 10.

```
SELECT ename, job, rand_1, rand_2, rand_3
FROM   rand_col(scott.emp, col_count => 3, low => -10, high => +10)
WHERE  job != 'SALESMAN';
```

<table>
<thead>
<tr>
<th>ENAME</th>
<th>JOB</th>
<th>RAND_1</th>
<th>RAND_2</th>
<th>RAND_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>CLERK</td>
<td>8.91760464</td>
<td>6.67366638</td>
<td>-9.2789076</td>
</tr>
<tr>
<td>JONES</td>
<td>MANAGER</td>
<td>6.78612961</td>
<td>-1.8617958</td>
<td>6.5282227</td>
</tr>
<tr>
<td>BLAKE</td>
<td>MANAGER</td>
<td>7.59545803</td>
<td>5.22269017</td>
<td>-2.7966401</td>
</tr>
<tr>
<td>CLARK</td>
<td>MANAGER</td>
<td>-6.4747304</td>
<td>-7.3650276</td>
<td>3.28388872</td>
</tr>
<tr>
<td>SCOTT</td>
<td>ANALYST</td>
<td>6.80492435</td>
<td>-3.2271045</td>
<td>-9.7099797</td>
</tr>
<tr>
<td>KING</td>
<td>PRESIDENT</td>
<td>-9.3161177</td>
<td>6.27762154</td>
<td>-1.8184785</td>
</tr>
<tr>
<td>ADAMS</td>
<td>CLERK</td>
<td>-1.6618848</td>
<td>3.13119089</td>
<td>8.06363075</td>
</tr>
<tr>
<td>JAMES</td>
<td>CLERK</td>
<td>2.86918245</td>
<td>-3.5187936</td>
<td>-7.2913809</td>
</tr>
</tbody>
</table>
173.9.10 PUT_ROW_SET Procedure

Writes a collection of new column values in the database.

You can use this procedure to write all new columns in a collection of rows in the database.

This procedure is overloaded. Rows are not replicated by default. You can use the ROW_REPLICATION procedure to set the replication factor.

Syntax

This syntax is used when rows are not replicated.

PROCEDURE PUT_ROW_SET(
   rowset IN ROW_SET_T);

This syntax is used when the replication factor is a constant.

PROCEDURE PUT_ROW_SET(
   rowset IN ROW_SET_T,
   replication_factor IN NATURAL);

This syntax is used when the replication factor is specified as an array with multiple values.

PROCEDURE PUT_ROW_SET(
   rowset IN ROW_SET_T,
   replication_factor IN TAB_NATURAL_T);

Parameters

Table 173-22  PUT_ROW_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowset</td>
<td>The collection of data and metadata</td>
</tr>
<tr>
<td>replication_factor</td>
<td>The replication factor per row</td>
</tr>
</tbody>
</table>

Example 173-14  DBMS_TF.PUT_ROW_SET Example

This code excerpt fetches a collection of rows and writes all new columns back to the database without any processing.

PROCEDURE fetch_rows
AS
   rowset DBMS_TF.ROW_SET_T;
BEGIN
   DBMS_TF.GET_ROW_SET(rowset);
   DBMS_TF.PUT_ROW_SET(rowset);
END;
173.9.10.1 Split Polymorphic Table Function Example

The split PTF example splits each row of the input table into specified pieces.

Example 173-15  Split Polymorphic Table Function Example

This PTF example splits each row of the input table into cnt pieces dividing the values of the split columns.

Create the PTF implementation package split_p.

The parameters are :
- tab - Input table
- col - The names of numeric (input) table columns to split
- cnt - The number of times each input row is to be split

CREATE PACKAGE split_p AS

FUNCTION describe(tab IN OUT DBMS_TF.table_t,
                   col       DBMS_TF.columns_t,
                   cnt       NATURALN)
RETURN DBMS_TF.describe_t;

PROCEDURE fetch_rows(cnt NATURALN);

END split_p;

Create the PTF implementation package body split_p. Each row of the input table is split into cnt pieces dividing the values of the split columns.

CREATE PACKAGE BODY split_p AS

FUNCTION describe(tab IN OUT DBMS_TF.Table_t,
                   col       DBMS_TF.Columns_t,
                   cnt       NATURALN)
RETURN DBMS_TF.describe_t
AS
new_cols DBMS_TF.columns_new_t;
    col_id       PLS_INTEGER := 1;
BEGIN
FOR i IN 1 .. tab.column.count LOOP
    FOR j IN 1 .. col.count LOOP
        IF (tab.column(i).description.name = col(j) AND
            tab.column(i).description.TYPE = DBMS_TF.type_number) THEN
            tab.column(i).pass_through := FALSE;
        END IF;
    END LOOP;
END LOOP;

...
tab.column(i).for_read := TRUE;
new_cols(col_id) := tab.column(i).description;
col_id := col_id + 1;
END IF;
END LOOP;
END LOOP;

RETURN DBMS_TF.describe_t(new_columns=>new_cols,
row_replication=>true);
END;

PROCEDURE fetch_rows(cnt NATURALN)
AS
inp_rs DBMS_TF.row_set_t;
out_rs DBMS_TF.row_set_t;
rows PLS_INTEGER;
BEGIN
DBMS_TF.get_row_set(inp_rs, rows);

FOR c IN 1 .. inp_rs.count() LOOP
  FOR r IN 1 .. rows LOOP
    FOR i IN 1 .. cnt LOOP
      out_rs(c).tab_number((r-1)*cnt+i) := inp_rs(c).tab_number(r)/
cnt;
    END LOOP;
  END LOOP;
END LOOP;

DBMS_TF.put_row_set(out_rs, replication_factor => cnt);
END;

END split_p;

Create the standalone PTF named split. Specify exactly one formal argument of type TABLE, specify the return type of the PTF as TABLE, specify a Row Semantics PTF type, and indicate the PTF implementation package to use is split_p.

CREATE FUNCTION split(tab TABLE, col columns, cnt NATURALN)
RETURN TABLE
PIPELINED ROW POLYMORPHIC USING split_p;

For all employees in department 30, display the ENAME, SAL, and COMM columns. Invoke the split PTF with the COLUMNS pseudo-operator to divide the value of SAL and COMM by 2 for each replicated row returned by the query. Each row is replicated twice.

SELECT ename, sal, comm
FROM split(scott.emp, COLUMNS(sal, comm), cnt => 2)
WHERE deptno=30;

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>COMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLEN</td>
<td>800</td>
<td>150</td>
</tr>
<tr>
<td>ALLEN</td>
<td>800</td>
<td>150</td>
</tr>
<tr>
<td>WARD</td>
<td>625</td>
<td>250</td>
</tr>
<tr>
<td>WARD</td>
<td>625</td>
<td>250</td>
</tr>
</tbody>
</table>
173.9.11 ROW_REPLICATION Procedure

Sets the row replication factor either as a fixed value or as a value per row.

This procedure is overloaded. A Row Semantics polymorphic table function will either produce a single output row for a given input row (one-to-one), or it can produce more output rows for a given input rows (one-to-many), or it can produce no output rows (one-to-none).

Syntax

Sets the row replication factor as a fixed value.

PROCEDURE ROW_REPLICATION(
   replication_factor IN NATURALN);

Sets the row replication factor as a value per row.

PROCEDURE ROW_REPLICATION(
   replication_factor IN TAB_NATURALN_T);

Parameters

Table 173-23  ROW_REPLICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replication_factor</td>
<td>The replication factor per row</td>
</tr>
</tbody>
</table>

Example 173-16  Replicate Polymorphic Table Function Example

This example creates a PTF that replicates each input row by the replication_factor that is given as a parameter.

Live SQL:

You can view and run this example on Oracle Live SQL at Replicate Polymorphic Table Function

Create the PTF implementation package replicate_p.

CREATE PACKAGE replicate_p
AS

   FUNCTION Describe(tab IN OUT DBMS_TF.TABLE_T,
                     replication_factor NATURAL)
      RETURN DBMS_TF.describe_t;
PROCEDURE Fetch_Rows(replication_factor NATURALN);

END replicate_p;

Create the PTF implementation package body replicate_p. The PTF replicates each input row by the replication_factor that is given as a parameter.

CREATE PACKAGE body replicate_p AS

FUNCTION Describe(tab IN OUT DBMS_TF.Table_t, replication_factor NATURAL)
  RETURN DBMS_TF.describe_t AS
  BEGIN
    RETURN DBMS_TF.describe_t(row_replication => True);
  END;

PROCEDURE Fetch_Rows(replication_factor NATURALN)
  AS
  BEGIN
    DBMS_TF.ROW_REPLICATION(replication_factor);
  END;

END replicate_p;

Create a standalone PTF named replicate. Specify exactly one formal argument of type TABLE, specify the return type of the PTF as TABLE, specify a Row Semantics PTF type, and indicate the PTF implementation package to use is replicate_p.

CREATE FUNCTION replicate(tab TABLE, replication_factor NATURAL)
  RETURN TABLE PIPELINED ROW POLYMORPHIC USING replicate_p;

This example sets the replication_factor to 2 which results in doubling the number of rows.

SELECT *
FROM replicate(dept, replication_factor => 2);

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
</tr>
</tbody>
</table>
This example sets the replication_factor to zero.

```
SELECT *
FROM replicate(dept, replication_factor => 0);
```

no rows selected

Count the number of employees in each department. Invoke the replicate PTF to report from the SCOTT.EMP table to set the replication_factor to 1000000.

```
SELECT deptno, COUNT(*)
FROM replicate(scott.emp, le6)
GROUP BY deptno;
```

```
<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>6000000</td>
</tr>
<tr>
<td>10</td>
<td>3000000</td>
</tr>
<tr>
<td>20</td>
<td>5000000</td>
</tr>
</tbody>
</table>
```

This sets the replication_factor to 1000000000.

```
SELECT COUNT(*)
FROM replicate(dual, le9);
```

```
COUNT(*)
----------
1000000000
```

### 173.9.12 ROW_TO_CHAR Function

The ROW_TO_CHAR function converts a row data value to a string representation.

**Syntax**

```
FUNCTION ROW_TO_CHAR(
    rowset ROW_SET_T,
    rid    PLS_INTEGER,
    format PLS_INTEGER DEFAULT FORMAT_JSON)
RETURN VARCHAR2;
```

**Parameters**

**Table 173-24 DBMS_TF.ROW_TO_CHAR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowset</td>
<td>The rowset whose value is to be converted</td>
</tr>
<tr>
<td>rid</td>
<td>Row number</td>
</tr>
<tr>
<td>format</td>
<td>The string format (default is FORMAT_JSON)</td>
</tr>
</tbody>
</table>

**Usage Notes**

Only the JSON format is supported.
Return Values

The string representation in JSON format.

Example 173-17  DBMS_TF.ROW_TO_CHAR Example

PROCEDURE Fetch_Rows as
    rowset DBMS_TF.ROW_SET_T;
    str    VARCHAR2(32000);
BEGIN
    DBMS_TF.GET_ROW_SET(rowset);
    str := DBMS_TF.ROW_TO_CHAR(rowset, 1)
END;

173.9.13 SUPPORTED_TYPE Function

This function tests if a specified type is supported with polymorphic table functions.

Syntax

FUNCTION SUPPORTED_TYPE(
    type_id PLS_INTEGER)
RETURN BOOLEAN;

Parameters

Table 173-25  DBMS_TF.SUPPORTED_TYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_id</td>
<td>The type</td>
</tr>
</tbody>
</table>

Return Values

Returns TRUE if the type_id is a scalar supported by PUT_COL and GET_COL.

See Also:

Echo Polymorphic Table Function Example for an example of DBMS_TF.SUPPORTED_TYPE use.

173.9.14 TRACE Procedure

Prints data structures to help development and problem diagnosis.

This procedure is overloaded.

Syntax

PROCEDURE TRACE(
    msg VARCHAR2,
    with_id BOOLEAN DEFAULT FALSE,
    separator VARCHAR2 DEFAULT NULL,
    prefix VARCHAR2 DEFAULT NULL);
PROCEDURE TRACE(
    rowset IN ROW_SET_T);

PROCEDURE TRACE(
    env IN ENV_T);

PROCEDURE TRACE(
    columns_new IN COLUMNS_NEW_T);

PROCEDURE trace(
    cols IN COLUMNS_T);

PROCEDURE trace(
    columns_with_type IN COLUMNS_WITH_TYPE_T);

PROCEDURE trace(
    tab IN TABLE_T);

PROCEDURE trace(
    col IN COLUMN_METADATA_T);

Parameters

Table 173-26  TRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>Custom user tracing message</td>
</tr>
<tr>
<td>with_id</td>
<td>Include the unique execution ID in the trace?</td>
</tr>
<tr>
<td>separator</td>
<td>Specify a string to use to separate values</td>
</tr>
<tr>
<td>prefix</td>
<td>Specify a string to prefix the actual values</td>
</tr>
<tr>
<td>rowset</td>
<td>Data for a rowset</td>
</tr>
<tr>
<td>env</td>
<td>Metadata about the polymorphic table function execution state</td>
</tr>
<tr>
<td>columns_new</td>
<td>Collection for new columns</td>
</tr>
<tr>
<td>cols</td>
<td>Collection containing column names</td>
</tr>
<tr>
<td>columns_with_type</td>
<td>Collection containing columns metadata</td>
</tr>
<tr>
<td>tab</td>
<td>Table descriptor</td>
</tr>
<tr>
<td>col</td>
<td>Metadata about an existing table column or a new column produced</td>
</tr>
</tbody>
</table>

Example 173-18  DBMS_TF.TRACE Example

This example adds tracing to a fetch_rows procedure.

PROCEDURE fetch_rows
AS
    rowset DBMS_TF.ROW_SET_T;
BEGIN
    DBMS_TF.TRACE('IDENTITY_PACKAGE.Fetch_Rows()', with_id => TRUE);
    DBMS_TF.TRACE(rowset);
    DBMS_TF.GET_ROW_SET(rowset);
    DBMS_TF.TRACE(rowset);
    DBMS_TF.PUT_ROW_SET(rowset);
173.9.15 XSTORE_CLEAR Procedure

Removes all key-value pairs from the XSTORE execution state.

Syntax

PROCEDURE XSTORE_CLEAR;

173.9.16 XSTORE_EXISTS Function

Returns TRUE if an item with a given key exists in the XSTORE.

Syntax

FUNCTION XSTORE_EXISTS(
    key      IN VARCHAR2,
    key_type IN PLS_INTEGER DEFAULT NULL)
RETURN BOOLEAN;

Parameters

Table 173-27  DBMS_TF.XSTORE_EXISTS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>A unique character key</td>
</tr>
<tr>
<td>key_type</td>
<td>The type of key (optional). Default : NULL</td>
</tr>
</tbody>
</table>

Return Values

Returns TRUE if the key has an associated value. When the key_type is NULL (default), it returns TRUE if the key has an associated value of any of the supported type.

When a key_type parameter value is passed, it returns TRUE if the key and specified type of key has an associated value. Otherwise, it returns FALSE.

See Also:

Table 173-6 for more information about supported key types.

173.9.17 XSTORE_GET Procedure

You can use the XSTORE_GET procedure to get the associated value for a given key stored for PTF Execution State Management.

XStore is the PTF execution state management interface. The XStore interface is used to set and store key-value pairs during PTF execution.
This procedure is overloaded. The XStore supports specifying key-value pairs for these scalar types: VARCHAR2, NUMBER, DATE, BOOLEAN.

See Table 173-6 for more information about supported key types.

Syntax

PROCEDURE XSTORE_GET(
    key   IN VARCHAR2,
    value IN OUT VARCHAR2);

PROCEDURE XSTORE_GET(
    key   IN VARCHAR2,
    value IN OUT NUMBER);

PROCEDURE XSTORE_GET(
    key   IN VARCHAR2,
    value IN OUT DATE);

PROCEDURE XSTORE_GET(
    key   IN VARCHAR2,
    value IN OUT BOOLEAN);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>A unique character key</td>
</tr>
<tr>
<td>value</td>
<td>Value corresponding to the key for supported types</td>
</tr>
</tbody>
</table>

Usage Notes

If the key is not found, the value is unchanged.

173.9.17.1 Row_num Polymorphic Table Function Example

The row_num PTF example appends a sequence column to a table.

Example 173-19 Row_num Polymorphic Table Function Example

Create the PTF implementation package row_num_p.

The parameters are:

- tab - The input table
- ini - The initial value (Default = 1)
• inc - The amount to increment (Default = 1)

CREATE PACKAGE row_num_p IS
  FUNCTION describe(tab IN OUT dbms_tf.table_t,
         ini NUMBER DEFAULT 1,
         inc NUMBER DEFAULT 1)
  RETURN dbms_tf.describe_t;

  PROCEDURE fetch_rows(ini NUMBER DEFAULT 1, inc NUMBER DEFAULT 1);
END;

This PTF accepts any input table and appends the sequence column ROW_ID to the table. The sequence values start with the specified value (ini) and each time it is incremented by the specified value (inc).

CREATE PACKAGE BODY row_num_p IS
  FUNCTION describe(tab IN OUT dbms_tf.table_t,
         ini NUMBER DEFAULT 1,
         inc NUMBER DEFAULT 1)
  RETURN dbms_tf.describe_t AS
  BEGIN
    RETURN dbms_tf.describe_t(new_columns =>
      dbms_tf.columns_new_t(1 =>
        dbms_tf.column_metadata_t(name => 'ROW_ID',
                                   TYPE =>
                                   dbms_tf.type_number)));
  END;

  PROCEDURE fetch_rows(ini NUMBER DEFAULT 1, inc NUMBER DEFAULT 1) IS
    row_cnt CONSTANT PLS_INTEGER := dbms_tf.get_env().row_count;
    rid NUMBER := ini;
    col dbms_tf.tab_number_t;
  BEGIN
    dbms_tf.xstore_get('rid', rid);
    FOR i IN 1 .. row_cnt LOOP col(i) := rid + inc*(i-1); END LOOP;
    dbms_tf.put_col(1, col);
    dbms_tf.xstore_set('rid', rid + inc*row_cnt);
  END;
END;

Create a standalone polymorphic table function named row_num. Specify exactly one formal argument of type TABLE, specify the return type of the PTF as TABLE, specify a Table Semantics PTF type, and indicate the PTF implementation package to use is row_num_p.

CREATE FUNCTION row_num(tab TABLE,
         ini NUMBER DEFAULT 1,
         inc NUMBER DEFAULT 1)
  RETURN TABLE
PIPELINED TABLE POLYMORPHIC USING row_num_p;
The `row_num` PTF invocation reporting from the `SCOTT.DEPT` table produces a new column `ROW_ID` with value starting at 1 and incremented by 1 in the row set.

```sql
SELECT * FROM row_num(scott.dept);
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
<th>ROW_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
<td>4</td>
</tr>
</tbody>
</table>

The `row_num` PTF invocation reporting from the `SCOTT.DEPT` table produces a new column `ROW_ID` with value starting at 100 and incremented by 1 in the row set.

```sql
SELECT * FROM row_num(scott.dept, 100);
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
<th>ROW_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
<td>101</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
<td>102</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
<td>103</td>
</tr>
</tbody>
</table>

The `row_num` PTF invocation reporting from the `SCOTT.DEPT` table produces a new column `ROW_ID` with value starting at 0 and decremented by 1 in the row set.

```sql
SELECT * FROM row_num(scott.dept, ini => 0, inc => -1);
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
<th>ROW_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
<td>-1</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
<td>-2</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
<td>BOSTON</td>
<td>-3</td>
</tr>
</tbody>
</table>

The `row_num` PTF invocation reporting from the `SCOTT.EMP` table produces a new column `ROW_ID` with value starting at 0 and incremented by 0.25 in the row set which is partitioned by department number and ordered by employee name.

```sql
SELECT deptno, ename, job, sal, row_id
  FROM   row_num(scott.emp PARTITION BY deptno ORDER BY ename, ini => 0, inc => 0.25)
  WHERE  deptno IN (10, 30);
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>SAL</th>
<th>ROW_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>CLARK</td>
<td>MANAGER</td>
<td>2450</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>KING</td>
<td>PRESIDENT</td>
<td>5000</td>
<td>.25</td>
</tr>
<tr>
<td>10</td>
<td>MILLER</td>
<td>CLERK</td>
<td>1300</td>
<td>.5</td>
</tr>
<tr>
<td>30</td>
<td>ALLEN</td>
<td>SALESMAN</td>
<td>1600</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>BLAKE</td>
<td>MANAGER</td>
<td>2850</td>
<td>.25</td>
</tr>
<tr>
<td>30</td>
<td>JAMES</td>
<td>CLERK</td>
<td>950</td>
<td>.5</td>
</tr>
<tr>
<td>30</td>
<td>MARTIN</td>
<td>SALESMAN</td>
<td>1250</td>
<td>.75</td>
</tr>
</tbody>
</table>
173.9.18 XSTORE_REMOVE Procedure

Removes an item associated with the given key and key_type.

Syntax

PROCEDURE XSTORE_REMOVE (  
    key      IN VARCHAR2,  
    key_type IN PLS_INTEGER DEFAULT NULL);  

Parameters

Table 173-29  DBMS_TF.XSTORE_REMOVE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>A unique character key</td>
</tr>
<tr>
<td>key_type</td>
<td>The type of key to remove (optional)</td>
</tr>
</tbody>
</table>

Usage Notes

When a key_type parameter value is passed, it removes the associated item for the key and specified type of key.

173.9.19 XSTORE_SET Procedure

Sets the value for the given key for PTF Execution State Management.

You can use this procedure to store and item key-value pair in the XStore. This procedure is overloaded. The XStore supports specifying key-value pairs for these scalar types: VARCHAR2, NUMBER, DATE, BOOLEAN.

Syntax

PROCEDURE XSTORE_SET (  
    key   IN VARCHAR2,  
    value IN VARCHAR2);  

PROCEDURE XSTORE_SET (  
    key   IN VARCHAR2,  
    value IN NUMBER);  

PROCEDURE XSTORE_SET (  
    key   IN VARCHAR2,  
    value IN DATE);  

PROCEDURE XSTORE_SET (  
    key   IN VARCHAR2,  
    value IN BOOLEAN);
Parameters

Table 173-30  DBMS_TF.XSTORE_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>A unique character key</td>
</tr>
<tr>
<td>value</td>
<td>Value corresponding to the key for supported types</td>
</tr>
</tbody>
</table>

Usage Notes

If an item for a given key already exists, the value is replaced.
The DBMS_TNS package provides the RESOLVE_TNSNAME function to resolve a TNS name and return the corresponding Oracle Net8 connection string.

This chapter contains the following topics:

- DBMS_TNS Overview
- DBMS_TNS Security Model
- Summary of DBMS_TNS Subprograms

174.1 DBMS_TNS Overview

The DBMS_TNS package contains one function, RESOLVE_TNSNAME, which returns the resolved connect string from configured sources that have been defined in the names.directory_path parameter in the sqlnet.ora file.

The RESOLVE_TNSNAME function in this package helps you to identify interconnected databases and trace the system change number (SCN) flow across distributed databases.

See Also:

- Oracle Database Administrator’s Guide for more information about finding database link information
- Oracle Database Reference for information about the ALL_DB_LINKS data dictionary view

174.2 DBMS_TNS Security Model

You must be granted the EXECUTE privilege on the DBMS_TNS package.

174.3 Summary of DBMS_TNS Subprograms

This table lists the DBMS_TNS subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOLVE_TNSNAME Function</td>
<td>Returns the resolved connect string from any configured source (tnsnames, LDAP, Easy Connect) as indicated in the sqlnet.ora names.directory_path parameter</td>
</tr>
</tbody>
</table>
174.3.1 RESOLVE_TNSNAME Function

This function returns the resolved connect string from any configured source (for example, TNS names, Lightweight Directory Access Protocol (LDAP), or Oracle Easy Connect) as indicated in the sqlnet.ora names.directory_path parameter.

**Syntax**

```sql
DBMS_TNS.RESOLVE_TNSNAME (tns_name VARCHAR2)
RETURN VARCHAR2;
```

**RESOLVE_TNSNAME Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tns_name</td>
<td>Database address for establishing the connection</td>
</tr>
</tbody>
</table>

**Usage Notes**

- Database administrators and system management tools can explicitly call the DBMS_TNS.RESOLVE_TNSNAME function to resolve a tns_name.
- Because the resolved value can change over time, Oracle does not recommend storing or caching this value, because these kinds of values can become invalid over time.
- You can use the DBMS_TNS.RESOLVE_TNSNAME function to resolve TNS names that were referred to by other data dictionary views such as ALL_DB_LINKS and USER_DB_LINKS. In addition, you can use it to find any TNS name entry in the tnsnames.ora file.

**Example**

```sql
SELECT DB_LINK, DBMS_TNS.RESOLVE_TNSNAME(HOST) FROM DBA_DB_LINKS;
```

```
DB_LINK
------------------------------------------
---------
DBMS_TNS.RESOLVE_TNSNAME(HOST)
------------------------------------------
---------
US.EXAMPLE.COM
{DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=may))(CONNECT_DATA=(SERVICE_NAME=us.example.com)(CID=(PROGRAM=oracle)(HOST=juno)(USER=psmith))}
```
175

DBMS_TRACE

The DBMS_TRACE package contains the interface to trace PL/SQL functions, procedures, and exceptions.

This chapter contains the following topics:

• Overview
• Security Model
• Constants
• Restrictions
• Operational Notes
• Summary of DBMS_TRACE Subprograms

175.1 DBMS_TRACE Overview

DBMS_TRACE provides subprograms to start and stop PL/SQL tracing in a session. Oracle collects the trace data as the program executes and writes it to database tables.

A typical session involves:

1. (Optional) Limit tracing to specific subprograms and choose a tracing level.
   Tracing all subprograms and exceptions in a large program can produce huge amounts of data that are difficult to manage.

2. Starting PL/SQL tracing in session (DBMS_TRACE.SET_PLSQL_TRACE).

3. Running an application to be traced.


After you have collected data with Trace, you can query the database tables that contain the performance data and analyze it in the same way that you analyze the performance data from Profiler.

175.2 DBMS_TRACE Security Model

This package must be created under SYS.

175.3 DBMS_TRACE Constants

DBMS_TRACE defines constants to use when specifying parameter values.

These constants are shown in the following table.
Table 175-1  DBMS_TRACE Event Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE_ALL_CALLS</td>
<td>INTEGER</td>
<td>1</td>
<td>Traces calls or returns</td>
</tr>
<tr>
<td>TRACE_ENABLED_CALLS</td>
<td>INTEGER</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>TRACE_ALL_EXCEPTIONS</td>
<td>INTEGER</td>
<td>4</td>
<td>Traces exceptions</td>
</tr>
<tr>
<td>TRACE_ENABLED_EXCEPTIONS</td>
<td>INTEGER</td>
<td>8</td>
<td>Traces exceptions and handlers</td>
</tr>
<tr>
<td>TRACE_LIMIT</td>
<td>INTEGER</td>
<td>16</td>
<td>Save only the last few records. This allows tracing up to a problem area, without filling the database up with masses of irrelevant information. If event 10940 is set, the limit is 1023*(the value of event 10940). This can be overridden by the use of &quot;TRACE_LIMIT&quot; flag.</td>
</tr>
<tr>
<td>TRACE_ALL_SQL</td>
<td>INTEGER</td>
<td>32</td>
<td>Traces SQL statements</td>
</tr>
<tr>
<td>TRACE_ENABLED_SQL</td>
<td>INTEGER</td>
<td>64</td>
<td>Traces SQL statements at PL/SQL level. This does not invoke SQL Trace</td>
</tr>
<tr>
<td>TRACE_ALL_LINES</td>
<td>INTEGER</td>
<td>128</td>
<td>Traces each line</td>
</tr>
<tr>
<td>TRACE_ENABLED_LINES</td>
<td>INTEGER</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>TRACE_PAUSE</td>
<td>INTEGER</td>
<td>4096</td>
<td>Pauses tracing</td>
</tr>
<tr>
<td>TRACE_RESUME</td>
<td>INTEGER</td>
<td>8192</td>
<td>Resume tracing</td>
</tr>
<tr>
<td>TRACE_STOP</td>
<td>INTEGER</td>
<td>16384</td>
<td>Stops tracing</td>
</tr>
<tr>
<td>NO_TRACE_ADMINISTRATIVE</td>
<td>INTEGER</td>
<td>32768</td>
<td>Prevents tracing of administrative events such as • PL/SQL Trace Tool started • Trace flags changed • PL/SQL Virtual Machine started • PL/SQL Virtual Machine stopped</td>
</tr>
<tr>
<td>NO_TRACEHandled Exceptions</td>
<td>INTEGER</td>
<td>65536</td>
<td>Prevents tracing of handled exceptions</td>
</tr>
</tbody>
</table>

Table 175-2  DBMS_TRACE Version Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE_MINOR_VERSION</td>
<td>INTEGER</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>TRACE_MAJOR_VERSION</td>
<td>INTEGER</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Oracle recommends using the symbolic form for all these constants.
175.4 DBMS_TRACE Restrictions

You cannot use PL/SQL tracing in a shared server environment.

175.5 DBMS_TRACE Operational Notes

Certain operational notes apply to DBMS_TRACE.

These are described in the following sections:

• Controlling Data Volume
• Creating Database Tables to Collect DBMS_TRACE Output
• Collecting Trace Data
• Collected Data
• Trace Control

Controlling Data Volume

Profiling large applications may produce a large volume of data. You can control the volume of data collected by enabling specific program units for trace data collection.

You can enable a program unit by compiling it debug. This can be done in one of two ways:

```sql
alter session set plsql_debug=true;
create or replace ... /* create the library units - debug information will be generated */

or:

/* recompile specific library unit with debug option */
alter [PROCEDURE | FUNCTION | PACKAGE BODY] <libunit-name> compile debug;
```

**Note:**

You cannot use the second method for anonymous blocks.

You can limit the amount of storage used in the database by retaining only the most recent 8,192 records (approximately) by including `TRACE_LIMIT` in the `TRACE_LEVEL` parameter of the `SET_PLSQL_TRACE` procedure.

Creating Database Tables to Collect DBMS_TRACE Output

You must create database tables into which the DBMS_TRACE package writes output. Otherwise, the data is not collected. To create these tables, run the script `TRACETAB.SQL`. The tables this script creates are owned by SYS.

Collecting Trace Data

The PL/SQL features you can trace are described in the script `DBMSPBTL.SQL`. Some of the key tracing features are:
• Tracing Calls
  • Tracing Exceptions
  • Tracing SQL
  • Tracing Lines

Additional features of DBMS_TRACE also allow pausing and resuming trace, and limiting the output.

**Tracing Calls**

Two levels of call tracing are available:

- **Level 1**: Trace all calls. This corresponds to the constant `TRACE_ALL_CALLS`.
- **Level 2**: Trace calls to enabled program units only. This corresponds to the constant `TRACE_ENABLED_CALLS`.

Enabling cannot be detected for remote procedure calls (RPCs); hence, RPCs are only traced with level 1.

**Tracing Exceptions**

Two levels of exception tracing are available:

- **Level 1**: Trace all exceptions. This corresponds to `TRACE_ALL_EXCEPTIONS`.
- **Level 2**: Trace exceptions raised in enabled program units only. This corresponds to `TRACE_ENABLED_EXCEPTIONS`.

**Tracing SQL**

Two levels of SQL tracing are available:

- **Level 1**: Trace all SQL. This corresponds to `TRACE_ALL_SQL`.
- **Level 2**: Trace SQL in enabled program units only. This corresponds to the constant `TRACE_ENABLED_SQL`.

**Tracing Lines**

Two levels of line tracing are available:

- **Level 1**: Trace all lines. This corresponds to the constant `TRACE_ALL_LINES`.
- **Level 2**: Trace lines in enabled program units only. This corresponds to the constant `TRACE_ENABLED_LINES`.

When tracing lines, Oracle adds a record to the database each time the line number changes. This includes line number changes due to procedure calls and returns.

---

**Note:**

For all types of tracing, level 1 overrides level 2. For example, if both level 1 and level 2 are enabled, then level 1 takes precedence.
Collected Data

If tracing is requested only for enabled program units, and if the current program unit is not enabled, then no trace data is written.

When tracing calls, both the call and return are traced. The check for whether tracing is "enabled" passes if either the called routine or the calling routine is "enabled".

Call tracing will always output the program unit type, program unit name, and line number for both the caller and the callee. It will output the caller's stack depth. If the caller is enabled, the caller's name will also be output. If the callee is enabled, the callee's name will also be output.

Exception tracing writes out the line number. Raising the exception shows information on whether the exception is user-defined or pre-defined. It also shows the exception number in the case of pre-defined exceptions. Both the place where the exceptions are raised and their handler is traced. The check for tracing being "enabled" is done independently for the place where the exception is raised and the place where the exception is handled. Enabling NO_TRACE_HANDLED_EXCEPTIONS limits data collection to unhandled exceptions.

All calls to DBMS_TRACE_SET_PLSQL_TRACE and DBMS_TRACE_CLEAR_PLSQL_TRACE place a special trace record in the database. Therefore, it is always possible to determine when trace settings were changed.

Trace Control

As well as determining which items are collected, you can pause and resume the trace process. No information is gathered between the time that tracing is paused and the time that it is resumed. The constants TRACE_PAUSE and TRACE_RESUME are used to accomplish this. Trace records are generated to indicate that the trace was paused/resumed.

It is also possible to retain only the last 8,192 trace events of a run by using the constant TRACE_LIMIT. This allows tracing to be turned on without filling up the database. When tracing stops, the last 8,192 records are saved. The limit is approximate, since it is not checked on every trace record. At least the requested number of trace records will be generated; up to 1,000 additional records may be generated. At least the requested number of trace records will be generated; up to 1,000 additional records may be generated. The 8,192 record limit can be changed. Setting event 10940 to level \( n \) changes the record limit to \( 1024 \times n \).

Enabling NO_TRACE_ADMINISTRATIVE prevents the generation of such administrative event records as PL/SQL Trace Tool started, Trace flags changed, PL/SQL Virtual Machine started, and PL/SQL Virtual Machine stopped.

175.6 Summary of DBMS_TRACE Subprograms

This table lists the DBMS_TRACE subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR_PLSQL_TRACE Procedure</td>
<td>Stops trace data dumping in session</td>
</tr>
<tr>
<td>GET_PLSQL_TRACE_LEVEL Function</td>
<td>Gets the trace level</td>
</tr>
</tbody>
</table>
### Table 175-3  (Cont.) DBMS_TRACE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLSQL_TRACE_VERSION Procedure</td>
<td>Gets the version number of the trace package</td>
</tr>
<tr>
<td>SET_PLSQL_TRACE Procedure</td>
<td>Starts tracing in the current session</td>
</tr>
</tbody>
</table>

#### 175.6.1 CLEAR_PLSQL_TRACE Procedure

This procedure disables trace data collection.

**Syntax**

```
DBMS_TRACE.CLEAR_PLSQL_TRACE;
```

#### 175.6.2 GET_PLSQL_TRACE_LEVEL Function

This procedure returns the current trace level as the sum of one or more `DBMS_TRACE` constants.

See Table 175-1 for a list of the constants.

**Syntax**

```
DBMS_TRACE.GET_PLSQL_TRACE_LEVEL
    RETURN BINARY_INTEGER;
```

#### 175.6.3 PLSQL_TRACE_VERSION Procedure

This procedure gets the version number of the trace package. It returns the major and minor version number of the `DBMS_TRACE` package.

**Syntax**

```
DBMS_TRACE.PLSQL_TRACE_VERSION (  
    major OUT BINARY_INTEGER,   
    minor OUT BINARY_INTEGER);  
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>Major version number of <code>DBMS_TRACE</code>.</td>
</tr>
<tr>
<td>minor</td>
<td>Minor version number of <code>DBMS_TRACE</code>.</td>
</tr>
</tbody>
</table>
175.6.4 SET_PLSQL_TRACE Procedure

This procedure enables PL/SQL trace data collection.

Syntax

DBMS_TRACE.SET_PLSQL_TRACE (trace_level INTEGER);

Parameters

Table 175-5  SET_PLSQL_TRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace_level</td>
<td>You must supply one or more of the constants as listed in Table 175-1. By summing the constants, you can enable tracing of multiple PL/SQL language features simultaneously. The control constants “TRACE_PAUSE”, “TRACE_RESUME” and “TRACE_STOP” should not be used in combination with other constants. Also see DBMS_TRACE Operational Notes: Collecting Trace Data for more information.</td>
</tr>
</tbody>
</table>
The DBMS_TRANSACTION package provides access to SQL transaction statements from stored procedures.

See Also:

Oracle Database SQL Language Reference

This chapter contains the following topics:

- DBMS_TRANSACTION Security Model
- Summary of DBMS_TRANSACTION Subprograms

176.1 DBMS_TRANSACTION Security Model

This package runs with the privileges of calling user, rather than the package owner SYS.

176.2 Summary of DBMS_TRANSACTION Subprograms

This table lists the DBMS_TRANSACTION subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVISE_COMMIT Procedure</td>
<td>Equivalent to the SQL statement:</td>
</tr>
<tr>
<td></td>
<td>ALTER SESSION ADVISE COMMIT</td>
</tr>
<tr>
<td>ADVISE NOTHING Procedure</td>
<td>Equivalent to the SQL statement:</td>
</tr>
<tr>
<td></td>
<td>ALTER SESSION ADVISE NOTHING</td>
</tr>
<tr>
<td>ADVISE_ROLLBACK Procedure</td>
<td>Equivalent to the SQL statement:</td>
</tr>
<tr>
<td></td>
<td>ALTER SESSION ADVISE ROLLBACK</td>
</tr>
<tr>
<td>COMMIT Procedure</td>
<td>Equivalent to the SQL statement:</td>
</tr>
<tr>
<td></td>
<td>COMMIT</td>
</tr>
<tr>
<td>COMMITCOMMENT Procedure</td>
<td>Equivalent to the SQL statement:</td>
</tr>
<tr>
<td></td>
<td>COMMIT COMMENT &lt;text&gt;</td>
</tr>
<tr>
<td>COMMIT_FORCE Procedure</td>
<td>Equivalent to the SQL statement:</td>
</tr>
<tr>
<td></td>
<td>COMMIT FORCE &lt;text&gt;, &lt;number&gt;</td>
</tr>
<tr>
<td>LOCAL_TRANSACTION_ID Function</td>
<td>Returns the local (to instance) unique identifier for the current transaction</td>
</tr>
</tbody>
</table>

176-1
Table 176-1  (Cont.) DBMS_TRANSACTION Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PURGE_LOST_DB_ENTRY Procedure</strong></td>
<td>Enables removal of incomplete transactions from the local site when the remote database is destroyed or re-created before recovery completes</td>
</tr>
<tr>
<td><strong>PURGE_MIXED Procedure</strong></td>
<td>Deletes information about a given mixed outcome transaction</td>
</tr>
<tr>
<td><strong>READ_ONLY Procedure</strong></td>
<td>Equivalent to the SQL statement: SET TRANSACTION READ ONLY</td>
</tr>
<tr>
<td><strong>READ_WRITE Procedure</strong></td>
<td>equivalent to the SQL statement: SET TRANSACTION READ WRITE</td>
</tr>
<tr>
<td><strong>ROLLBACK Procedure</strong></td>
<td>Equivalent to the SQL statement: ROLLBACK</td>
</tr>
<tr>
<td><strong>ROLLBACK_FORCE Procedure</strong></td>
<td>Equivalent to the SQL statement: ROLLBACK FORCE &lt;text&gt;</td>
</tr>
<tr>
<td><strong>ROLLBACK_SAVEPOINT Procedure</strong></td>
<td>Equivalent to the SQL statement: ROLLBACK TO SAVEPOINT &lt;savepoint_name&gt;</td>
</tr>
<tr>
<td><strong>SAVEPOINT Procedure</strong></td>
<td>Equivalent to the SQL statement: SAVEPOINT &lt;savepoint_name&gt;</td>
</tr>
<tr>
<td><strong>STEP_ID Function</strong></td>
<td>Returns local (to local transaction) unique positive integer that orders the DML operations of a transaction</td>
</tr>
<tr>
<td><strong>USE_ROLLBACK_SEGMENT Procedure</strong></td>
<td>Equivalent to the SQL statement: SET TRANSACTION USE ROLLBACK SEGMENT &lt;rb_seg_name&gt;</td>
</tr>
</tbody>
</table>

176.2.1 ADVISE_COMMIT Procedure

This procedure is equivalent to the SQL statement: ALTER SESSION ADVISE COMMIT

Syntax

```
DBMS_TRANSACTION.ADVISE_COMMIT;
```

176.2.2 ADVISE NOTHING Procedure

This procedure is equivalent to the SQL statement: ALTER SESSION ADVISE NOTHING

Syntax

```
DBMS_TRANSACTION.ADVISE NOTHING;
```
176.2.3 ADVISE_ROLLBACK Procedure

This procedure is equivalent to the SQL statement: ALTER SESSION ADVISE ROLLBACK

Syntax

DBMS_TRANSACTION.ADVISE_ROLLBACK;

176.2.4 COMMIT Procedure

This procedure is equivalent to the SQL statement: COMMIT

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

Syntax

DBMS_TRANSACTION.COMMIT;

176.2.5 COMMIT_COMMENT Procedure

This procedure is equivalent to the SQL statement: COMMIT COMMENT <text>

Syntax

DBMS_TRANSACTION.COMMIT_COMMENT (
  cmnt VARCHAR2);

Parameters

Table 176-2  COMMIT_COMMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmnt</td>
<td>Comment to associate with this commit.</td>
</tr>
</tbody>
</table>

176.2.6 COMMIT_FORCE Procedure

This procedure is equivalent to the SQL statement: COMMIT FORCE <text>, <number>"

Syntax

DBMS_TRANSACTION.COMMIT_FORCE {
  xid VARCHAR2,
  scn VARCHAR2 DEFAULT NULL};

Parameters

Table 176-3  COMMIT_FORCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>Local or global transaction ID.</td>
</tr>
</tbody>
</table>
Table 176-3  (Cont.) COMMIT_FORCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scn</td>
<td>System change number.</td>
</tr>
</tbody>
</table>

176.2.7 LOCAL_TRANSACTION_ID Function

This function returns the local (to instance) unique identifier for the current transaction. It returns null if there is no current transaction.

Syntax

```
DBMS_TRANSACTION.LOCAL_TRANSACTION_ID (
    create_transaction BOOLEAN := FALSE)
RETURN VARCHAR2;
```

Parameters

Table 176-4  LOCAL_TRANSACTION_ID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_transaction</td>
<td>If true, then start a transaction if one is not currently active.</td>
</tr>
</tbody>
</table>

176.2.8 PURGE_LOST_DB_ENTRY Procedure

Procedure `PURGE_LOST_DB_ENTRY` purges entries that control database recovery from a local site.

When a failure occurs during commit processing, automatic recovery consistently resolves the results at all sites involved in the transaction. However, if the remote database is destroyed or re-created before recovery completes, then the entries used to control recovery in `DBA_2PC_PENDING` and associated tables are never removed, and recovery will periodically retry. Procedure `PURGE_LOST_DB_ENTRY` enables removal of such transactions from the local site.

Syntax

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY (
    xid VARCHAR2);
```

Parameters

Table 176-5  PURGE_LOST_DB_ENTRY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>Must be set to the value of the <code>LOCAL_TRAN_ID</code> column in the <code>DBA_2PC_PENDING</code> table.</td>
</tr>
</tbody>
</table>
### Usage Notes

**WARNING:**

PURGE_LOST_DB_ENTRY should *only* be used when the other database is lost or has been re-created. Any other use may leave the other database in an unrecoverable or inconsistent state.

Before automatic recovery runs, the transaction may show up in DBA_2PC_PENDING as state "collecting", "committed", or "prepared". If the DBA has forced an in-doubt transaction to have a particular result by using "commit force" or "rollback force", then states "forced commit" or "forced rollback" may also appear. Automatic recovery normally deletes entries in any of these states. The only exception is when recovery finds a forced transaction which is in a state inconsistent with other sites in the transaction; in this case, the entry is left in the table and the MIXED column has the value 'yes'.

However, under certain conditions, it may not be possible for automatic recovery to run. For example, a remote database may have been permanently lost. Even if it is re-created, it gets a new database ID, so that recovery cannot identify it (a possible symptom is ORA-02062). In this case, the DBA may use the procedure PURGE_LOST_DB_ENTRY to clean up the entries in any state other than "prepared". The DBA does not need to be in any particular hurry to resolve these entries, because they are not holding any database resources.

The following table indicates what the various states indicate about the transaction and what the DBA actions should be:

### Table 176-6  PURGE_LOST_DB_ENTRY Procedure States

<table>
<thead>
<tr>
<th>State of Column</th>
<th>State of Global Transaction</th>
<th>State of Local Transaction</th>
<th>Normal DBA Action</th>
<th>Alternative DBA Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collecting</td>
<td>Rolled back</td>
<td>Rolled back</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY (See Note 1)</td>
</tr>
<tr>
<td>Committed</td>
<td>Committed</td>
<td>Committed</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY (See Note 1)</td>
</tr>
<tr>
<td>Prepared</td>
<td>Unknown</td>
<td>Prepared</td>
<td>None</td>
<td>FORCE COMMIT or ROLLBACK</td>
</tr>
<tr>
<td>Forced commit</td>
<td>Unknown</td>
<td>Committed</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY (See Note 1)</td>
</tr>
<tr>
<td>Forced rollback</td>
<td>Unknown</td>
<td>Rolled back</td>
<td>None</td>
<td>PURGE_LOST_DB_ENTRY (See Note 1)</td>
</tr>
<tr>
<td>Forced commit (mixed)</td>
<td>Mixed</td>
<td>Committed</td>
<td>(See Note 2)</td>
<td></td>
</tr>
<tr>
<td>Forced rollback (mixed)</td>
<td>Mixed</td>
<td>Rolled back</td>
<td>(See Note 2)</td>
<td></td>
</tr>
</tbody>
</table>
176.2.9 PURGE_MIXED Procedure

This procedure deletes information about a given mixed outcome transaction.

When in-doubt transactions are forced to commit or rollback (instead of letting automatic recovery resolve their outcomes), there is a possibility that a transaction can have a mixed outcome; some sites commit, and others rollback. Such inconsistency cannot be resolved automatically by Oracle. However, Oracle flags entries in `DBA_2PC_PENDING` by setting the `MIXED` column to a value of 'yes'.

Oracle never automatically deletes information about a mixed outcome transaction. When the application or DBA is certain that all inconsistencies that might have arisen as a result of the mixed transaction have been resolved, this procedure can be used to delete the information about a given mixed outcome transaction.

Syntax

```
DBMS_TRANSACTION.PURGE_MIXED (
        xid VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>Must be set to the value of the <code>LOCAL_TRAN_ID</code> column in the <code>DBA_2PC_PENDING</code> table.</td>
</tr>
</tbody>
</table>

176.2.10 READ_ONLY Procedure

This procedure is equivalent to the SQL statement `SET TRANSACTION READ ONLY`.

Syntax

```
DBMS_TRANSACTION.READ_ONLY;
```
176.2.11 READ_WRITE Procedure

This procedure is equivalent to the SQL statement:

```
SET TRANSACTION READ WRITE
```

**Syntax**

```
DBMS_TRANSACTION.READ_WRITE;
```

176.2.12 ROLLBACK Procedure

This procedure is equivalent to the SQL statement `ROLLBACK`.

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

**Syntax**

```
DBMS_TRANSACTION.ROLLBACK;
```

176.2.13 ROLLBACK_FORCE Procedure

This procedure is equivalent to the SQL statement `ROLLBACK FORCE <text>`.

**Syntax**

```
DBMS_TRANSACTION.ROLLBACK_FORCE (
    xid VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>Local or global transaction ID.</td>
</tr>
</tbody>
</table>

176.2.14 ROLLBACK_SAVEPOINT Procedure

This procedure is equivalent to the SQL statement `ROLLBACK TO SAVEPOINT <savepoint_name>`.

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

**Syntax**

```
DBMS_TRANSACTION.ROLLBACK_SAVEPOINT (
    savept VARCHAR2);
```
Parameters

Table 176-9  ROLLBACK_SAVEPOINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>savept</td>
<td>Savepoint identifier.</td>
</tr>
</tbody>
</table>

176.2.15 SAVEPOINT Procedure

This procedure is equivalent to the SQL statement `SAVEPOINT <savepoint_name>`. This procedure is included for completeness, the feature being already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.SAVEPOINT (savept VARCHAR2);
```

Parameters

Table 176-10  SAVEPOINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>savept</td>
<td>Savepoint identifier.</td>
</tr>
</tbody>
</table>

176.2.16 STEP_ID Function

This function returns local (to local transaction) unique positive integer that orders the DML operations of a transaction.

Syntax

```
DBMS_TRANSACTION.STEP_ID RETURN NUMBER;
```

176.2.17 USE_ROLLBACK_SEGMENT Procedure

This procedure is equivalent to the SQL statement `SET TRANSACTION USE ROLLBACK SEGMENT <rb_seg_name>`.  

Syntax

```
DBMS_TRANSACTION.USE_ROLLBACK_SEGMENT (rb_name VARCHAR2);
```
Parameters

Table 176-11  USE_ROLLBACK_SEGMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rb_name</td>
<td>Name of rollback segment to use.</td>
</tr>
</tbody>
</table>
177

DBMS_TRANSFORM

The **DBMS_TRANSFORM** package provides an interface to the message format transformation features of Oracle Advanced Queuing.

This chapter contains the following topic:

- Summary of DBMS_TRANSFORM Subprograms

---

**See Also:**

*Oracle Database Advanced Queuing User's Guide* for more on message format transformations.

### 177.1 Summary of DBMS_TRANSFORM Subprograms

This table lists the **DBMS_TRANSFORM** subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_TRANSFORMATION Procedure</td>
<td>Creates a transformation that maps an object of the source type to an object of the destination type</td>
</tr>
<tr>
<td>DROP_TRANSFORMATION Procedure</td>
<td>Drops the given transformation</td>
</tr>
<tr>
<td>MODIFY_TRANSFORMATION Procedure</td>
<td>Modifies an existing transformation</td>
</tr>
</tbody>
</table>

### 177.1.1 CREATE_TRANSFORMATION Procedure

This procedure creates a transformation that maps an object of the source type to an object of the target type. The transformation expression can be a SQL expression or a PL/SQL function. It must return an object of the target type.

**Syntax**

```sql
DBMS_TRANSFORM.CREATE_TRANSFORMATION (  
schema       VARCHAR2 (30),  
name         VARCHAR2 (30),  
from_schema  VARCHAR2 (30),  
from_type    VARCHAR2 (30),  
to_schema    VARCHAR2 (30),  
to_type      VARCHAR2 (30),  
transformation VARCHAR2 (4000));
```
Parameters

Table 177-2  CREATE_TRANSFORMATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Specifies the schema of the transformation.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the transformation.</td>
</tr>
<tr>
<td>from_schema</td>
<td>Specifies the schema of the source type.</td>
</tr>
<tr>
<td>from_type</td>
<td>Specifies the source type.</td>
</tr>
<tr>
<td>to_schema</td>
<td>Specifies the target type schema.</td>
</tr>
<tr>
<td>to_type</td>
<td>Specifies the target type.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies the transformation expression, returning an object of the target type. The expression must be a function returning an object of the target type or a constructor expression for the target type. You can choose not to specify a transformation expression and instead specify transformations for attributes of the target type using MODIFY_TRANSFORMATION.</td>
</tr>
</tbody>
</table>

Usage Notes

• The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type.

• To create, modify or drop transformations, a user must be granted execute privileges on DBMS_TRANSFORM. The user must also have execute privileges on the user defined types that are the source and destination types of the transformation. In addition, the user must also have execute privileges on any PLSQL function being used in the transformation function.

• The transformation cannot write database state (perform DML) or commit or rollback the current transaction.

• The transformation must be a SQL function with source type as input type, returning an object of the target type. It could also be a SQL expression of target type, referring to a source type. All references to the source type must be of the form source.user_data.

• Both source and target types must be non-scalar database types. A null transformation expression maps to a null target object.

For using the transformation at enqueue and dequeue time, the login user invoking the operation must have execute privileges on the PLSQL functions used by the transformation. For propagation, the owning schema of the queue must have these privileges.

177.1.2 DROP_TRANSFORMATION Procedure

This procedure drops the given transformation.

Syntax

DBMS_TRANSFORM.DROP_TRANSFORMATION (  
  schema VARCHAR2(30),  
)
Parameters

Table 177-3  DROP_TRANSFORMATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Specifies the schema of the transformation.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the transformation.</td>
</tr>
</tbody>
</table>

177.1.3 MODIFY_TRANSFORMATION Procedure

This procedure modifies the transformation expression for the given transformation.

Syntax

```sql
DBMS_TRANSFORM.MODIFY_TRANSFORMATION (
    schema             VARCHAR2(30),
    name               VARCHAR2(30),
    attribute_number   INTEGER,
    transformation     VARCHAR2(4000));
```

Parameters

Table 177-4  MODIFY_TRANSFORMATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Specifies the schema of the transformation.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the transformation.</td>
</tr>
<tr>
<td>attribute_number</td>
<td>The attribute of the target type for which the new transformation expression is being specified. When specifying the new transformation as a single expression of the target type, specify a value of 0.</td>
</tr>
<tr>
<td>transformation</td>
<td>The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type. If the attribute_number is 0, then the expression must be a PL/SQL function returning an object of the target type or a constructor expression for the target type.</td>
</tr>
</tbody>
</table>

Usage Notes

- If the new transformation is a single expression of the target type, it may be specified with an attribute_number of 0. The new transformation may also be specified for each attribute of the target type.
- You can use this procedure to define the transformation as a separate expression for each attribute of the target type. For large transformations, this representation may be more readable and allow the application of fine grain control over the transformation. If the transformation expression was left unspecified for some of the attributes of the target type, they are evaluated to null when the transformation is applied.
The DBMS_TSDP_MANAGE package provides an interface to import and manage sensitive columns and sensitive column types in the database, and is used in conjunction with the DBMS_TSDP_PROTECT package with regard to transparent sensitive data protection (TSDP) policies.

DBMS_TSDP_MANAGE is available with the Enterprise Edition only.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_TSDP_MANAGE Subprograms

Related Topics

- DBMS_TSDP_PROTECT
  The DBMS_TSDP_PROTECT package provides an interface to configure transparent sensitive data protection (TSDP) policies in conjunction with the DBMS_TSDP_MANAGE package.

See Also:
Oracle Database Security Guide

178.1 DBMS_TSDP_MANAGE Overview

The DBMS_TSDP_MANAGE package lets you manage sensitive columns and sensitive types in the Oracle database.

The identified sensitive columns are classified based on the sensitive types. By Using the DBMS_TSDP_PROTECT package to create a policy that protects data for a given class based on a column type rather than the data itself, you can then manage security for these types in a uniform fashion and modify the settings to accommodate changing compliance regulations.

You also can export the policies to other databases, when you perform a full export using Data Pump. You cannot export the policy itself, but an export of the database will include the TSDP policies.
178.2 DBMS_TSDP_MANAGE Security Model

All procedures are executed with invoker's rights. The DBMS_TSDP_MANAGE package is owned by SYS.

The EXECUTE privilege on this package should be granted as appropriate. Typically, an application database administrator should be granted the EXECUTE privilege for this package, while the DBMS_TSDP_PROTECT package would be governed by a security administrator.

178.3 Summary of DBMS_TSDP_MANAGE Subprograms

This table lists the DBMS_TSDP_MANAGE subprograms and briefly describes them.

Table 178-1 DBMS_TSDP_MANAGE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_SENSITIVE_COLUMN</td>
<td>Adds a column to the sensitive column list</td>
</tr>
<tr>
<td>ADD_SENSITIVE_TYPE</td>
<td>Creates and adds a sensitive column type to the list of sensitive column types in the database</td>
</tr>
<tr>
<td>ALTER_SENSITIVE_COLUMN</td>
<td>Alters the sensitive type and/or the comment of a column in the sensitive column list.</td>
</tr>
<tr>
<td>DROP_SENSITIVE_COLUMN</td>
<td>Removes columns from the sensitive column list</td>
</tr>
<tr>
<td>DROP_SENSITIVE_TYPE</td>
<td>Drops a sensitive column type from the list sensitive column types in the database</td>
</tr>
<tr>
<td>DROP_SENSITIVE_TYPE_SOURCE</td>
<td>Drops sensitive column types corresponding to a source from the list sensitive column types in the database</td>
</tr>
<tr>
<td>IMPORT_DISCOVERY_RESULT</td>
<td>Imports sensitive columns from an external source. This can be an Application Data Model (ADM) from an Oracle Enterprise Manager Cloud Control instance</td>
</tr>
<tr>
<td>IMPORT_SENSITIVE_TYPES</td>
<td>Imports a list of sensitive column types from a source</td>
</tr>
<tr>
<td>REMOVE_DISCOVERY_RESULT</td>
<td>Removes sensitive columns corresponding to an Application Data Model (ADM) from an Oracle Enterprise Manager Cloud Control instance.</td>
</tr>
</tbody>
</table>

178.3.1 ADD_SENSITIVE_COLUMN Procedure

This procedure adds a column to the sensitive column list.

Syntax

```sql
DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN (
    schema_name        IN VARCHAR2,
    table_name         IN VARCHAR2,
    column_name        IN VARCHAR2,
    sensitive_type     IN VARCHAR2,
    user_comment       IN VARCHAR2 DEFAULT NULL);
```
Parameters

**Table 178-2  ADD_SENSITIVE_COLUMN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema to which the column belongs</td>
</tr>
<tr>
<td>table_name</td>
<td>Table containing the column</td>
</tr>
<tr>
<td>column_name</td>
<td>Sensitive column name</td>
</tr>
<tr>
<td>sensitive_type</td>
<td>Identifier of the sensitive column type</td>
</tr>
<tr>
<td>user_comment</td>
<td>User comment regarding the sensitive column</td>
</tr>
</tbody>
</table>

Examples

Add a column SAL in SCOTT.EMP:

```sql
DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN (
  schema_name        => 'SCOTT',
  table_name         => 'EMP',
  column_name        => 'SAL',
  sensitive_type     => 'SALARY_TYPE',
  user_comment       => 'Salary column');
```

178.3.2 ALTER_SENSITIVE_COLUMN Procedure

This procedure alters the Sensitive Type and/or the Comment of a Column in the sensitive column list.

Syntax

```sql
DBMS_TSDP_MANAGE.ALTER_SENSITIVE_COLUMN (
  schema_name        IN VARCHAR2,
  table_name         IN VARCHAR2,
  column_name        IN VARCHAR2,
  sensitive_type     IN VARCHAR2,
  user_comment       IN VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 178-3  ALTER_SENSITIVE_COLUMN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema to which the column belongs</td>
</tr>
<tr>
<td>table_name</td>
<td>Table containing the column</td>
</tr>
<tr>
<td>column_name</td>
<td>Sensitive column name</td>
</tr>
<tr>
<td>sensitive_type</td>
<td>Identifier of the sensitive column type</td>
</tr>
<tr>
<td>user_comment</td>
<td>User comment regarding the sensitive column</td>
</tr>
</tbody>
</table>

Examples

Alter the column SAL in SCOTT.EMP that is listed in the sensitive column list:
DBMS_TSDP_MANAGE.ALTER_SENSITIVE_COLUMN (  
  schema_name        => 'SCOTT',  
  table_name         => 'EMP',  
  column_name        => 'SAL',  
  sensitive_type     => 'FINANCE_Type',  
  user_comment       => 'Finance Type. Earlier categorized as Salary Type');

178.3.3 ADD_SENSITIVE_TYPE Procedure

This procedure creates and adds a sensitive column type to the list sensitive column types in the database.

Syntax

DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE (  
  sensitive_type     IN  VARCHAR2,  
  user_comment       IN  VARCAHR2 DEFAULT NULL);

Parameters

Table 178-4  ADD_SENSITIVE_TYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensitive_type</td>
<td>Name of the sensitive column type</td>
</tr>
<tr>
<td>user_comment</td>
<td>User comment regarding the sensitive column</td>
</tr>
</tbody>
</table>

Examples

Add a sensitive column type called SALARY_TYPE that is intended to be associated with columns containing salary data:

DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE (  
  sensitive_type     => 'SALARY_TYPE',  
  user_comment       => 'Salary data');

178.3.4 DROP_SENSITIVE_COLUMN Procedure

This procedure removes columns from the sensitive column list.

Syntax

DBMS_TSDP_MANAGE.DROP_SENSITIVE_COLUMN (  
  schema_name        IN VARCHAR2 DEFAULT '%',  
  table_name         IN VARCAHR2 DEFAULT '%',  
  column_name        IN VARCAHR2 DEFAULT '%');

Parameters

Table 178-5  DROP_SENSITIVE_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema to which the column belongs</td>
</tr>
<tr>
<td>table_name</td>
<td>Table containing the column</td>
</tr>
</tbody>
</table>
Table 178-5  (Cont.) *DROP_SENSITIVE_COLUMN* Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_name</td>
<td>Sensitive column name</td>
</tr>
</tbody>
</table>

**Examples**

Remove column SAL in SCOTT.EMP from the sensitive column list:

```sql
DBMS_TSDP_MANAGE.DROP_SENSITIVE_COLUMN (
    schema_name        => 'SCOTT',
    table_name         => 'EMP',
    column_name        => 'SAL');
```

### 178.3.5 DROP_SENSITIVE_TYPE Procedure

This procedure drops a sensitive column type from the list sensitive column types in the database.

**Syntax**

```sql
DBMS_TSDP_MANAGE.DROP_SENSITIVE_TYPE (
    sensitive_type     IN VARCHAR2);
```

**Parameters**

Table 178-6  *DROP_SENSITIVE_TYPE* Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensitive_type</td>
<td>Name of the sensitive column type to be dropped</td>
</tr>
</tbody>
</table>

**Examples**

To drop `SALARY_TYPE`:

```sql
DBMS_TSDP_MANAGE.DROP_SENSITIVE_TYPE (
    sensitive_type     => 'SALARY_TYPE');
```

### 178.3.6 DROP_SENSITIVE_TYPE_SOURCE Procedure

This procedure drops sensitive column types corresponding to a source from the list sensitive column types in the database.

**Syntax**

```sql
DBMS_TSDP_MANAGE.DROP_SENSITIVE_TYPE_SOURCE (
    source      IN VARCHAR2);
```
Parameters

Table 178-7  DROP_SENSITIVE_TYPE_SOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Name of the source</td>
</tr>
</tbody>
</table>

Examples

To drop all sensitive column types corresponding to an Application Data Model (ADM) from an Oracle Enterprise Manager Cloud Control instance, `ADM_DEMO`:

```sql
DBMS_TSDP_MANAGE.DROP_SENSITIVE_TYPE_SOURCE (
  source      =>  'ADM_DEMO');
```

178.3.7 IMPORT_DISCOVERY_RESULT Procedure

This procedure can be used to import sensitive columns, along with the associated sensitive types, from an external source. The external source can be an Application Data Model (ADM) instance from Oracle Enterprise Manager Cloud Control.

Syntax

```sql
DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT (
  discovery_result         IN  CLOB,
  discovery_source         IN  VARCHAR2,
  force                    IN  FORCE DEFAULT FALSE);
```

```sql
DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT (
  discovery_result         IN  XMLTYPE,
  discovery_source         IN  VARCHAR2,
  force                    IN  FORCE DEFAULT FALSE);
```

Parameters

Table 178-8  IMPORT_DISCOVERY_RESULT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discovery_result</td>
<td>List of sensitive columns, along with the optional list of (the definitions of) the sensitive column types in XML format (possibly as a CLOB).</td>
</tr>
<tr>
<td>discovery_source</td>
<td>Source of the import. The discovery_source name identifies the list of imported sensitive columns. In case of ADM, this should be the ADM name.</td>
</tr>
</tbody>
</table>
Table 178-8  (Cont.) IMPORT_DISCOVERY_RESULT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>Specifies if the discovery result should be imported or not when the discovery result contains columns sensitive columns that are already identified as sensitive by another source.</td>
</tr>
<tr>
<td></td>
<td>- FALSE (default) - the discovery result will not be imported in case of conflicting columns. None of the columns and the sensitive types are imported.</td>
</tr>
<tr>
<td></td>
<td>- TRUE - the discovery result is imported and the attributes of the conflicting columns is set based on the incoming discovery result</td>
</tr>
</tbody>
</table>

Examples

Import the list of sensitive columns of ADM instance, ADM_Demo:

```
DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT (
    discovery_results => xml_adm_result,
    discovery_source => 'ADM_Demo');
```

178.3.8 IMPORT_SENSITIVE_TYPES Procedure

This procedure imports a list of sensitive column types from a source.

Syntax

```
DBMS_TSDP_MANAGE.IMPORT_SENSITIVE_TYPES (
    sensitive_types    IN CLOB,
    source             IN VARCHAR2);
```

```
DBMS_TSDP_MANAGE.IMPORT_SENSITIVE_TYPES (
    sensitive_types    IN XMLTYPE,
    source             IN VARCHAR2);
```

Parameters

Table 178-9  IMPORT_SENSITIVE_TYPES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensitive_types</td>
<td>List of sensitive column types in XML Format (possibly as a CLOB)</td>
</tr>
<tr>
<td>source</td>
<td>Source of the import. The source identifies the list of imported sensitive column types. In case of Application Data Model (ADM) from an Oracle Enterprise Manager Cloud Control instance, this should be the ADM name.</td>
</tr>
</tbody>
</table>

Examples

Import the list of sensitive column types of ADM instance, ADM_Demo:

```
DBMS_TSDP_MANAGE.IMPORT_SENSITIVE_TYPES (
    sensitive_types => xml_adm_result,
    source => 'ADM_Demo');
```
178.3.9 REMOVE_DISCOVERY_RESULT Procedure

This procedure removes sensitive columns corresponding to an Application Data Model (ADM) from an Oracle Enterprise Manager Cloud Control instance.

Syntax

```sql
DBMS_TSDP_MANAGE.REMOVE_DISCOVERY_RESULT (
    discovery_source  IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discovery_source</td>
<td>Source of the import. In case of ADM, this should be the ADM name, the results of which is to be removed.</td>
</tr>
</tbody>
</table>

Examples

Remove the sensitive columns corresponding to ADM instance, ADM_Demo:

```sql
DBMS_TSDP_MANAGE.REMOVE_DISCOVERY_RESULT (
    discovery_source => 'ADM_Demo');
```
The DBMS_TSDP_PROTECT package provides an interface to configure transparent sensitive data protection (TSDP) policies in conjunction with the DBMS_TSDP_MANAGE package.

DBMS_TSDP_PROTECT is available with the Enterprise Edition only.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Data Structures
- Summary of DBMS_TSDP_PROTECT Subprograms

Related Topics

- DBMS_TSDP_MANAGE
  The DBMS_TSDP_MANAGE package provides an interface to import and manage sensitive columns and sensitive column types in the database, and is used in conjunction with the DBMS_TSDP_PROTECT package with regard to transparent sensitive data protection (TSDP) policies.

See Also:

Oracle Database Security Guide

179.1 DBMS_TSDP_PROTECT Overview

Use the DBMS_TSDP_PROTECT package to create transparent sensitive data protection policies, configure protection by associating the policies with sensitive types, and to enable and disable the configured protection.

Sensitive types can be added using the DBMS_TSDP_MANAGE package.

179.2 DBMS_TSDP_PROTECT Security Model

All procedures are executed with invoker's rights. Typically, a security administrator should have the EXECUTE privilege for this package.
179.3 DBMS_TSDP_PROTECT Constants

DBMS_TSDP_PROTECT defines the TSDP_PARAM_MAX constant for use when specifying parameter values.

This constant is described in the following table.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSDP_PARAM_MAX</td>
<td>INTEGER</td>
<td>4000</td>
<td>Maximum length of the parameter value that can be specified in FEATURE_OPTIONS</td>
</tr>
</tbody>
</table>

179.4 DBMS_TSDP_PROTECT Data Structures

The DBMS_TSDP_PROTECT package defines two TABLE types.

Table Types

• FEATURE_OPTIONS Table Type

• POLICY_CONDITIONS Table Type

179.4.1 FEATURE_OPTIONS Table Type

The following type is an associative array of VARCHAR2(TSDP_PARAM_MAX) that is indexed by VARCHAR2(M_IDEN).

Syntax

```plaintext
TYPE FEATURE_OPTIONS IS TABLE OF VARCHAR2(TSDP_PARAM_MAX)
INDEX BY VARCHAR2(M_IDEN);
```

179.4.2 POLICY_CONDITIONS Table Type

The following type is an associative array of VARCHAR2(TSDP_PARAM_MAX) that is indexed by PLS_INTEGER.

Syntax

```plaintext
TYPE POLICY_CONDITIONS IS TABLE OF VARCHAR2(TSDP_PARAM_MAX)
INDEX BY PLS_INTEGER;
```

179.5 Summary of DBMS_TSDP_PROTECT Subprograms

This table lists the DBMS_TSDP_PROTECT subprograms and briefly describes them.
Table 179-2  DBMS_TSDP_PROTECT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_POLICY Procedure</td>
<td>Creates a TDSP policy</td>
</tr>
<tr>
<td>ALTER_POLICY Procedure</td>
<td>Alters a TDSP policy</td>
</tr>
<tr>
<td>ASSOCIATE_POLICY Procedure</td>
<td>Associates or disassociates a TDSP policy with a sensitive column type</td>
</tr>
<tr>
<td>DISABLE_PROTECTION_COLUMN Procedure</td>
<td>Disables protection for columns</td>
</tr>
<tr>
<td>DISABLE_PROTECTION_SOURCE Procedure</td>
<td>Disables protection based on the source of truth for the sensitive columns</td>
</tr>
<tr>
<td>DISABLE_PROTECTION_TYPE Procedure</td>
<td>Disables protection for a sensitive column type</td>
</tr>
<tr>
<td>DROP_POLICY Procedure</td>
<td>Removes a TDSP policy</td>
</tr>
<tr>
<td>ENABLE_PROTECTION_COLUMN Procedure</td>
<td>Enables protection for columns</td>
</tr>
<tr>
<td>ENABLE_PROTECTION_SOURCE Procedure</td>
<td>Enables protection based on the source of truth for the sensitive columns</td>
</tr>
<tr>
<td>ENABLE_PROTECTION_TYPE Procedure</td>
<td>Enables protection for a sensitive column type</td>
</tr>
</tbody>
</table>

179.5.1 ADD_POLICY Procedure

This procedure creates a TDSP policy.

Syntax

```sql
DBMS_TSDP_PROTECT.ADD_POLICY (reur
    policy_name              IN VARCHAR2,
    security_feature         IN PLS_INTEGER,
    policy_enable_options    IN FEATURE_OPTIONS,
    policy_apply_condition   IN POLICY_CONDITION DEFAULT TSDP$default_condition);
```

Parameters

Table 179-3  ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_name</td>
<td>Name of the policy being created. The maximum length for this identifier is M_IDEN. This follows the Oracle naming convention.</td>
</tr>
<tr>
<td>security_feature</td>
<td>Oracle security feature with which the policy is associated. Allowed values:</td>
</tr>
<tr>
<td>policy_enable_options</td>
<td>Initialized with the parameter-value pairs corresponding to the security_feature setting</td>
</tr>
<tr>
<td>policy_apply_condition</td>
<td>DBMS_TSDP_PROTECT.REDACT</td>
</tr>
<tr>
<td></td>
<td>DBMS_TSDP_PROTECT.VPD</td>
</tr>
<tr>
<td></td>
<td>DBMS_TSDP_PROTECT.UNIFIED_AUDIT</td>
</tr>
<tr>
<td></td>
<td>DBMS_TSDP_PROTECT.FINE_GRAINED_AUDIT</td>
</tr>
<tr>
<td></td>
<td>DBMS_TSDP_PROTECT.COLUMN_ENCRYPTION</td>
</tr>
</tbody>
</table>
### ADD_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_apply_condition</td>
<td>Initialized with the property-value pairs that must be satisfied in order to apply the corresponding policy_enable_options. This is an associative array with Property as the key (PLS_INTEGER). Example: example_policy_condition(Property)=property_value. Permissible values for Property:</td>
</tr>
<tr>
<td></td>
<td>· DBMS_TSDP_PROPERTY.DATATYPE</td>
</tr>
<tr>
<td></td>
<td>· DBMS_TSDP_PROPERTY.LENGTH</td>
</tr>
<tr>
<td></td>
<td>· DBMS_TSDP_PROPERTY.PARENT_SCHEMA</td>
</tr>
<tr>
<td></td>
<td>· DBMS_TSDP_PROPERTY.PARENT_TABLE</td>
</tr>
</tbody>
</table>

**Usage Notes**

To create the TDSP policy, you must include the procedure in an anonymous block that defines the type of security feature that will use the policy and conditions to test when the policy is enabled. For more information, see Oracle Database Security Guide.

**Examples**

Create a policy PARTIAL_MASK_POLICY:

```sql
DECLARE
  redact_feature_options DBMS_TSDP_PROTECT.FEATURE_OPTIONS;
  policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
BEGIN
  redact_feature_options ('expression') := 
    'SYS_CONTEXT(''USERENV'',''SESSION_USER'')      =''APPUSER''';
  redact_feature_options ('function_type') := 'DBMS_REDACT.PARTIAL';
  redact_feature_options ('function_parameters') := 'STR, VVVVVVVVV,VVVVVVVVV,'
    , 1, 6';
  policy_conditions(DBMS_TSDP_PROTECT.DATATYPE) := 'VARCHAR2';
  DBMS_TSDP_PROTECT.ADD_POLICY
    ('PARTIAL_MASK_POLICY', DBMS_TSDP_PROTECT.REDACT, redact_feature_options,
     policy_conditions);
END;
```

### 179.5.2 ALTER_POLICY Procedure

This procedure alters an existing TDSP policy

**Syntax**

```sql
DBMS_TSDP_PROTECT.ALTER_POLICY (
  policy_name IN VARCHAR2,
  policy_enable_options IN FEATURE_OPTIONS,
  policy_apply_condition IN POLICY_CONDITION default TSDP$default_condition);
```
Parameters

Table 179-4  ALTER_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_name</td>
<td>Name of the policy to alter</td>
</tr>
<tr>
<td>policy_enable_options</td>
<td>Initialized with the parameter-value pairs corresponding to the security feature</td>
</tr>
<tr>
<td>policy_apply_condition</td>
<td>Initialized with the property-value pairs that must be satisfied in order to apply the corresponding policy_enable_options. This is an associative array with Property as the key (PLS_INTEGER). Example: example_policy_condition(Property)=property_value. Permissible values for Property:</td>
</tr>
<tr>
<td></td>
<td>• DBMS_TSDP_PROPERTY.DATATYPE</td>
</tr>
<tr>
<td></td>
<td>• DBMS_TSDP_PROPERTY.LENGTH</td>
</tr>
<tr>
<td></td>
<td>• DBMS_TSDP_PROPERTY.PARENT_SCHEMA</td>
</tr>
<tr>
<td></td>
<td>• DBMS_TSDP_PROPERTY.PARENT_TABLE</td>
</tr>
</tbody>
</table>

Usage Notes

• If the policy_apply_condition matches an existing condition for the policy, then the corresponding enable options are updated with policy_enable_options.

• If the policy_apply_condition does not match any existing condition for the policy, the combination of policy_enable_options and policy_apply_condition is added to the policy.

Examples

Add a new combination of policy_apply_condition and policy_enable_options to an existing policy PARTIAL_MASK_POLICY:

```sql
DECLARE
    redact_feature_options DBMS_TSDP_PROTECT.FEATURE_OPTIONS;
    policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
BEGIN
    redact_feature_options ('expression') := 'SYS_CONTEXT(''USERENV'',''SESSION_USER'')='APPUSER'';
    redact_feature_options ('function_type') := 'DBMS_REDACT.PARTIAL';
    redact_feature_options ('function_parameters') := 'STR, VVVVVVVVV, VVVVVVVVV, *, 1, 6';
    policy_conditions (DBMS_TSDP_PROTECT.DATATYPE) := 'VARCHAR2';
    DBMS_TSDP_PROTECT.ALTER_POLICY ('PARTIAL_MASK_POLICY', redact_feature_options, policy_conditions);
END;
```

179.5.3 ASSOCIATE_POLICY Procedure

This procedure associates or disassociates a TSDP policy with a sensitive column type.

Syntax

```sql
DBMS_TSDP_PROTECT.ASSOCIATE_POLICY (  
    policy_name   IN  VARCHAR2,
```
sensitive_type     IN  VARCHAR2,
associate          IN  BOOLEAN DEFAULT TRUE);

Parameters

Table 179-5  ASSOCIATE_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_name</td>
<td>Name of the TDSP policy</td>
</tr>
<tr>
<td>sensitive_type</td>
<td>Name of the sensitive column type:</td>
</tr>
<tr>
<td>associate</td>
<td>Associate or Disassociate. TRUE implies Associate</td>
</tr>
</tbody>
</table>

Usage Notes

Both the policy and the sensitive column type should exist in the database.

Examples

Associate PARTIAL_MASK_POLICY with SSN_TYPE:

```sql
DBMS_TSDP_PROTECT.ASSOCIATE_POLICY ('PARTIAL_MASK_POLICY', 'SSN_TYPE');
```

179.5.4 DISABLE_PROTECTION_COLUMN Procedure

This procedure disables protection for columns.

Syntax

```sql
DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN (
    schema_name        IN  VARCHAR2 DEFAULT '%',
    table_name         IN  VARCHAR2 DEFAULT '%',
    column_name        IN  VARCHAR2 DEFAULT '%',
    policy_name        IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 179-6  DISABLE_PROTECTION_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema containing the column</td>
</tr>
<tr>
<td>table_name</td>
<td>Table containing the column</td>
</tr>
<tr>
<td>column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>policy_name</td>
<td>Optional policy name. If given, only this policy is disabled.</td>
</tr>
</tbody>
</table>

Examples

Disable TSDP policies associated with the corresponding sensitive column types for columns that reside in schema with name like %PAYROLL%, table name like EMP%, and column name like SAL%:

```sql
EXEC DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN ('%PAYROLL%', 'EMP%', 'SAL%');
```
179.5.5 DISABLE_PROTECTION_SOURCE Procedure

This procedure disables protection based on the source of truth for the sensitive columns.

Syntax

```sql
DBMS_TSDP_PROTECT.DISABLE_PROTECTION_SOURCE (discovery_sourcename IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discovery_sourcename</td>
<td>Name of the discovery source. This could be the Application Data Model (ADM) name or the database user.</td>
</tr>
</tbody>
</table>

Examples

Disable protection for all columns corresponding to ADM_Demo:

```sql
DBMS_TSDP_PROTECT.DISABLE_PROTECTION_SOURCE ('ADM_Demo');
```

179.5.6 DISABLE_PROTECTION_TYPE Procedure

This procedure disables protection for a sensitive column type.

Syntax

```sql
DBMS_TSDP_PROTECT.DISABLE_PROTECTION_TYPE (sensitive_type IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensitive_type</td>
<td>Name of the sensitive column type</td>
</tr>
</tbody>
</table>

Examples

Disable protection for all columns identified by SSN_TYPE:

```sql
DBMS_TSDP_PROTECT.DISABLE_PROTECTION_TYPE ('SSN_TYPE');
```

179.5.7 DROP_POLICY Procedure

This procedure removes a TDSP policy or one of its condition-enable_options combinations.

Syntax

```sql
DBMS_TSDP_PROTECT.DROP_POLICY (policy_name IN VARCHAR2, ...
```
policy_apply_condition IN POLICY_CONDITIONS);

DBMS_TSDP_PROTECT.DROP_POLICY (policy_name IN VARCHAR2);

Parameters

Table 179-9  DROP_POLICY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_name</td>
<td>Name of the policy to drop</td>
</tr>
<tr>
<td>policy_apply_condition</td>
<td>To be initialized with the relevant condition</td>
</tr>
</tbody>
</table>

Usage Notes

- The combination of policy_conditions and policy_enable_options can be dropped from a TSDP policy by giving the policy_apply_condition parameter. The default condition-default options combination can also be dropped (if it exists for the policy) by passing an empty associative array of type DBMS_TSDP_PROTECT.POLICY_CONDITION.
- If the condition-enable_options combination that is being dropped is the last condition-enable_options combination for the policy, the policy itself is dropped.
- A policy can be completely dropped by using the overloaded of the procedure that takes only policy_name.
- A policy or one of its conditions can be dropped only if the policy is not associated with any sensitive column type. This also means that a policy that is being dropped is not enabled on any column (object).

Examples

Dropping the condition-enable_options combination based on a specific condition:

DECLARE
  policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
BEGIN
  policy_conditions (DBMS_TSDP_PROTECT.DATATYPE) := 'VARCHAR2';
  DBMS_TSDP_PROTECT.DROP_POLICY ('PARTIAL_MASK_POLICY', policy_conditions);
END;

The default condition-enable_options combination can be dropped by passing an empty associative array of type DBMS_TSDP_PROTECT.POLICY_CONDITIONS for the policy_apply_condition parameter:

DECLARE
  policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
BEGIN
  DBMS_TSDP_PROTECT.DROP_POLICY ('redact_partial_cc', policy_conditions);
END;

Dropping a TSDP policy:

BEGIN
  DBMS_TSDP_PROTECT.DROP_POLICY(
    policy_name => 'PARTIAL_MASK_POLICY');
END;
179.5.8 ENABLE_PROTECTION_COLUMN Procedure

This procedure enables protection for columns.

Syntax

```sql
DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN (  
    schema_name        IN  VARCHAR2 DEFAULT '%',
    table_name         IN  VARCHAR2 DEFAULT '%',
    column_name        IN  VARCHAR2 DEFAULT '%',
    policy_name        IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema containing the column</td>
</tr>
<tr>
<td>table_name</td>
<td>Table containing the column</td>
</tr>
<tr>
<td>column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>policy_name</td>
<td>Optional policy name. If given, only this policy is enabled.</td>
</tr>
</tbody>
</table>

Usage Notes

- Only a TSDP Policy that is associated with the sensitive column type of the sensitive column can be enabled using this Procedure.
- LIKE condition is used for schema_name, table_name and column_name. AND semantics is followed.

Examples

Enable TSDP policies associated with the corresponding sensitive column types for columns that reside in schema with name like %PAYROLL%, table name like EMP%, and column name like SAL%:

```sql
DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN ('%PAYROLL%', 'EMP%', 'SAL%');
```

179.5.9 ENABLE_PROTECTION_SOURCE Procedure

This procedure enables protection based on the source of truth for the sensitive columns.

Syntax

```sql
DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE (  
    discovery_sourcename    IN  VARCHAR2);
```
Parameters

Table 179-11  ENABLE_PROTECTION_SOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discovery_sourcename</td>
<td>Name of the discovery source. This could be the Application Data Model (ADM) name or the database user.</td>
</tr>
</tbody>
</table>

Examples

Enable protection for all columns corresponding to ADM_Demo:

DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE ('ADM_Demo');

179.5.10 ENABLE_PROTECTION_TYPE Procedure

This procedure enables protection for a sensitive column type.

Syntax

DBMS_TSDP_PROTECT.ENABLE_PROTECTION_TYPE (sensitive_type IN VARCHAR2);

Parameters

Table 179-12  ENABLE_PROTECTION_TYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensitive_type</td>
<td>Name of the sensitive column type</td>
</tr>
</tbody>
</table>

Examples

Enable protection for all columns identified by SSN_TYPE:

DBMS_TSDP_PROTECT.ENABLE_PROTECTION_TYPE ('SSN_TYPE');
The DBMS_TTS package checks if the transportable set is self-contained. All violations are inserted into a temporary table that can be selected from the view TRANSPORT_SET_VIOLATIONS.

This chapter contains the following topics:

- Security Model
- Exceptions
- Operational Notes
- Summary of DBMS_TTS Subprograms

See Also:
- Oracle Database Administrator's Guide
- Oracle Database Upgrade Guide

180.1 DBMS_TTS Security Model

Only users having the execute_catalog_role can execute this procedure. This role is initially only assigned to user SYS.

180.2 DBMS_TTS Exceptions

The DBMS_TTS package creates exceptions for missing or invalid transportable tablespaces.

```sql
ts_not_found  EXCEPTION;
PRAGMA exception_init(ts_not_found, -29304);
ts_not_found_num NUMBER := -29304;

invalid_ts_list  EXCEPTION;
PRAGMA exception_init(invalid_ts_list, -29346);
invalid_ts_list_num NUMBER := -29346;

sys_or_tmp_ts     EXCEPTION;
PRAGMA exception_init(sys_or_tmp_ts, -29351);
sys_or_tmp_ts_num NUMBER := -29351;
```
180.3 DBMS_TTS Operational Notes

With respect to transportable tablespaces, disabled and enabled referential integrity constraints are handled differently.

- A disabled referential integrity constraint does not violate the transportability rules and is dropped during the import phase.
- An enabled referential integrity constraint violates the transportability rules if it references a table in a tablespace outside the transportable set.

180.4 Summary of DBMS_TTS Subprograms

The two procedures listed in the table are designed to be called by database administrators.

**Table 180-1  DBMS_TTS Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNGRADE Procedure</td>
<td>Downgrades transportable tablespace-related data</td>
</tr>
<tr>
<td>TRANSPORT_SET_CHECK Procedure</td>
<td>Checks if a set of tablespaces (to be transported) is self-contained</td>
</tr>
</tbody>
</table>

180.4.1 DOWNGRADE Procedure

This procedure downgrades transportable tablespace related data.

**Syntax**

```sql
DBMS_TTS.DOWNGRADE;
```

180.4.2 TRANSPORT_SET_CHECK Procedure

This procedure checks if a set of tablespaces (to be transported) is self-contained. After calling this procedure, the user may select from a view to see a list of violations, if there are any.

**Syntax**

```sql
DBMS_TTS.TRANSPORT_SET_CHECK (ts_list IN CLOB,
                               incl_constraints IN BOOLEAN DEFAULT FALSE,
                               full_check IN BOOLEAN DEFAULT FALSE);
```

**Parameters**

**Table 180-2  TRANSPORT_SET_CHECK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts_list</td>
<td>List of one or more tablespaces, separated by comma.</td>
</tr>
</tbody>
</table>
Table 180-2  (Cont.) TRANSPORT_SET_CHECK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incl_constraints</td>
<td>TRUE if you want to count in referential integrity constraints when examining if the set of tablespaces is self-contained. (The incl_constraints parameter is a default so that TRANSPORT_SET_CHECK will work if it is called with only the ts_list argument.)</td>
</tr>
<tr>
<td>full_check</td>
<td>Indicates whether a full or partial dependency check is required. If TRUE, treats all IN and OUT pointers (dependencies) and captures them as violations if they are not self-contained in the transportable set. The parameter should be set to TRUE for TSPITR or if a strict version of transportable is desired. By default the parameter is set to FALSE. It will only consider OUT pointers as violations.</td>
</tr>
</tbody>
</table>

Examples

If the view does not return any rows, then the set of tablespaces is self-contained. For example,

```sql
SQLPLUS> EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('foo,bar', TRUE);
SQLPLUS> SELECT * FROM TRANSPORT_SET_VIOLATIONS;
```
DBMS_TYPES

The `DBMS_TYPES` package consists of constants, which represent the built-in and user-defined types.

This chapter contains the following topics:

- Constants
- Exceptions

### 181.1 DBMS_TYPES Constants

The `DBMS_TYPES` package defines several constants to use when specifying parameter values.

These constants are listed in the following table.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DATA</td>
<td>Is only relevant if PieceWise is called, for a collection or any dataset. Denotes the end of collection/any dataset when all the elements have been accessed.</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>The operation succeeded.</td>
</tr>
<tr>
<td>TYPECODE_BDOUBLE</td>
<td>A NUMBER type</td>
</tr>
<tr>
<td>TYPECODE_BFILE</td>
<td>A BFILE type</td>
</tr>
<tr>
<td>TYPECODE_BFLOAT</td>
<td>A NUMBER type</td>
</tr>
<tr>
<td>TYPECODE_BLOB</td>
<td>A BLOB type</td>
</tr>
<tr>
<td>TYPECODE_CFILE</td>
<td>A CFILE type</td>
</tr>
<tr>
<td>TYPECODE_CHAR</td>
<td>A CHAR type</td>
</tr>
<tr>
<td>TYPECODE_CLOB</td>
<td>A CLOB type</td>
</tr>
<tr>
<td>TYPECODE_DATE</td>
<td>A DATE type</td>
</tr>
<tr>
<td>TYPECODE_INTERVAL_DS</td>
<td>An INTERVAL_DS type</td>
</tr>
<tr>
<td>TYPECODE_INTERVAL_YM</td>
<td>An INTERVAL_YM type</td>
</tr>
<tr>
<td>TYPECODE_MLSLABEL</td>
<td>An MLSLABEL type</td>
</tr>
<tr>
<td>TYPECODE_NAMEDCOLLECTION</td>
<td>A named collection (VARRAY/nested table) type</td>
</tr>
<tr>
<td>TYPECODE_NCHAR</td>
<td>A NCHAR type</td>
</tr>
<tr>
<td>TYPECODE_NCLOB</td>
<td>A NCLOB type</td>
</tr>
<tr>
<td>TYPECODE_NUMBER</td>
<td>A NUMBER type</td>
</tr>
<tr>
<td>TYPECODE_NVARCHAR2</td>
<td>A NVARCHAR2 type</td>
</tr>
<tr>
<td>TYPECODE_OBJECT</td>
<td>An OBJECT type</td>
</tr>
</tbody>
</table>
### Table 181-1 (Cont.) DBMS_TYPES Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPECODE_OPAQUE</td>
<td>An OPAQUE type</td>
</tr>
<tr>
<td>TYPECODE_RAW</td>
<td>A RAW type</td>
</tr>
<tr>
<td>TYPECODE_REF</td>
<td>A REF type</td>
</tr>
<tr>
<td>TYPECODE_TABLE</td>
<td>A nested table collection type</td>
</tr>
<tr>
<td>TYPECODE_TIMESTAMP</td>
<td>A TIMESTAMP type</td>
</tr>
<tr>
<td>TYPECODE_TIMESTAMP_LTZ</td>
<td>A TIMESTAMP_LTZ type</td>
</tr>
<tr>
<td>TYPECODE_TIMESTAMP_TZ</td>
<td>A TIMESTAMP_TZ type</td>
</tr>
<tr>
<td>TYPECODE_UROWID</td>
<td>A UROWID type</td>
</tr>
<tr>
<td>TYPECODE_VARCHAR2</td>
<td>A VARCHAR2 type</td>
</tr>
<tr>
<td>TYPECODE_VARCHAR</td>
<td>A VARCHAR type</td>
</tr>
<tr>
<td>TYPECODE_VARRAY</td>
<td>A VARCHAR collection type</td>
</tr>
</tbody>
</table>

### 181.2 DBMS_TYPES Exceptions

DBMS_TYPES throws these exceptions.

- INVALID_PARAMETERS
- INCORRECT_USAGE
- TYPE_MISMATCH
The DBMS_UMF package provides an interface for deploying the Remote Management Framework (RMF) for an Oracle Database. The RMF is used for collecting performance statistics for an Oracle Database.

See Also:

Oracle Database Performance Tuning Guide for more information about configuring the RMF for an Oracle Database.

This chapter contains the following topic:

• Summary of DBMS_UMF Subprograms

182.1 Summary of DBMS_UMF Subprograms

This topic lists the DBMS_UMF subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIGURE_NODE Procedure</td>
<td>Configures a node in the RMF topology</td>
</tr>
<tr>
<td>CREATE_LINK Procedure</td>
<td>Creates a database link between two nodes in the RMF topology</td>
</tr>
<tr>
<td>CREATE_TOPOLOGY Procedure</td>
<td>Creates the RMF topology</td>
</tr>
<tr>
<td>DROP_LINK Procedure</td>
<td>Removes a database link between two nodes in the RMF topology</td>
</tr>
<tr>
<td>DROP_TOPOLOGY Procedure</td>
<td>Deletes the RMF topology</td>
</tr>
<tr>
<td>ENABLE_SERVICE Procedure</td>
<td>Enables a service on a node in the RMF topology</td>
</tr>
<tr>
<td>GET_NODE_ID_LOCAL Function</td>
<td>Returns the node ID of a node in the RMF topology</td>
</tr>
<tr>
<td>GET_NODE_NAME_LOCAL Function</td>
<td>Returns the node name of a node in the RMF topology</td>
</tr>
<tr>
<td>GET_TARGET_ID Function</td>
<td>Returns the destination ID in the RMF topology</td>
</tr>
<tr>
<td>GET_TOPOLOGY_NAME_LOCAL Function</td>
<td>Returns the RMF topology name of the local node</td>
</tr>
<tr>
<td>QUERY_LINK_INFO Procedure</td>
<td>Returns the information about a database link in the RMF topology</td>
</tr>
<tr>
<td>QUERY_NODE_INFO Procedures</td>
<td>Returns the information about a node in the RMF topology</td>
</tr>
<tr>
<td>REGISTER_NODE Function and Procedure</td>
<td>Registers a node in the RMF topology</td>
</tr>
</tbody>
</table>
Table 182-1  (Cont.) DBMS_UMF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH_DESTINATION Procedure</td>
<td>Designates a source node as a destination node in the RMF topology</td>
</tr>
<tr>
<td>UNCONFIGURE_NODE Procedure</td>
<td>Resets the configuration of a node in the RMF topology</td>
</tr>
<tr>
<td>UNREGISTER_NODE Procedure</td>
<td>Removes the registration of a node in the RMF topology</td>
</tr>
</tbody>
</table>

182.1.1 CONFIGURE_NODE Procedure

This procedure configures a node that needs to be registered with the RMF topology. This procedure must be executed on the node that needs to be configured.

Syntax

```sql
DBMS_UMF.CONFIGURE_NODE(
    node_name          IN VARCHAR2 DEFAULT NULL,
    dblink_to_target   IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_name</td>
<td>Name of the node. Each node in the RMF topology must be assigned a unique name. If a name is not provided for a node in this procedure, then the value of the initialization parameter DB_UNIQUE_NAME is assigned as the name for the node by default.</td>
</tr>
<tr>
<td>dblink_to_target</td>
<td>Database link from this node to the destination node.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users SYS and SYS$UMF can execute this procedure.

182.1.2 CREATE_LINK Procedure

This procedure creates database links between two nodes in the RMF topology. This procedure must be executed only on the destination node.

Syntax

```sql
DBMS_UMF.CREATE_LINK(
    topology_name      IN VARCHAR2,
    node_a_name        IN VARCHAR2,
```
CREATE_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
<tr>
<td>node_a_name</td>
<td>Name of the first node.</td>
</tr>
<tr>
<td>node_b_name</td>
<td>Name of the second node.</td>
</tr>
<tr>
<td>dblink_a_to_b</td>
<td>Database link from the first node to the second node.</td>
</tr>
<tr>
<td>dblink_b_to_a</td>
<td>Database link from the second node to the first node.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users SYS and SYS$UMF can execute this procedure.

182.1.3 CREATE_TOPOLOGY Procedure

This procedure creates the RMF topology and designates the node on which it is executed as the destination node for that topology.

Syntax

```sql
DBMS_UMF.CREATE_TOPOLOGY(
    topology_name IN VARCHAR2);
```

Parameters

Table 182-4  CREATE_TOPOLOGY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users SYS and SYS$UMF can execute this procedure.
182.1.4 DROP_LINK Procedure

This procedure removes the database links between two nodes in the RMF topology. This procedure must be executed only on the destination node.

Syntax

```
DBMS_UMF.DROP_LINK(
    topology_name  IN VARCHAR2,
    node_a_name    IN VARCHAR2,
    node_b_name    IN VARCHAR2);
```

Parameters

Table 182-5  DROP_LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
<tr>
<td>node_a_name</td>
<td>Name of the first node.</td>
</tr>
<tr>
<td>node_b_name</td>
<td>Name of the second node.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users SYS and SYS$UMF can execute this procedure.

182.1.5 DROP_TOPOLOGY Procedure

This procedure deletes the RMF topology. This procedure must be executed only on the destination node.

Syntax

```
DBMS_UMF.DROP_TOPOLOGY(
    topology_name IN VARCHAR2);
```

Parameters

Table 182-6  DROP_TOPOLOGY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the topology to delete.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users SYS and SYS$UMF can execute this procedure.
182.1.6 ENABLE_SERVICE Procedure

This procedure enables a service, such as the AWR service, on a node in the RMF topology. This procedure must be executed only on the destination node.

Syntax

```sql
DBMS_UMF.ENABLE_SERVICE(
    topology_name   IN VARCHAR2,
    node_name       IN VARCHAR2,
    service_type    IN NUMBER);
```

Parameters

**Table 182-7  ENABLE_SERVICE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
<tr>
<td>node_name</td>
<td>Name of the node on which a specific service needs to be enabled.</td>
</tr>
<tr>
<td>service_type</td>
<td>Numeric constant identifying the service. The only allowed value for this parameter is <code>UMF_SERVICE_TYPE_AWR</code>, which is Automatic Workload Repository (AWR) service.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users `SYS` and `SYS$UMF` can execute this procedure.

182.1.7 GET_NODE_ID_LOCAL Function

This function returns the node ID of the node in the RMF topology on which this function is executed.

Syntax

```sql
DBMS_UMF.GET_NODE_ID_LOCAL(
    topology_name  IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

Parameters

**Table 182-8  GET_NODE_ID_LOCAL Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology with which the node is registered.</td>
</tr>
</tbody>
</table>
Return Value
Returns the node ID of the node in the RMF topology on which this function is executed.

Security Model
Only the database users **SYS** and **SYSSMF** can execute this function.

### 182.1.8 GET_NODE_NAME_LOCAL Function

This function returns the name of the node in the RMF topology on which this function is executed.

Syntax

```sql
DBMS_UMF.GET_NODE_NAME_LOCAL RETURN VARCHAR2;
```

Return Value
Returns the name of the node in the RMF topology on which this function is executed.

Security Model
Only the database users **SYS** and **SYSSMF** can execute this function.

### 182.1.9 GET_TARGET_ID Function

This function returns the ID of the destination node in the RMF topology. This function can be executed on any node in the RMF topology.

Syntax

```sql
DBMS_UMF.GET_TARGET_ID(
    topology_name  IN  VARCHAR2)
RETURN NUMBER;
```

Parameters

**Table 182-9  GET_TARGET_ID Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
</tbody>
</table>

Return Value
Returns the ID of the destination node in the RMF topology.

Security Model
Only the database users **SYS** and **SYSSMF** can execute this procedure.
182.1.10 GET_TOPOLOGY_NAME_LOCAL Function

This function returns the name of the active RMF topology of the node on which this function is executed.

Syntax

```
DBMS_UMF.GET_TOPOLOGY_NAME_LOCAL RETURN VARCHAR2;
```

Return Value

Returns the name of the active RMF topology of the node on which this function is executed.

Security Model

Only the database users `SYS` and `SYS$UMF` can execute this procedure.

182.1.11 QUERY_LINK_INFO Procedure

This procedure returns the name of the database link between two nodes in the RMF topology. This procedure can be executed on any node in the RMF topology.

Syntax

```
DBMS_UMF.QUERY_LINK_INFO(
    topology_name      IN   VARCHAR2,
    from_node_id       IN   NUMBER,
    to_node_id         IN   NUMBER,
    link_name          OUT  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
<tr>
<td>from_node_id</td>
<td>Node ID of the first node.</td>
</tr>
<tr>
<td>to_node_id</td>
<td>Node ID of the second node.</td>
</tr>
<tr>
<td>link_name</td>
<td>Name of the database link from the first node to the second node returned by the procedure.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users `SYS` and `SYS$UMF` can execute this procedure.
182.1.12 QUERY_NODE_INFO Procedures

This procedure returns information about a node in the RMF topology. This procedure can be executed on any node in the RMF topology.

Syntax

```
DBMS_UMF.QUERY_NODE_INFO(
    topology_name      IN    VARCHAR2  DEFAULT NULL,
    node_name          IN    VARCHAR2,
    node_id            OUT   NUMBER);

DBMS_UMF.QUERY_NODE_INFO(
    node_id            IN    NUMBER,
    topology_name      OUT   VARCHAR2,
    node_name          OUT   VARCHAR2);
```

Parameters

Table 182-11  QUERY_NODE_INFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology with which the node is registered.</td>
</tr>
<tr>
<td>node_name</td>
<td>Name of the node.</td>
</tr>
<tr>
<td>node_id</td>
<td>Identifier of the node.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users SYS and SYS$UMF can execute this procedure.

182.1.13 REGISTER_NODE Function and Procedure

This function and procedure registers a node with the RMF topology. This procedure and function must be executed only on the destination node in the RMF topology.

Syntax

```
DBMS_UMFREGISTER_NODE(
    topology_name     IN    VARCHAR2,
    node_name         IN    VARCHAR2,
    dblink_to_node    IN    VARCHAR2 DEFAULT NULL,
    dblink_from_node  IN    VARCHAR2 DEFAULT NULL,
    as_source         IN    VARCHAR2 DEFAULT 'TRUE',
    as_candidate_target IN    VARCHAR2 DEFAULT 'FALSE');

DBMS_UMFREGISTER_NODE(
    topology_name     IN    VARCHAR2,
    node_name         IN    VARCHAR2,
    dblink_to_node    IN    VARCHAR2 DEFAULT NULL,
    dblink_from_node  IN    VARCHAR2 DEFAULT NULL,
```
as_source              IN  VARCHAR2 DEFAULT 'TRUE',
as_candidate_target    IN  VARCHAR2 DEFAULT 'FALSE');
node_id                OUT VARCHAR2);

DBMS_UMF.REGISTER_NODE(
    topology_name          IN  VARCHAR2,
    node_name              IN  VARCHAR2,
    dblink_to_node         IN  VARCHAR2 DEFAULT NULL,
    dblink_from_node       IN  VARCHAR2 DEFAULT NULL,
    as_source              IN  VARCHAR2 DEFAULT 'TRUE',
    as_candidate_target    IN  VARCHAR2 DEFAULT 'FALSE')
RETURN NUMBER;

Parameters

Table 182-12  REGISTER_NODE Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
<tr>
<td>node_name</td>
<td>Name of the node to register.</td>
</tr>
<tr>
<td>dblink_to_node</td>
<td>Name for the database link from the destination to the node.</td>
</tr>
<tr>
<td>dblink_from_node</td>
<td>Name for the database link from the node to the destination.</td>
</tr>
<tr>
<td>as_source</td>
<td>Set to TRUE, if the node is a source, else set to FALSE.</td>
</tr>
<tr>
<td>as_candidate_target</td>
<td>Set to TRUE, if the node is a candidate destination, else set to FALSE.</td>
</tr>
<tr>
<td>node_id</td>
<td>Node ID returned by the procedure.</td>
</tr>
</tbody>
</table>

Return Value

Returns the node ID of the registered node.

Security Model

Only the database users SYS and SYSSUMP can execute this procedure.

182.1.14 SWITCH_DESTINATION Procedure

This procedure makes the candidate destination as the new destination in the RMF topology. This procedure must be executed only on the candidate destination node.

Syntax

DBMS_UMF.SWITCH_DESTINATION(
    topology_name      IN VARCHAR2,
    force_switch       IN BOOLEAN DEFAULT TRUE);
### Parameters

**Table 182-13  SWITCH_DESTINATION Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
<tr>
<td>force_switch</td>
<td>If set to FALSE, the execution of this procedure fails in the following situations:</td>
</tr>
<tr>
<td></td>
<td>• Candidate destination is a read-only database, and hence it cannot become the new destination.</td>
</tr>
<tr>
<td></td>
<td>• Candidate destination does not have database links to one or more sources in the topology.</td>
</tr>
<tr>
<td></td>
<td>• Candidate destination is unable to get the latest AWR data from the old destination.</td>
</tr>
<tr>
<td></td>
<td>If set to TRUE, the execution of this procedure fails in the following situation:</td>
</tr>
<tr>
<td></td>
<td>• Candidate destination is a read-only database, and hence it cannot become the new destination.</td>
</tr>
</tbody>
</table>

**Security Model**

Only the database users SYS and SYSSUMF can execute this procedure.

### 182.1.15 UNCONFIGURE_NODE Procedure

This procedure removes the configuration details of the node on which this procedure is executed.

**Syntax**

```sql
DBMS_UMF.UNCONFIGURE_NODE;
```

**Security Model**

Only the database users SYS and SYSSUMF can execute this procedure.

### 182.1.16 UNREGISTER_NODE Procedure

This procedure removes a node for the RMF topology. This procedure must be executed only on the destination node.

**Syntax**

```sql
DBMS_UMF.UNREGISTER_NODE(
    topology_name    IN VARCHAR2,
    node_name        IN VARCHAR2);
```
Parameters

Table 182-14  UNREGISTER_NODE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topology_name</td>
<td>Name of the RMF topology.</td>
</tr>
<tr>
<td>node_name</td>
<td>Name of the node which needs to be removed from the topology.</td>
</tr>
</tbody>
</table>

Security Model

Only the database users SYS and SYSSMF can execute this procedure.
DBMS_UTILITY

The DBMS_UTILITY package provides various utility subprograms.

This chapter contains the following topics:

• DBMS_UTILITY Deprecated Subprograms
• Security Model
• Constants
• Exceptions
• Data Structures
• Summary of DBMS_UTILITY Subprograms

183.1 DBMS_UTILITY Deprecated Subprograms

These DBMS_UTILITY subprograms are deprecated in Oracle Database 12c release 12.2.

>Note:

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

• GET_PARAMETER_VALUE Function
  Query v$parameter directly to find the value of an init.ora parameter.

• ANALYZE_PART_OBJECT Procedure
  Use DBMS_STATS to gather statistics.

• GET_DEPENDENCY Procedure
  There is no replacement for this subprogram. However, you can directly query the dictionary views.

There are no replacements for these subprograms.

183.2 DBMS_UTILITY Security Model

DBMS_UTILITY runs with the privileges of the calling user for the NAME_RESOLVE procedure and the COMPARE_SCHEMA procedure. This is necessary so that the SQL works correctly.

The package does not run as SYS. The privileges are checked using DBMS_DDL.
Related Topics

- **NAME_RESOLVE Procedure**
  This procedure resolves the given name, including synonym translation and authorization checking as necessary.

- **COMPILE_SCHEMA Procedure**
  This procedure compiles all procedures, functions, packages, views and triggers in the specified schema.

### 183.3 DBMSUTILITY Constants

The DBMSUTILITY package defines one constant to use when specifying parameter values.

This constant is shown in the following table.

**Table 183-1  DBMSUTILITY Constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV_ERROR_ON_REST_RICTIONS</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>This constant is the only legal value for the p_option_flags parameter of the INVALIDATE subprogram</td>
</tr>
</tbody>
</table>

### 183.4 DBMSUTILITY Exceptions

This table lists the exceptions raised by DBMSUTILITY.

**Table 183-2  Exceptions Raised by DBMSUTILITY**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV_NOT_EXIST_OR_NO_PRIV</td>
<td>-24237</td>
<td>Raised by the INVALIDATE subprogram when the object_id argument is NULL or invalid, or when the caller does not have CREATE privileges on the object being invalidated</td>
</tr>
<tr>
<td>INV_MALFORMED_SETTINGS</td>
<td>-24238</td>
<td>Raised by the INVALIDATE subprogram if a compiler setting is specified more than once in the p_plsql_object_settings parameter</td>
</tr>
<tr>
<td>INV.Restricted/Object</td>
<td>-24239</td>
<td>Raised by the INVALIDATE subprogram when different combinations of conditions pertaining to the p_object_id parameter are contravened</td>
</tr>
</tbody>
</table>

### 183.5 DBMSUTILITY Data Structures

The DBMSUTILITY package defines a single RECORD type and TABLE types.

**Record Types**

- **INSTANCE_RECORD Record Type**
Table Types

- DBLINK_ARRAY TABLE Type
- INDEX_TABLE_TYPE Table Type
- INSTANCE_TABLE Table Type
- LNAME_ARRAY Table Type
- NAME_ARRAY Table Type
- NUMBER_ARRAY Table Type
- UNCL_ARRAY Table Type

183.5.1 DBMS_UTILITY INSTANCE_RECORD Record Type

This type describes a list of active instance number-name pairs.

Syntax

```sql
TYPE INSTANCE_RECORD IS RECORD (
  inst_number   NUMBER,
  inst_name     VARCHAR2(60));
```

Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst_number</td>
<td>Active instance number</td>
</tr>
<tr>
<td>inst_name</td>
<td>Instance name</td>
</tr>
</tbody>
</table>

183.5.2 DBMS_UTILITY DBLINK_ARRAY TABLE Type

This type stores a list of database links.

Syntax

```sql
TYPE DBLINK_ARRAY IS TABLE OF VARCHAR2(128) INDEX BY BINARY_INTEGER;
```

183.5.3 DBMS_UTILITY INDEX_TABLE_TYPE Table Type

This type describes the order in which generated objects are returned to a user.

Syntax

```sql
TYPE INDEX_TABLE_TYPE IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```
183.5.4 DBMS_UTILITY INSTANCE_TABLE Table Type

This type describes a table of INSTANCE_RECORD Record Type.

Syntax

TYPE INSTANCE_TABLE IS TABLE OF INSTANCE_RECORD INDEX BY BINARY_INTEGER;

Usage Notes

The starting index of INSTANCE_TABLE is 1; INSTANCE_TABLE is Dense.

Related Topics

- DBMS_UTILITY INSTANCE_RECORD Record Type
  This type describes a list of active instance number-name pairs.

183.5.5 DBMS_UTILITY LNAME_ARRAY Table Type

This type stores lists of LONG NAME including fully qualified attribute names.

Syntax

TYPE LNAME_ARRAY IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;

183.5.6 DBMS_UTILITY NAME_ARRAY Table Type

This type stores lists of NAME.

Syntax

TYPE NAME_ARRAY IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;

183.5.7 DBMS_UTILITY NUMBER_ARRAY Table Type

This type describes the order in which generated objects are returned to users.

Syntax

TYPE NUMBER_ARRAY IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;

183.5.8 DBMS_UTILITY UNCL_ARRAY Table Type

This type stores lists of "user"."name"."column"@link

Syntax

TYPE UNCL_ARRAY IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;

183.6 Summary of DBMS_UTILITY Subprograms

This table lists the DBMS_UTILITY subprograms and briefly describes them.
## DBMSUTILITY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE_INSTANCES Procedure</td>
<td>Returns the active instance</td>
</tr>
<tr>
<td>ANALYZE_DATABASE Procedure</td>
<td>Analyzes all the tables, clusters and indexes in a database</td>
</tr>
<tr>
<td>ANALYZE_PART_OBJECT Procedure</td>
<td>Analyzes the given tables and indexes</td>
</tr>
<tr>
<td></td>
<td>This procedure is deprecated from the DBMSUTILITY package with Oracle Database 12c release 12.2 and later. Use DBMS_STATS to gather statistics.</td>
</tr>
<tr>
<td>ANALYZE_SCHEMA Procedure</td>
<td>Analyzes all the tables, clusters and indexes in a schema</td>
</tr>
<tr>
<td>CANONICALIZE Procedure</td>
<td>Canonicalizes a given string</td>
</tr>
<tr>
<td>COMMA_TO_TABLE Procedures</td>
<td>Converts a comma-delimited list of names into a PL/SQL table of names</td>
</tr>
<tr>
<td>COMPILE_SCHEMA Procedure</td>
<td>Compiles all procedures, functions, packages, views and triggers in the specified schema</td>
</tr>
<tr>
<td>CREATE_ALTER_TYPE_ERROR_TABLE Procedure</td>
<td>Creates an error table to be used in the EXCEPTION clause of the ALTER TYPE statement</td>
</tr>
<tr>
<td>CURRENT_INSTANCE Function</td>
<td>Returns the current connected instance number</td>
</tr>
<tr>
<td>DATA_BLOCK_ADDRESS_BLOCK Function</td>
<td>Gets the block number part of a data block address</td>
</tr>
<tr>
<td>DATA_BLOCK_ADDRESS_FILE Function</td>
<td>Gets the file number part of a data block address</td>
</tr>
<tr>
<td>DB_VERSION Procedure</td>
<td>Returns version information for the database</td>
</tr>
<tr>
<td>EXECDDLSTATEMENT Procedure</td>
<td>Executes the DDL statement in parse_string</td>
</tr>
<tr>
<td>EXPAND_SQL_TEXT Procedure</td>
<td>Recursively replaces any view references in the input SQL query with the corresponding view subquery</td>
</tr>
<tr>
<td>FORMAT_CALL_STACK Function</td>
<td>Formats the current call stack</td>
</tr>
<tr>
<td>FORMAT_ERROR_BACKTRACE Function</td>
<td>Formats the backtrace from the point of the current error to the exception handler where the error has been caught</td>
</tr>
<tr>
<td>FORMAT_ERROR_STACK Function</td>
<td>Formats the current error stack</td>
</tr>
<tr>
<td>GET_CPU_TIME Function</td>
<td>Returns the current CPU time in 100th's of a second</td>
</tr>
<tr>
<td>GET_DEPENDENCY Procedure</td>
<td>Shows the dependencies on the object passed in.</td>
</tr>
<tr>
<td>GET_ENDIANNESS Function</td>
<td>Gets the endianness of the database platform</td>
</tr>
<tr>
<td>GET_HASH_VALUE Function</td>
<td>Computes a hash value for the given string</td>
</tr>
<tr>
<td>GET_PARAMETER_VALUE Function</td>
<td>Gets the value of specified init.ora parameter.</td>
</tr>
<tr>
<td>GET_SQL_HASH Function</td>
<td>Computes a hash value for the given string using MD5 algorithm</td>
</tr>
<tr>
<td>GET_TIME Function</td>
<td>Returns the current time in 100th's of a second</td>
</tr>
<tr>
<td>GET_TZ_TRANSITIONS Procedure</td>
<td>Returns time zone transitions by regionid from the timezone.dat file</td>
</tr>
</tbody>
</table>
Table 183-4  (Cont.) DBMS_UTILITY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALIDATE Procedure</td>
<td>Invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings</td>
</tr>
<tr>
<td>IS_BIT_SET Function</td>
<td>Checks the bit setting for the given bit in the given RAW value</td>
</tr>
<tr>
<td>IS_CLUSTER_DATABASE Function</td>
<td>Determines if the database is running in cluster database mode</td>
</tr>
<tr>
<td>MAKE_DATA_BLOCK_ADDRESS Function</td>
<td>Creates a data block address given a file number and a block number</td>
</tr>
<tr>
<td>NAME_RESOLVE Procedure</td>
<td>Resolves the given name</td>
</tr>
<tr>
<td>NAME_TOKENIZE Procedure</td>
<td>Calls the parser to parse the given name</td>
</tr>
<tr>
<td>OLD_CURRENT_SCHEMA Function</td>
<td>Returns the session value from SYS_CONTEXT('USERENV', 'CURRENT_SCHEMA')</td>
</tr>
<tr>
<td>OLD_CURRENT_USER Function</td>
<td>Returns the session value from SYS_CONTEXT('USERENV', 'CURRENT_USER')</td>
</tr>
<tr>
<td>PORT_STRING Function</td>
<td>Returns a string that uniquely identifies the version of Oracle and the operating system</td>
</tr>
<tr>
<td>SQLID_TO_SQLHASH Function</td>
<td>Converts a SQL ID into a hash value</td>
</tr>
<tr>
<td>TABLE_TO_COMMA Procedures</td>
<td>Converts a PL/SQL table of names into a comma-delimited list of names</td>
</tr>
<tr>
<td>VALIDATE Procedure</td>
<td>Makes invalid database objects valid</td>
</tr>
<tr>
<td>WAIT_ON_PENDING_DML Function</td>
<td>Waits until all transactions (other than the caller’s own) that have locks on the listed tables and began prior to the specified SCN have either committed or been rolled back</td>
</tr>
</tbody>
</table>

183.6.1 ACTIVE_INSTANCES Procedure

This procedure returns the active instance.

Syntax

```sql
DBMS_UTILITY.ACTIVE_INSTANCES (
    instance_table OUT INSTANCE_TABLE,
    instance_count OUT NUMBER);
```

Parameters

Table 183-5  ACTIVE_INSTANCES Procedure Parameters

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance_table</td>
<td>Contains a list of the active instance numbers and names. When no instance is up, the list is empty.</td>
</tr>
<tr>
<td>instance_count</td>
<td>Number of active instances</td>
</tr>
</tbody>
</table>
183.6.2 ANALYZE_DATABASE Procedure

This procedure analyzes all the tables, clusters and indexes in a database.

Syntax

```sql
DBMS_UTILITY.ANALYZE_DATABASE (
    method             IN  VARCHAR2,
    estimate_rows      IN  NUMBER DEFAULT NULL,
    estimate_percent   IN  NUMBER DEFAULT NULL,
    method_opt         IN  VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE then either estimate_rows or estimate_percent must be nonzero.</td>
</tr>
<tr>
<td>estimate_rows</td>
<td>Number of rows to estimate</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate. If estimate_rows is specified ignore this parameter.</td>
</tr>
<tr>
<td>method_opt</td>
<td>Method options of the following format:</td>
</tr>
<tr>
<td></td>
<td>[ FOR TABLE ]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL [INDEXED] COLUMNS] [ SIZE n]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL INDEXES ]</td>
</tr>
</tbody>
</table>

Exceptions

ORA-20000: Insufficient privileges for some object in this database

183.6.3 ANALYZE_PART_OBJECT Procedure

This procedure is equivalent to SQL: "ANALYZE TABLE|INDEX [<schema>.]<object_name> PARTITION <pname> [<command_type>] [<command_opt>] [<sample_clause>]

**Note:**

This subprogram has been deprecated and replaced by improved technology. It is maintained only for purposes of backward compatibility. As an alternative, you can use DBMS_STATS to gather statistics.

Syntax

```sql
DBMS_UTILITY.ANALYZE_PART_OBJECT (
    schema        IN VARCHAR2 DEFAULT NULL,
    object_name   IN VARCHAR2 DEFAULT NULL,
    object_type   IN CHAR DEFAULT 'T',
    command_type  IN CHAR DEFAULT 'E',
    command_opt   IN VARCHAR2 DEFAULT NULL,
    sample_clause IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 183-7  ANALYZE_PART_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Schema of the object_name</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of object to be analyzed, must be partitioned</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of object, must be T (table) or I (index)</td>
</tr>
<tr>
<td>command_type</td>
<td>Must be V (validate structure)</td>
</tr>
<tr>
<td>command_opt</td>
<td>Other options for the command type. For C, E it can be FOR table, FOR all LOCAL indexes, FOR all columns or combination of some of the 'for' options of analyze statistics (table). For V, it can be CASCADE when object_type is T.</td>
</tr>
<tr>
<td>sample_clause</td>
<td>Sample clause to use when command_type is 'E'</td>
</tr>
</tbody>
</table>

Usage Notes

For each partition of the object, run in parallel using job queues.

183.6.4 ANALYZE_SCHEMA Procedure

This procedure analyzes all the tables, clusters and indexes in a schema.

Syntax

```sql
DBMS_UTILITY.ANALYZE_SCHEMA (  
    schema IN VARCHAR2,  
    method IN VARCHAR2,  
    estimate_rows IN NUMBER DEFAULT NULL,  
    estimate_percent IN NUMBER DEFAULT NULL,  
    method_opt IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 183-8  ANALYZE_SCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>method</td>
<td>One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE then either estimate_rows or estimate_percent must be nonzero.</td>
</tr>
<tr>
<td>estimate_rows</td>
<td>Number of rows to estimate</td>
</tr>
<tr>
<td>estimate_percent</td>
<td>Percentage of rows to estimate. If estimate_rows is specified ignore this parameter.</td>
</tr>
</tbody>
</table>
**Table 183-8 (Cont.) ANALYZE_SCHEMA Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method_opt</td>
<td>Method options of the following format:</td>
</tr>
<tr>
<td></td>
<td>[ FOR TABLE ]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL [INDEXED] COLUMNS][SIZE n]</td>
</tr>
<tr>
<td></td>
<td>[ FOR ALL INDEXES ]</td>
</tr>
</tbody>
</table>

**Exceptions**

ORA-20000: Insufficient privileges for some object in this schema

### 183.6.5 CANONICALIZE Procedure

This procedure canonicalizes the given string. The procedure handles a single reserved or key word (such as 'table'), and strips off white spaces for a single identifier so that 'table' becomes TABLE.

**Syntax**

```sql
DBMS_UTILITY.CANONICALIZE(
    name           IN    VARCHAR2,
    canon_name     OUT   VARCHAR2,
    canon_len      IN    BINARY_INTEGER);
```

**Parameters**

**Table 183-9 CANONICALIZE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String to be canonicalized</td>
</tr>
<tr>
<td>canon_name</td>
<td>Canonicalized string</td>
</tr>
<tr>
<td>canon_len</td>
<td>Length of the string (in bytes) to canonicalize</td>
</tr>
</tbody>
</table>

**Return Values**

Returns the first `canon_len` bytes in `canon_name`.

**Usage Notes**

- If `name` is `NULL`, `canon_name` becomes `NULL`.
- If `name` is not a dotted name, and if `name` begins and ends with a double quote, remove both quotes. Alternatively, convert to upper case with NLS_UPPER. Note that this case does not include a name with special characters, such as a space, but is not doubly quoted.
- If `name` is a dotted name (such as a."b".c), for each component in the dotted name in the case in which the component begins and ends with a double quote, no transformation will be performed on this component. Alternatively, convert to upper case with NLS_UPPER and apply begin and end double quotes to the capitalized form of this component. In such
a case, each canonicalized component will be concatenated together in the input position, separated by ".".

- Any other character after a[b]* will be ignored.
- The procedure does not handle cases like 'A B.'

Examples

- `a` becomes `A`
- "a" becomes `a`
- "a".b becomes "a"."B"
- "a".b,c.f becomes "a"."B" with",c.f" ignored.

183.6.6 COMMA_TO_TABLE Procedures

These procedures convert a comma-delimited list of names into a PL/SQL table of names. The second version supports fully-qualified attribute names.

Syntax

```sql
DBMS_UTILITY.COMMA_TO_TABLE (  
    list   IN  VARCHAR2,  
    tablen OUT BINARY_INTEGER,  
    tab    OUT uncl_array);

DBMS_UTILITY.COMMA_TO_TABLE (  
    list   IN  VARCHAR2,  
    tablen OUT BINARY_INTEGER,  
    tab    OUT lname_array);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Comma separated list of list of 'names', where a name should have the following format for the first overloading: a [. b [. c ]]@d] and the following format for the second overloading: a [. b]* where a, b, c, d are simple identifiers (quoted or unquoted).</td>
</tr>
<tr>
<td>tablen</td>
<td>Number of tables in the PL/SQL table</td>
</tr>
<tr>
<td>tab</td>
<td>PL/SQL table which contains list of names</td>
</tr>
</tbody>
</table>

Return Values

A PL/SQL table is returned, with values 1..n and n+1 is null.

Usage Notes

- The list must be a non-empty comma-delimited list: Anything other than a comma-delimited list is rejected. Commas inside double quotes do not count.
- Entries in the comma-delimited list cannot include multibyte characters.
• The values in tab are copied from the original list, with no transformations.
• The procedure fails if the string between separators is longer than 30 bytes.

183.6.7 COMPILE_SCHEMA Procedure

This procedure compiles all procedures, functions, packages, views and triggers in the specified schema.

Syntax

```sql
DBMS_UTILITY.COMPILE_SCHEMA (
    schema    IN VARCHAR2,
    compile_all IN BOOLEAN DEFAULT TRUE,
    reuse_settings IN BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>compile_all</td>
<td>If TRUE, will compile everything within the schema regardless of whether it is VALID</td>
</tr>
<tr>
<td></td>
<td>If FALSE, will compile only INVALID objects</td>
</tr>
<tr>
<td>reuse_settings</td>
<td>Indicates whether the session settings in the objects should be reused, or whether the current session settings should be adopted instead</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20000</td>
<td>Insufficient privileges for some object in this schema</td>
</tr>
<tr>
<td>ORA-20001</td>
<td>Cannot recompile SYS objects</td>
</tr>
<tr>
<td>ORA-20002</td>
<td>Maximum iterations exceeded. Some objects may not have been recompiled.</td>
</tr>
</tbody>
</table>

Usage Notes

• Note that this subprogram is a wrapper for the RECOMP_SERIAL Procedure included with the UTL_RECOMP package.
• After calling this procedure, you should select from view ALL_OBJECTS for items with status of INVALID to see if all objects were successfully compiled.
• To see the errors associated with INVALID objects, you may use the Enterprise Manager command:

  ```sql
  SHOW ERRORS <type> <schema>.<name>
  ```
183.6.8 CREATE_ALTER_TYPE_ERROR_TABLE Procedure

This procedure creates an error table to be used in the EXCEPTION clause of the ALTER TYPE statement.

**Syntax**

```sql
DBMS_UTILITY.CREATE_ALTER_TYPE_ERROR_TABLE(
    schema_name     IN     VARCHAR2,
    table_name      IN     VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Name of the schema</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table created</td>
</tr>
</tbody>
</table>

**Exceptions**

An error is returned if the table already exists.

183.6.9 CURRENT_INSTANCE Function

This function returns the current connected instance number. It returns NULL when connected instance is down.

**Syntax**

```sql
DBMS_UTILITY.CURRENT_INSTANCE
    RETURN NUMBER;
```

183.6.10 DATA_BLOCK_ADDRESS_BLOCK Function

This function gets the block number part of a data block address.

**Syntax**

```sql
DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK (
    dba NUMBER)
    RETURN NUMBER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba</td>
<td>Data block address</td>
</tr>
</tbody>
</table>
Pragmas
pragma restrict_references(data_block_address_block, WNDS, RNDS, WNPS, RNPS);

Return Values
Block offset of the block.

Usage Notes
This function should not be used with datablocks which belong to bigfile tablespaces.

183.6.11 DATA_BLOCK_ADDRESS_FILE Function

This function gets the file number part of a data block address.

Syntax
```sql
DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE (dba NUMBER) RETURN NUMBER;
```

Parameters

Table 183-15  DATA_BLOCK_ADDRESS_FILE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba</td>
<td>Data block address</td>
</tr>
</tbody>
</table>

Pragmas
pragma restrict_references (data_block_address_file, WNDS, RNDS, WNPS, RNPS);

Return Values
File that contains the block.

Usage Notes
This function should not be used with datablocks which belong to bigfile tablespaces.

183.6.12 DB_VERSION Procedure

This procedure returns version information for the database.

Syntax
```sql
DBMS_UTILITY.DB_VERSION (version OUT VARCHAR2, compatibility OUT VARCHAR2);
```
Parameters

Table 183-16  DB_VERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>A string which represents the internal software version of the database (for example, 7.1.0.0.0). The length of this string is variable and is determined by the database version.</td>
</tr>
<tr>
<td>compatibility</td>
<td>The compatibility setting of the database determined by the &quot;compatible&quot; init.ora parameter. If the parameter is not specified in the init.ora file, then NULL is returned.</td>
</tr>
</tbody>
</table>

183.6.13 EXEC_DDL_STATEMENT Procedure

This procedure executes the DDL statement in parse_string.

Syntax

```
DBMS_UTILITY.EXEC_DDL_STATEMENT (
    parse_string IN VARCHAR2);
```

Parameters

Table 183-17  EXEC_DDL_STATEMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse_string</td>
<td>DDL statement to be executed</td>
</tr>
</tbody>
</table>

183.6.14 EXPAND_SQL_TEXT Procedure

This procedure recursively replaces any view references in the input SQL query with the corresponding view subquery.

Syntax

```
DBMS_UTILITY.EXPAND_SQL_TEXT (
    input_sql_text IN CLOB,
    output_sql_text OUT NOCOPY CLOB);
```

Parameters

Table 183-18  EXPAND_SQL_TEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_sql_text</td>
<td>Input SQL query text</td>
</tr>
<tr>
<td>output_sql_text</td>
<td>View-expanded query text</td>
</tr>
</tbody>
</table>
Exceptions

Table 183-19  EXPAND_SQL_TEXT Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-00942</td>
<td>Current user does not have select privileges on all the views and tables recursively referenced in the input_sql_text</td>
</tr>
<tr>
<td>ORA-24251</td>
<td>input_sql_text is not a SELECT statement</td>
</tr>
<tr>
<td>ORA-00900</td>
<td>Input is not valid</td>
</tr>
<tr>
<td>ORA-29477</td>
<td>Input LOB size exceeds maximum size of 4GB - 1</td>
</tr>
</tbody>
</table>

Usage Notes

The expanded and merged SQL statement text is copied to output_sql_text on successful completion. The resulting query text only contains references to underlying tables and is semantically equivalent with some caveats:

- If there are invoker rights functions called from any of the views, they may be called as a different user in the resulting query text if the view owner is different from the user who will eventually compile/run the expanded SQL text.
- The VPD policy expands differently if there is a function supplied to generate the dynamic WHERE clause. This function would return differently, for example, if the userid caused the expansion to be different.
- If there are references to remote objects, results are undetermined.

183.6.15 FORMAT_CALL_STACK Function

This function formats the current call stack. This can be used on any stored procedure or trigger to access the call stack. This can be useful for debugging.

Syntax

```sql
DBMS_UTILITY.FORMAT_CALL_STACK
RETURN VARCHAR2;
```

Pragmas

```sql
pragma restrict_references(format_call_stack,WNDS);
```

Return Values

This returns the call stack, up to 2000 bytes.

183.6.16 FORMAT_ERROR_BACKTRACE Function

This function displays the call stack at the point where an exception was raised, even if the subprogram is called from an exception handler in an outer scope.

The output is similar to the output of the SQLERRM function, but not subject to the same size limitation.
Syntax

DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
   RETURN VARCHAR2;

Return Values

The backtrace string. A NULL string is returned if no error is currently being handled.

Examples

CREATE OR REPLACE PROCEDURE Log_Errors ( i_buff in varchar2 ) IS
   g_start_pos integer := 1;
   g_end_pos  integer;

   FUNCTION Output_One_Line RETURN BOOLEAN IS
      BEGIN
         g_end_pos := Instr ( i_buff, Chr(10), g_start_pos );

         CASE g_end_pos > 0
            WHEN true THEN
               DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
                  g_end_pos-g_start_pos ) );
               g_start_pos := g_end_pos+1;
               RETURN TRUE;
            WHEN FALSE THEN
               DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
                  (Length(i_buff)-g_start_pos)+1 ) );
               RETURN FALSE;
            END CASE;
         END Output_One_Line;

      BEGIN
         WHILE Output_One_Line() LOOP NULL;
      END LOOP;
   END Log_Errors;
/

CREATE OR REPLACE PROCEDURE P0 IS
   e_01476 EXCEPTION; pragma exception_init ( e_01476, -1476 );
   BEGIN
      RAISE e_01476;
   END P0;
/

CREATE OR REPLACE PROCEDURE P1 IS
   BEGIN
      P0();
   END P1;
/

CREATE OR REPLACE PROCEDURE P2 IS
   BEGIN
      /

CREATE OR REPLACE PROCEDURE P3 IS
BEGIN
P2();
END P3;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P4 IS
BEGIN P3(); END P4;
/
CREATE OR REPLACE PROCEDURE P5 IS
BEGIN P4(); END P5;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE Top_Naive IS
BEGIN
P5();
END Top_Naive;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE Top_With_Logging IS
-- NOTE: SqlErrm in principle gives the same info as Format_Error_Stack.
-- But SqlErrm is subject to some length limits,
-- while Format_Error_Stack is not.
BEGIN
P5();
EXCEPTION
WHEN OTHERS THEN
  Log_Errors ( 'Error_Stack...' || Chr(10) ||
               DBMS_UTILITY.FORMAT_ERROR_STACK() );
  Log_Errors ( 'Error_Backtrace...' || Chr(10) ||
               DBMS_UTILITY.FORMAT_ERROR_BACKTRACE() );
  DBMS_OUTPUT.PUT_LINE ( '----------' );
END Top_With_Logging;
/
SHOW ERRORS

--------------------------------------------------------------------------------
Set ServerOutput On
call Top_Naive() /*
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-006512: at "U.P0", line 4
ORA-006512: at "U.P1", line 3
ORA-006512: at "U.P2", line 3
ORA-006512: at "U.P3", line 3
ORA-006512: at "U.P4", line 2
ORA-006512: at "U.P5", line 2
ORA-006512: at "U.TOP_NAIVE", line 3 */
;
Set ServerOutput On

call Top_With_Logging()

/*
Error Stack...
ORA-01476: divisor is equal to zero
Error Backtrace...
ORA-06512: at "U.P0", line 4
ORA-06512: at "U.P1", line 3
ORA-06512: at "U.P2", line 3
ORA-06512: at "U.P3", line 3
ORA-06512: at "U.P4", line 2
ORA-06512: at "U.P5", line 2
ORA-06512: at "U.TOP_WITH_LOGGING", line 6
----------
*/

/*
ORA-06512:
Cause:
    Backtrace message as the stack is
    unwound by unhandled exceptions.
Action:
    Fix the problem causing the exception
    or write an exception handler for this condition.
    Or you may need to contact your application administrator
    or database administrator.
*/

183.6.17 FORMAT_ERROR_STACK Function

This function formats the current error stack. This can be used in exception handlers to look at the full error stack.

Syntax

```
DBMS_UTILITY.FORMAT_ERROR_STACK
   RETURN VARCHAR2;
```

Return Values

This returns the error stack, up to 2000 bytes.

183.6.18 GET_CPU_TIME Function

This function returns a measure of current CPU processing time in hundredths of a second. The difference between the times returned from two calls measures the CPU processing time (not the total elapsed time) between those two points.

Syntax

```
DBMS_UTILITY.GET_CPU_TIME
   RETURN NUMBER;
```

Return Values

Time is the number of 100th's of a second from some arbitrary epoch.
Usage Notes

The amount of work performed is calculated by measuring the difference between a start point and end point for a particular operation.

183.6.19 GET_DEPENDENCY Procedure

This deprecated procedure shows the dependencies on the object passed in.

Note:

This subprogram has been deprecated and replaced in Oracle Database 12c release 12.2 and later. Oracle recommends that you do not use deprecated subprograms. It is maintained only for purposes of backward compatibility.

Syntax

```sql
DBMS_UTILITY.GETDEPENDENCY
    type IN VARCHAR2,
    schema IN VARCHAR2,
    name IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of the object, for example if the object is a table give the type as &quot;TABLE&quot;</td>
</tr>
<tr>
<td>schema</td>
<td>Schema name of the object</td>
</tr>
<tr>
<td>name</td>
<td>Name of the object</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure uses the DBMS_OUTPUT package to display results, and so you must declare SET SERVEROUTPUT ON if you wish to view dependencies. Alternatively, any application that checks the DBMS_OUTPUT output buffers can invoke this subprogram and then retrieve the output by means of DBMS_OUTPUT subprograms such as GET_LINES.

183.6.20 GET_ENDIANNESS Function

This function gets the endianness of the database platform.

Syntax

```sql
DBMS_UTILITY.GET_ENDIANNESS
    RETURN NUMBER;
```
Return Values

A NUMBER value indicating the endianness of the database platform: 1 for big-endian or 2 for little-endian.

183.6.21 GET_HASH_VALUE Function

This function computes a hash value for the given string.

Syntax

DBMS_UTILITY.GET_HASH_VALUE ( 
    name VARCHAR2, 
    base NUMBER, 
    hash_size NUMBER) 
RETURN NUMBER;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String to be hashed.</td>
</tr>
<tr>
<td>base</td>
<td>Base value for the returned hash value at which to start</td>
</tr>
<tr>
<td>hash_size</td>
<td>Desired size of the hash table</td>
</tr>
</tbody>
</table>

Pragmas

pragma restrict_references(get_hash_value, WNDS, RNDS, WNPS, RNPS);

Return Values

A hash value based on the input string. For example, to get a hash value on a string where the hash value should be between 1000 and 3047, use 1000 as the base value and 2048 as the hash_size value. Using a power of 2 for the hash_size parameter works best.

183.6.22 GET_PARAMETER_VALUE Function

This deprecated function gets the value of specified init.ora parameter.

Note:

This subprogram has been deprecated and replaced by improved technology. It is maintained only for purposes of backward compatibility. As an alternative, you can query v$parameter directly.

Syntax

DBMS_UTILITY.GET_PARAMETER_VALUE ( 
    parnam IN VARCHAR2,
intval    IN OUT    BINARY_INTEGER,
strval    IN OUT    VARCHAR2,
listno    IN        BINARY_INTEGER DEFAULT 1)
RETURN BINARY_INTEGER;

Parameters

Table 183-22   GET_PARAMETER_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parnam</td>
<td>Parameter name</td>
</tr>
<tr>
<td>intval</td>
<td>Value of an integer parameter or the value length of a string parameter</td>
</tr>
<tr>
<td>strval</td>
<td>Value of a string parameter</td>
</tr>
<tr>
<td>listno</td>
<td>List item number. If retrieving parameter values for a parameter that can be specified multiple times to accumulate values, use this parameter to get each individual parameter.</td>
</tr>
</tbody>
</table>

Return Values

Parameter type:

- 0 if parameter is an INTEGER/BOOLEAN parameter
- 1 if parameter is a string/file parameter

Usage Notes

- To execute the this function, you must have the SELECT privilege on the V$PARAMETER dynamic view.

Examples

DECLARE
    parnam VARCHAR2(256);
    intval BINARY_INTEGER;
    strval VARCHAR2(256);
    partyp BINARY_INTEGER;
BEGIN
    partyp := dbms_utility.get_parameter_value('max_dump_file_size', intval, strval);
    dbms_output.put('parameter value is: ');
    IF partyp = 1 THEN
        dbms_output.put_line(strval);
    ELSE
        dbms_output.put_line(intval);
    END IF;
    IF partyp = 1 THEN
        dbms_output.put('parameter value length is: ');
        dbms_output.put_line(intval);
    END IF;
    dbms_output.put('parameter type is: ');
    IF partyp = 1 THEN
        dbms_output.put_line('string');
    ELSE
        dbms_output.put_line('integer');
    END IF;
END;
183.6.23 GET_SQL_HASH Function

This function computes a hash value for the given string using MD5 algorithm.

**Syntax**

```
Dbms_utility.get_sql_hash (  
      name            IN    VARCHAR2,  
      hash            OUT   RAW,  
      pre10ihash      OUT   NUMBER)  
RETURN NUMBER;
```

**Pragmas**

Pragma Restrict_references(Get_sql_hash, Wnds, Rnds, Wnps, Rnps);  

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String to be hashed</td>
</tr>
<tr>
<td>hash</td>
<td>Stores all 16 bytes of returned hash value</td>
</tr>
<tr>
<td>pre10ihash</td>
<td>Stores the pre 10i database version hash value</td>
</tr>
</tbody>
</table>

**Return Values**

A hash value (last 4 bytes) based on the input string. The MD5 hash algorithm computes a 16 byte hash value, but we only return the last 4 bytes so that we can return an actual number. Use the hash parameter to get all 16 bytes and pre10i hash parameter to store the pre 10i hash value of 4 bytes.

183.6.24 GET_TIME Function

This function determines the current time in hundredths of a second. This subprogram is primarily used for determining elapsed time. The subprogram is called twice – at the beginning and end of some process – and then the first (earlier) number is subtracted from the second (later) number to determine the time elapsed.

**Syntax**

```
DBMS_UTILITY.GET_TIME  
  RETURN NUMBER;
```

**Return Values**

Time is the number of hundredths of a second from the point in time at which the subprogram is invoked.

**Usage Notes**

Numbers are returned in the range -2147483648 to 2147483647 depending on platform and machine, and your application must take the sign of the number into account in determining the interval. For instance, in the case of two negative numbers,
application logic must allow that the first (earlier) number will be larger than the second (later) number which is closer to zero. By the same token, your application should also allow that the first (earlier) number be negative and the second (later) number be positive.

183.6.25 GET_TZ_TRANSITIONS Procedure

This procedure returns time zone transitions by regionid from the timezone.dat file.

Syntax

```
DBMS_UTILITY.GET_TZ_TRANSITIONS
   regionid  IN     NUMBER,
   transitions OUT  MAXRAW);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionid</td>
<td>Number corresponding to the region</td>
</tr>
<tr>
<td>transitions</td>
<td>Raw bytes from the timezone.dat file</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-6502: PL/SQL: NUMERIC OR VALUE ERROR</td>
<td>For an invalid regionid</td>
</tr>
</tbody>
</table>

183.6.26 INVALIDATE Procedure

This procedure invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings. It also invalidates any objects that (directly or indirectly) depend on the object being invalidated.

Syntax

```
DBMS_UTILITY.INVALIDATE (  
   p_object_id              NUMBER,
   p_plsql_object_settings  VARCHAR2 DEFAULT NULL,
   p_option_flags           PLS_INTEGER DEFAULT 0);
```
Parameters

Table 183-26  INVALIDATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_object_id</td>
<td>ID number of object to be invalidated. This is the same as the value of the OBJECT_ID column from ALL_OBJECTS. If the object_id argument is NULL or invalid then the exception inv_not_exist_or_no_priv is raised. The caller of this procedure must have create privileges on the object being invalidated else the inv_not_exist_or_no_priv exception is raised.</td>
</tr>
<tr>
<td>p_plsql_object_settings</td>
<td>Optional parameter that ignored if the object specified by p_object_id is not a PL/SQL object. If no value is specified for this parameter then the PL/SQL compiler settings are left unchanged, that is, equivalent to REUSE SETTINGS. If a value is provided, it must specify the values of the PL/SQL compiler settings separated by one or more spaces. Each setting can be specified only once else inv_malformed_settings exception will be raised. The setting values are changed only for the object specified by p_object_id and do not affect dependent objects that may be invalidated. The setting names and values are case insensitive. If a setting is omitted and REUSE SETTINGS is specified, then if a value was specified for the compiler setting in an earlier compilation of this library unit, Oracle Database uses that earlier value. If a setting is omitted and REUSE SETTINGS was not specified or no value has been specified for the parameter in an earlier compilation, then the database will obtain the value for that setting from the session environment.</td>
</tr>
<tr>
<td>p_option_flags</td>
<td>Optional parameter defaults to zero (no flags). Option flags supported by invalidate.</td>
</tr>
<tr>
<td></td>
<td>· inv_error_on_restrictions (see Constants): The subprogram imposes various restrictions on the objects that can be invalidated. For example, the object specified by p_object_id cannot be a table. By default, invalidate quietly returns on these conditions (and does not raise an exception). If the caller sets this flag, the exception inv_restricted_object is raised.</td>
</tr>
</tbody>
</table>

Exceptions

Table 183-27  INVALIDATE Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV_NOT_EXIST_OR_NO_PRIV</td>
<td>Raised when the object_id argument is NULL or invalid, or when the caller does not have CREATE privileges on the object being invalidated.</td>
</tr>
<tr>
<td>INV_MALFORMED_SETTINGS</td>
<td>Raised if a compiler setting is specified more than once in the p_plsql_object_settings parameter.</td>
</tr>
<tr>
<td>INV_RESTRICTED_OBJECT</td>
<td>Raised when different combinations of conditions pertaining to the p_object_id parameter are contravened.</td>
</tr>
</tbody>
</table>
Usage Notes

The object type (object_type column from ALL_OBJECTS) of the object specified by p_object_id must be a PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, TYPE, TYPE BODY, LIBRARY, VIEW, OPERATOR, SYNONYM, or JAVA CLASS. If the object is not one of these types and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

If the object specified by p_object_id is the package specification of STANDARD, DBMS_STANDARD, or specification or body of DBMS_UTILITY and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

If the object specified by p_object_id is an object type specification and there exist tables which depend on the type and the flag inv_error_on_restrictions is specified in p_option_flags then the exception inv_restricted_object is raised, else no action is taken.

Examples

Example 1

DBMS_UTILITY.INVALIDATE (1232, 'PLSQL_OPTIMIZE_LEVEL = 2 REUSE SETTINGS');

Assume that the object_id 1232 refers to the procedure remove_emp in the HR schema. Then the above call will mark the remove_emp procedure invalid and change its PLSQL_OPTIMIZE_LEVEL compiler setting to 2. The values of other compiler settings will remain unchanged since REUSE SETTINGS is specified.

Objects that depend on hr.remove_emp will also get marked invalid. Their compiler parameters will not be changed.

Example 2

DBMS_UTILITY.INVALIDATE (40775, 'plsql_code_type = native');

Assume that the object_id 40775 refers to the type body leaf_category_typ in the OE schema. Then the above call will mark the type body invalid and change its PLSQL_CODE_TYPE compiler setting to NATIVE. The values of other compiler settings will be picked up from the current session environment since REUSE SETTINGS has not been specified.

Since no objects can depend on bodies, there are no cascaded invalidations.

Example 3

DBMS_UTILITY.INVALIDATE (40796);

Assume that the object_id 40796 refers to the view oc_orders in the OE schema. Then the above call will mark the oc_orders view invalid.

Objects that depend on oe.oc_orders will also get marked invalid.
183.6.27 IS_BIT_SET Function

This function checks the bit setting for the given bit in the given RAW value.

**Syntax**

```sql
DBMS_UTILITY.IS_BIT_SET (r IN RAW, n IN NUMBER)
RETURN NUMBER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW source</td>
</tr>
<tr>
<td>n</td>
<td>Bit in r to check</td>
</tr>
</tbody>
</table>

**Return Values**

This function returns 1 if bit n in raw r is set, zero otherwise. Bits are numbered high to low with the lowest bit being bit number 1.

183.6.28 IS_CLUSTER_DATABASE Function

This function finds out if this database is running in cluster database mode.

**Syntax**

```sql
DBMS_UTILITY.IS_CLUSTER_DATABASE
RETURN BOOLEAN;
```

**Return Values**

This function returns TRUE if this instance was started in cluster database mode; FALSE otherwise.

183.6.29 MAKE_DATA_BLOCK_ADDRESS Function

This function creates a data block address given a file number and a block number.

A data block address is the internal structure used to identify a block in the database. This function is useful when accessing certain fixed tables that contain data block addresses.

**Syntax**

```sql
DBMS_UTILITY.MAKE_DATA_BLOCK_ADDRESS (file NUMBER,
    block NUMBER)
RETURN NUMBER;
```
Parameters

Table 183-29  MAKE_DATA_BLOCK_ADDRESS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File that contains the block</td>
</tr>
<tr>
<td>block</td>
<td>Offset of the block within the file in terms of block increments</td>
</tr>
</tbody>
</table>

Pragmas

pragma restrict_references (make_data_block_address, WNDS, RNDS, WNPS, RNPS);

Return Values

Data block address.

183.6.30 NAME_RESOLVE Procedure

This procedure resolves the given name, including synonym translation and authorization checking as necessary.

Syntax

DBMS_UTILITY.NAME_RESOLVE (  
    name          IN  VARCHAR2,  
    context       IN  NUMBER,  
    schema        OUT VARCHAR2,  
    part1         OUT VARCHAR2,  
    part2         OUT VARCHAR2,  
    dblink        OUT VARCHAR2,  
    part1_type    OUT NUMBER,  
    object_number OUT NUMBER);  

Parameters

Table 183-30  NAME_RESOLVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the object. This can be of the form [[a].b].c[@d], where a, b, c are SQL identifier and d is a dblink. No syntax checking is performed on the dblink. If a dblink is specified, or if the name resolves to something with a dblink, then object is not resolved, but the schema, part1, part2 and dblink OUT parameters are filled in. a, b and c may be delimited identifiers, and may contain Globalization Support (NLS) characters (single and multibyte).</td>
</tr>
</tbody>
</table>
Table 183-30  (Cont.) NAME_RESOLVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Must be an integer between 0 and 9.</td>
</tr>
<tr>
<td></td>
<td>• 0 - table</td>
</tr>
<tr>
<td></td>
<td>• 1 - PL/SQL (for 2 part names)</td>
</tr>
<tr>
<td></td>
<td>• 2 - sequences</td>
</tr>
<tr>
<td></td>
<td>• 3 - trigger</td>
</tr>
<tr>
<td></td>
<td>• 4 - Java Source</td>
</tr>
<tr>
<td></td>
<td>• 5 - Java resource</td>
</tr>
<tr>
<td></td>
<td>• 6 - Java class</td>
</tr>
<tr>
<td></td>
<td>• 7 - type</td>
</tr>
<tr>
<td></td>
<td>• 8 - Java shared data</td>
</tr>
<tr>
<td></td>
<td>• 9 - index</td>
</tr>
<tr>
<td>schema</td>
<td>Schema of the object: c. If no schema is specified in name, then the schema is determined by resolving the name.</td>
</tr>
<tr>
<td>part1</td>
<td>First part of the name. The type of this name is specified part1_type (synonym or package).</td>
</tr>
<tr>
<td>part2</td>
<td>If this is non-NULL, then this is a subprogram name. If part1 is non-NULL, then the subprogram is within the package indicated by part1. If part1 is NULL, then the subprogram is a top-level subprogram.</td>
</tr>
<tr>
<td>dblink</td>
<td>If this is non-NULL, then a database link was either specified as part of name or name was a synonym which resolved to something with a database link. In this case, if further name translation is desired, then you must call the DBMS_UTILITY.NAME_RESOLVE procedure on this remote node.</td>
</tr>
<tr>
<td>part1_type</td>
<td>Type of part1 is:</td>
</tr>
<tr>
<td></td>
<td>• 5 - synonym</td>
</tr>
<tr>
<td></td>
<td>• 7 - procedure (top level)</td>
</tr>
<tr>
<td></td>
<td>• 8 - function (top level)</td>
</tr>
<tr>
<td></td>
<td>• 9 - package</td>
</tr>
<tr>
<td>object_number</td>
<td>Object identifier</td>
</tr>
</tbody>
</table>

Exceptions

All errors are handled by raising exceptions. A wide variety of exceptions are possible, based on the various syntax error that are possible when specifying object names.

183.6.31 NAME_TOKENIZE Procedure

This procedure calls the parser to parse the given name as a [. b [. c ]][@ dblink ].

It strips double quotes, or converts to uppercase if there are no quotes. It ignores comments of all sorts, and does no semantic analysis. Missing values are left as NULL.

Syntax

DBMS_UTILITY.NAME_TOKENIZE (
    name IN VARCHAR2,
    a OUT VARCHAR2,
    b OUT VARCHAR2,
Parameters

Table 183-31  NAME_RESOLVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Input name, consisting of SQL identifiers (for example, scott.foo@dblink)</td>
</tr>
<tr>
<td>a</td>
<td>Output for the first token of the name</td>
</tr>
<tr>
<td>b</td>
<td>Output for the second token of the name (if applicable)</td>
</tr>
<tr>
<td>c</td>
<td>Output for the third token of the name (if applicable)</td>
</tr>
<tr>
<td>dblink</td>
<td>Output for the dblink of the name</td>
</tr>
<tr>
<td>nextpos</td>
<td>Next position after parsing the input name</td>
</tr>
</tbody>
</table>

183.6.32 OLD_CURRENT_SCHEMA Function

This function returns the session value from `sys_context('userenv','current_schema')`.

Syntax

```sql
DBMS_UTILITY.OLD_CURRENT_SCHEMA
RETURN VARCHAR2;
```

183.6.33 OLD_CURRENT_USER Function

This function returns the session value from `sys_context('userenv','current_user')`.

Syntax

```sql
DBMS_UTILITY.OLD_CURRENT_USER
RETURN VARCHAR2;
```

183.6.34 PORT_STRING Function

This function returns a string that identifies the operating system and the two task protocol version of the database. For example, "VAX/VMX-7.1.0.0"

The maximum length is port-specific.

Syntax

```sql
DBMS_UTILITY.PORT_STRING
RETURN VARCHAR2;
```

Pragmas

```sql
pragma restrict_references(port_string, WNDS, RNDS, WNPS, RNPS);
```
183.6.35 SQLID_TO_SQLHASH Function

This function converts a SQL ID into a hash value.

**Syntax**

```sql
DBMS_UTILITY.SQLID_TO_SQLHASH (sql_id IN VARCHAR2) RETURN NUMBER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>SQL ID of a SQL statement. Must be VARCHAR2(13).</td>
</tr>
</tbody>
</table>

183.6.36 TABLE_TO_COMMA Procedures

This procedure converts a PL/SQL table of names into a comma-delimited list of names.

This takes a PL/SQL table, 1..n, terminated with n+1 null. The second version supports fully-qualified attribute names.

**Syntax**

```sql
DBMS_UTILITY.TABLE_TO_COMMA (tab IN UNCL_ARRAY, tablen OUT BINARY_INTEGER, list OUT VARCHAR2);
```

```sql
DBMS_UTILITY.TABLE_TO_COMMA (tab IN lname_array, tablen OUT BINARY_INTEGER, list OUT VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab</td>
<td>PL/SQL table which contains list of table names</td>
</tr>
<tr>
<td>tablen</td>
<td>Number of tables in the PL/SQL table</td>
</tr>
<tr>
<td>list</td>
<td>Comma separated list of tables</td>
</tr>
</tbody>
</table>

**Return Values**

A comma-delimited list and the number of elements found in the table.
183.6.37 VALIDATE Procedure

This procedure makes invalid database objects valid.

Syntax

```sql
DBMS_UTILITY.VALIDATE(
    object_id       NUMBER);

DBMS_UTILITY.VALIDATE(
    owner          VARCHAR2,
    objname        VARCHAR2,
    namespace      NUMBER,
    edition_name   := SYS_CONTEXT ('USERENV',
                                'CURRENT_EDITION'));
```

Parameters

Table 183-34 VALIDATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Name of the user who owns the object. Same as the OWNER field in ALL_OBJECTS.</td>
</tr>
<tr>
<td>objname</td>
<td>Name of the object to be validated. Same as the OBJECT_NAME field in ALL_OBJECTS.</td>
</tr>
</tbody>
</table>
| namespace | Namespace of the object. Same as the namespace field in obj$. Equivalent numeric values are as follows:  
  - 1 — TABLE/PROCEDURE/TUPLE  
  - 2 — BODY  
  - 3 — TRIGGER  
  - 4 — INDEX  
  - 5 — CLUSTER  
  - 8 — LOB  
  - 9 — DIRECTORY  
  - 10 — QUEUE  
  - 11 — REPLICATION OBJECT GROUP  
  - 12 — REPLICATION PROPAGATOR  
  - 13 — JAVA SOURCE  
  - 14 — JAVA RESOURCE  
  - 58 — (Data Mining) MODEL |
| edition_name | [Note: Currently not operable. Reserved for future use] |

Usage Notes

- No errors are raised if the object does not exist or is already valid or is an object that cannot be validated.
- If the object being validated is not actual in the specified edition, the subprogram automatically switches into the edition in which the object is actual prior to validation. That is, a call to VALIDATE will not actualize the object in the specified edition.
- The INVALIDATE Procedure invalidates a database object and optionally changes its PL/SQL compiler parameter settings. The object to be invalidated is specified by its
object_id. The subprogram automatically switches to the edition in which the object is actual prior to invalidation. That is, a call to INVALIDATE will not actualize the object in the current edition.

183.6.38 WAIT_ON_PENDING_DML Function

This function waits until all transactions (other than the caller's own) that have locks on the listed tables and began prior to the specified scn have either committed or been rolled back.

Syntax

DBMS_UTILITY.WAIT_ON_PENDING_DML ( 
  tables IN VARCHAR2,
  timeout IN BINARY_INTEGER,
  scn IN OUT NUMBER)
RETURN BOOLEAN;

Parameters

Table 183-35  WAIT_ON_PENDING_DML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tables</td>
<td>Comma-separated list of one or more table names. The list must be valid for COMMA_TO_TABLE Procedures, and each item valid to the NAME_RESOLVE Procedure. Neither column specifiers nor DBLINK (database link) specifiers are allowed in the names, and each name must resolve to an existing table in the local database.</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum number of seconds to wait, totalled across all tables/transactions. A NULL or negative value will cause a very long wait.</td>
</tr>
<tr>
<td>scn</td>
<td>SCN prior to which transactions must have begun to be considered relevant to this request. If the value is NULL or not recognized as a meaningful scn on input, the most current SCN across all instances will be used and will be set into the passed argument as an output. If a meaningful value is passed in, its value will be preserved in the output.</td>
</tr>
</tbody>
</table>

Return Values

TRUE if all relevant transactions have committed or been rolled back, FALSE if the timeout occurred prior to all relevant transactions committing or being rolled back.
DBMS_WARNING

The DBMS_WARNING package provides a way to manipulate the behavior of PL/SQL warning messages, in particular by reading and changing the setting of the PLSQL_WARNINGS initialization parameter to control what kinds of warnings are suppressed, displayed, or treated as errors. This package provides the interface to query, modify and delete current system or session settings.

This chapter contains the following topics:

- Security Model
- Summary of DBMS_WARNING Subprograms

184.1 DBMS_WARNING Security Model

Note that for all the following interfaces, if value of the scope parameter is SYSTEM, then the user must have ALTER SYSTEM privilege.

184.2 Summary of DBMS_WARNING Subprograms

This table lists the DBMS_WARNING subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_WARNING_SETTING_CAT Procedure</td>
<td>Modifies the current session or system warning settings of the warning_category previously supplied</td>
</tr>
<tr>
<td>ADD_WARNING_SETTING_NUM Procedure</td>
<td>Modifies the current session or system warning settings of the or warning_number previously supplied</td>
</tr>
<tr>
<td>GET_CATEGORY Function</td>
<td>Returns the category name, given the message number</td>
</tr>
<tr>
<td>GET_WARNING_SETTING_CAT Function</td>
<td>Returns the specific warning category in the session</td>
</tr>
<tr>
<td>GET_WARNING_SETTING_NUM Function</td>
<td>Returns the specific warning number in the session</td>
</tr>
<tr>
<td>GET_WARNING_SETTING_STRING Function</td>
<td>Returns the entire warning string for the current session</td>
</tr>
<tr>
<td>SET_WARNING_SETTING_STRING Procedure</td>
<td>Replaces previous settings with the new value</td>
</tr>
</tbody>
</table>
184.2.1 ADD_WARNING_SETTING_CAT Procedure

You can modify the current session's or system's warning settings with the value supplied in this procedure. The value will be added to the existing parameter setting if the value for the `warning_category` or `warning_value` has not been set, or override the existing value.

The effect of calling this function is same as adding the qualifier (ENABLE/DISABLE/ERROR) on the category specified to the end of the current session or system setting.

Syntax

```sql
DBMS_WARNING.ADD_WARNING_SETTING_CAT (
    warning_category    IN    VARCHAR2,
    warning_value       IN    VARCHAR2,
    scope               IN    VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning_category</td>
<td>Name of the category. Allowed values are ALL, INFORMATIONAL, SEVERE and PERFORMANCE.</td>
</tr>
<tr>
<td>warning_value</td>
<td>Value for the category. Allowed values are ENABLE, DISABLE, and ERROR.</td>
</tr>
<tr>
<td>scope</td>
<td>Specifies if the changes are being performed in the session context or the system context. Allowed values are SESSION or SYSTEM.</td>
</tr>
</tbody>
</table>

184.2.2 ADD_WARNING_SETTING_NUM Procedure

You can modify the current session or system warning settings with the value supplied in this procedure. If the value was already set, you will override the existing value.

The effect of calling this function is same as adding the qualifier (ENABLE/DISABLE/ERROR) on the category specified to the end of the current session or system setting.

Syntax

```sql
DBMS_WARNING.ADD_WARNING_SETTING_NUM (
    warning_number      IN    NUMBER,
    warning_value       IN    VARCHAR2,
    scope               IN    VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning_number</td>
<td>The warning number. Allowed values are all valid warning numbers.</td>
</tr>
</tbody>
</table>
Table 184-3  (Cont.) ADD_WARNING_SETTING_NUM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning_value</td>
<td>Value for the category. Allowed values are ENABLE, DISABLE, and ERROR.</td>
</tr>
<tr>
<td>scope</td>
<td>Specifies if the changes are being performed in the session context or the system context. Allowed values are SESSION or SYSTEM.</td>
</tr>
</tbody>
</table>

Example 184-1  Enabling the Deprecation Warnings

This example shows how to enable the DEPRECATE pragma warnings using the DBMS_WARNING package for the session.

```plaintext
BEGIN
    DBMS_WARNING.ADD_WARNING_SETTING_NUM (6019, 'ENABLE', 'SESSION');
    DBMS_WARNING.ADD_WARNING_SETTING_NUM (6020, 'ENABLE', 'SESSION');
    DBMS_WARNING.ADD_WARNING_SETTING_NUM (6021, 'ENABLE', 'SESSION');
    DBMS_WARNING.ADD_WARNING_SETTING_NUM (6022, 'ENABLE', 'SESSION');
END;
```

184.2.3 GETCATEGORY Function

This function returns the category name, given the message number.

Syntax

```plaintext
DBMS_WARNING.GETCATEGORY ( warning_number IN pls_integer) RETURN VARCHAR2;
```

Parameters

Table 184-4  GETCATEGORY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning_number</td>
<td>The warning message number.</td>
</tr>
</tbody>
</table>
184.2.4 GET_WARNING_SETTING_CAT Function

This function returns the specific warning category setting for the current session.

Syntax

```sql
DBMS_WARNING.GET_WARNING_SETTING_CAT (
    warning_category IN VARCHAR2)
RETURN warning_value;
```

Parameters

Table 184-5  GET_WARNING_SETTING_CAT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning_category</td>
<td>Name of the category. Allowed values are all valid category names (ALL, INFORMATIONAL, SEVERE and PERFORMANCE).</td>
</tr>
</tbody>
</table>

184.2.5 GET_WARNING_SETTING_NUM Function

This function returns the specific warning number setting for the current session.

Syntax

```sql
DBMS_WARNING.GET_WARNING_SETTING_NUM (
    warning_number IN NUMBER)
RETURN warning_value;
```

Parameters

Table 184-6  GET_WARNING_SETTING_NUM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning_number</td>
<td>Warning number. Allowed values are all valid warning numbers.</td>
</tr>
</tbody>
</table>

184.2.6 GET_WARNING_SETTING_STRING Function

This function returns the entire warning string for the current session.

Syntax

```sql
DBMS_WARNING.GET_WARNING_SETTING_STRING
RETURN VARCHAR2;
```

Usage Notes

Use this function when you do not have SELECT or READ privilege on v$parameter or v$parameter2 fixed tables, or if you want to parse the warning string yourself and then modify and set the new value using SET_WARNING_SETTING_STRING.
184.2.7 SET_WARNING_SETTING_STRING Procedure

This procedure replaces previous settings with the new value.

The warning string may contain mix of category and warning numbers using the same syntax as used on the right hand side of ‘=’ when issuing an ALTER SESSION or SYSTEM SET PLSQL_WARNINGS command. This will have same effect as ALTER SESSION or ALTER SYSTEM command.

Syntax

```
DBMS_WARNING.SET_WARNING_SETTING_STRING (    
   warning_value   IN   VARCHAR2,    
   scope           IN   VARCHAR2);
```

Parameters

**Table 184-7 SET_WARNING_SETTING_STRING Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning_value</td>
<td>The new string that will constitute the new value.</td>
</tr>
<tr>
<td>scope</td>
<td>This will specify if the changes are being done in the session context, or system context. Allowed values are SESSION or SYSTEM.</td>
</tr>
</tbody>
</table>
The DBMS_WM package provides an interface to Oracle Database Workspace Manager (often referred to as Workspace Manager).

For a complete description of this package, see DBMS_WM in the Oracle Database Workspace Manager Developer's Guide.
The **DBMS_WORKLOAD_CAPTURE** package configures the Workload Capture system and produce the workload capture data.

Replay of this capture is implemented by way of the **DBMS_WORKLOAD_REPLAY** package.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Summary of DBMS_WORKLOAD_CAPTURE Subprograms**

<table>
<thead>
<tr>
<th>See Also:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Database Testing Guide for more information about database replay</td>
</tr>
</tbody>
</table>

### 186.1 DBMS_WORKLOAD_CAPTURE Overview

Since the capture infrastructure is instance wide (and also within an Oracle Real Application Clusters (Oracle RAC)), only one workload capture is being produced at any point in time. Thus capture interfaces do not need a state object passed in as a parameter since there is one single state at any point in time. This means that all subprograms cannot be methods of an object but are package wide PL/SQL subprograms.

### 186.2 DBMS_WORKLOAD_CAPTURE Security Model

The security model describes the privileges needed for using **DBMS_WORKLOAD_CAPTURE**.

The following code describes the minimal set of privileges required to:

- Create directory objects
- Operate the interface provided by the **DBMS_WORKLOAD_CAPTURE** and **DBMS_WORKLOAD_REPLAY** packages
- Act as a replay client user (wrc someuser/somepassword or wrc USER=someuser PASSWORD=somepassword)

```sql
DROP USER rom1 CASCADE;
CREATE USER rom1 IDENTIFIED BY rom1;

GRANT EXECUTE ON DBMS_WORKLOAD_CAPTURE TO rom1;
GRANT EXECUTE ON DBMS_WORKLOAD_REPLAY TO rom1;

GRANT CREATE SESSION TO rom1;
GRANT CREATE ANY DIRECTORY TO rom1;
```
GRANT SELECT_CATALOG_ROLE TO rom1;
GRANT BECOME USER TO rom1;

Appropriate OS permissions are required to access and manipulate files and directories on both the capture and replay system. This means that the Oracle process(es) and the OS user performing the capture or replay must be able to access and manipulate at least one common directory accessible from the host where the instance is running. Additionally, the OS user performing the replay should be able to execute wrc on hosts that are used for the replay clients and be able to access the file system appropriately to copy the capture to the replay clients' hosts if required.

186.3 Summary of DBMS_WORKLOAD_CAPTURE Subprograms

This table lists the DBMS_WORKLOAD_CAPTURE package subprograms in alphabetical order.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_FILTER Procedures</td>
<td>Adds a specified filter</td>
</tr>
<tr>
<td>DECRYPT_CAPTURE Procedure</td>
<td>This procedure decrypts sensitive data in workload capture that was encrypted using various advanced encryption standards such as AES128, AES192, or AES256.</td>
</tr>
<tr>
<td>DELETE_CAPTURE_INFO Procedure</td>
<td>Deletes the rows in the DBA_WORKLOAD_CAPTURES and DBA_WORKLOAD_FILTERS views that corresponds to the given workload capture ID</td>
</tr>
<tr>
<td>DELETE_FILTER Procedure</td>
<td>Deletes a specified filter</td>
</tr>
<tr>
<td>ENCRYPT_CAPTURE Procedure</td>
<td>This procedure encrypts sensitive data in workload capture using various advanced encryption standards such as AES128, AES192, or AES256.</td>
</tr>
<tr>
<td>EXPORT_AWR Procedure</td>
<td>Exports the AWR snapshots associated with a given capture ID</td>
</tr>
<tr>
<td>FINISH_CAPTURE Procedure</td>
<td>Finalizes the workload capture by signaling all connected sessions to stop capture, and stops future requests to the database from being captured</td>
</tr>
<tr>
<td>GET_CAPTURE_INFO Function</td>
<td>Retrieves all the information regarding a workload capture present in the stipulated directory, imports the information into the DBA_WORKLOAD_CAPTURES and DBA_WORKLOAD_FILTERS views, and returns the appropriate DBA_WORKLOAD_CAPTURES.ID</td>
</tr>
<tr>
<td>IMPORT_AWR Function</td>
<td>Imports the AWR snapshots associated with a given capture ID</td>
</tr>
<tr>
<td>REPORT Function</td>
<td>Returns a report on the workload capture under consideration using one or more different sources</td>
</tr>
<tr>
<td>START_CAPTURE Procedure</td>
<td>Initiates workload capture on all instances</td>
</tr>
</tbody>
</table>
186.3.1 ADD_FILTER Procedures

This procedure adds a filter to capture a subset of the workload.

Syntax

```sql
DBMS_WORKLOAD_CAPTURE.ADD_FILTER (
    fname           IN   VARCHAR2 NOT NULL,
    fattribute      IN   VARCHAR2 NOT NULL,
    fvalue          IN   VARCHAR2 NOT NULL);
```

```sql
DBMS_WORKLOAD_CAPTURE.ADD_FILTER (
    fname           IN   VARCHAR2 NOT NULL,
    fattribute      IN   VARCHAR2 NOT NULL,
    fvalue          IN   NUMBER NOT NULL);
```

Parameters

Table 186-2  ADD_FILTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>Name for the filter to be added. Can be used to delete the filter later if it is not required. (Mandatory)</td>
</tr>
<tr>
<td>fattribute</td>
<td>Specifies the attribute on which the filter needs to be applied (Mandatory). The possible values are:</td>
</tr>
<tr>
<td></td>
<td>• INSTANCE_NUMBER - type NUMBER</td>
</tr>
<tr>
<td></td>
<td>• USER - type STRING</td>
</tr>
<tr>
<td></td>
<td>• MODULE - type STRING</td>
</tr>
<tr>
<td></td>
<td>• ACTION - type STRING</td>
</tr>
<tr>
<td></td>
<td>• PROGRAM - type STRING</td>
</tr>
<tr>
<td></td>
<td>• SERVICE - type STRING</td>
</tr>
<tr>
<td></td>
<td>• PDB_NAME - type STRING</td>
</tr>
<tr>
<td>fvalue</td>
<td>Specifies the value to which the given attribute should be equal to for the filter to be considered active. Wildcards like ‘%’ are acceptable for all attributes that are of type STRING. This means that the filter for a NUMBER attribute is parsed as &quot;attribute = value&quot;, with the filter for a STRING attribute parsed as &quot;attribute like value&quot; (Mandatory).</td>
</tr>
</tbody>
</table>

Usage Notes

- The workload capture filters work in either the DEFAULT INCLUSION or the DEFAULT EXCLUSION mode as determined by the default_action input to the START_CAPTURE Procedure.
- ADD_FILTER adds a new filter that affects the next workload capture, and whether the filters are considered as INCLUSION filters or EXCLUSION filters depends on the value of the default_action input to START_CAPTURE Procedure.
- Filters once specified are valid only for the next workload capture. If the same set of filters need to be used for subsequent capture, they need to be specified each time before the START_CAPTURE Procedure is executed.
- All the filters are listed in the DBA_WORKLOAD_FILTERS view.
• You can capture the workload for a particular PDB by specifying a filter of PDB type.

Examples

• By default, a capture works in an INCLUSION mode, which records everything except for those requests that satisfy conditions of specified filters. For example, if you want to exclude all requests from SCOTT, you can add the following filter before starting a capture.

  EXEC DBMS_WORKLOAD_CAPTURE.ADD_FILTER ('filter user1', 'USER', 'SCOTT');

• Multiple filters are evaluated according to the logical disjunction operator OR. Therefore, if you want to record workload for both SCOTT and JOHN, you add an additional filter:

  EXEC DBMS_WORKLOAD_CAPTURE.ADD_FILTER ('filter user2', 'USER', 'JOHN');

• In a CDB, you exclude the workload of a particular PDB by the filter:

  EXEC DBMS_WORKLOAD_CAPTURE.ADD_FILTER ('filter pdb workload', 'PDB_NAME', 'CDB1_PDB1');

• To use DBMS_APPLICATION_INFO to identify workload that is issued to the database:

  DBMS_APPLICATION_INFO.SET_MODULE('ORDER_ENTRY', NULL);
  -- run some SQL here
  DBMS_APPLICATION_INFO.SET_ACTION('ORDER_ENTRY_LOG');
  -- run logging SQL

• If having captured workload, you want to exclude the logging SQL from the captured, specify a filter for capture:

  DBMS_WORKLOAD_CAPTURE.ADD_FILTER('filter logging operations', 'ACTION', 'ORDER_ENTRY_LOG');

• To filter out the full order entry transaction, define a filter:

  DBMS_WORKLOAD_CAPTURE.ADD_FILTER('filter order entry', 'MODULE', 'ORDER_ENTRY');

186.3.2 DECRYPT_CAPTURE Procedure

This procedure decrypts sensitive data in workload capture that was encrypted using various advanced encryption standards such as AES128, AES192, or AES256.

Syntax

  DBMS_WORKLOAD_CAPTURE.DECRYPT_CAPTURE (  
    src_dir               IN   VARCHAR2,  
    dst_dir               IN   VARCHAR2);
Parameters

| Table 186-3  DECRYPT_CAPTURE Procedure Parameters |
|-------------|---------------------------------------------------|
| Parameter   | Description                                       |
| src_dir     | A directory object pointing to the workload capture to be decrypted. This parameter is case sensitive. |
| dst_dir     | A directory object pointing to an OS path that has write permissions. The decrypted capture files will be written to this directory. This parameter is case sensitive. |

Usage Notes

This procedure relies on a software keystore. The identifier is oracle.rat.database_replay.encryption (case-sensitive).

186.3.3 DELETE_CAPTURE_INFO Procedure

This procedure deletes the rows in the DBA_WORKLOAD_CAPTURES and DBA_WORKLOAD_FILTERS views that corresponds to the given workload capture ID.

Syntax

```
DBMS_WORKLOAD_CAPTURE.DELETE_CAPTURE_INFO
  capture_id     IN   NUMBER);
```

Parameters

| Table 186-4  DELETE_CAPTURE_INFO Procedure Parameters |
|-------------|---------------------------------------------------|
| Parameter   | Description                                       |
| capture_id  | ID of the workload capture that needs to be deleted. Corresponds to DBA_WORKLOAD_CAPTURES.ID. (Mandatory) |

Usage Notes

Passing the ID of a capture that is in progress will first automatically stop that capture.

186.3.4 DELETE_FILTER Procedure

This procedure deletes a specified filter.

Syntax

```
DBMS_WORKLOAD_CAPTURE.DELETE_FILTER (
  filter_name           IN   VARCHAR2(40) NOT NULL);
```


Parameters

Table 186-5  DELETE_FILTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter_name</td>
<td>Filter to be deleted</td>
</tr>
</tbody>
</table>

Usage Notes

The DELETE_FILTER Procedure only affects filters that have not been used by any previous capture. Consequently, filters can be deleted only if they have been added using the ADD_FILTER Procedures after any capture has been completed. Filters that have been added using ADD_FILTER before a START_CAPTURE and FINISH_CAPTURE cannot be deleted anymore using this subprogram.

186.3.5 ENCRYPT_CAPTURE Procedure

This procedure encrypts sensitive data in workload capture using various advanced encryption standards such as AES128, AES192, or AES256. To encrypt capture on the fly, use encryption parameter in START_CAPTURE procedure.

Syntax

```sql
DBMS_WORKLOAD_CAPTURE.ENCRYPT_CAPTURE
    src_dir     IN VARCHAR2,
    dst_dir     IN VARCHAR2,
    encryption  IN VARCHAR2 DEFAULT 'AES256');
```

Parameters

Table 186-6  ENCRYPT_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_dir</td>
<td>A directory object pointing to the workload capture to be encrypted. The parameter value is case sensitive.</td>
</tr>
<tr>
<td>dst_dir</td>
<td>A directory object pointing to an OS path that has write permissions. The encrypted capture files will be written to this directory. The parameter value is case sensitive.</td>
</tr>
<tr>
<td>encryption</td>
<td>Specifies if sensitive data in workload capture is encrypted or not. The possible values are: AES128, AES192, and AES256. The default value is AES256.</td>
</tr>
</tbody>
</table>

Usage Notes

For encrypted capture, the ENCRYPT_CAPTURE Procedure relies on a software keystore. The identifier is oracle.rat.database_replay.encryption (case-sensitive).
186.3.6 EXPORT_AWR Procedure

This procedure exports the AWR snapshots associated with a given capture ID.

Syntax

```sql
DBMS_WORKLOAD_CAPTURE.EXPORT_AWR (
    capture_id IN NUMBER);
```

Parameters

Table 186-7  EXPORT_AWR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_id</td>
<td>ID of the capture whose AWR snapshots are to be exported. (Mandatory)</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure works only if the corresponding workload capture was performed in the current database (meaning that the corresponding row in `DBA_WORKLOAD_CAPTURES` was not created by calling the `GET_CAPTURE_INFO Function`) and the AWR snapshots that correspond to the original capture time period are still available.

186.3.7 FINISH_CAPTURE Procedure

This procedure signals all connected sessions to stop the workload capture and stops future requests to the database from being captured.

Syntax

```sql
DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE
    timeout IN NUMBER DEFAULT 30
    reason IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 186-8  FINISH_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Specifies in seconds for how long the procedure should wait before it times out. Pass 0 if you want to cancel the current workload capture and not wait for any sessions to flush it's capture buffers. Default value: 30 seconds</td>
</tr>
<tr>
<td>reason</td>
<td>Specifies a reason for calling the procedure. The reason appears in the column <code>ERROR_MESSAGE</code> of the view <code>DBA_WORKLOAD_CAPTURES</code>.</td>
</tr>
</tbody>
</table>

Usage Notes

- By default, `FINISH_CAPTURE` waits for 30 seconds to receive a successful acknowledgement from all sessions in the database cluster before timing out.
• All sessions that either were in the middle of executing a user request or received a new user request, while FINISH_CAPTURE was waiting for acknowledgements, flush their buffers and send back their acknowledgement to FINISH_CAPTURE.

• If a database session remains idle (waiting for the next user request) throughout the duration of FINISH_CAPTURE, the session might have unflushed capture buffers and does not send its acknowledgement to FINISH_CAPTURE.

To avoid this, do not have sessions that remain idle (waiting for the next user request) while invoking FINISH_CAPTURE. Either close the database session(s) before running FINISH_CAPTURE or send new database requests to those sessions during FINISH_CAPTURE.

186.3.8 GET_CAPTURE_INFO Function

This procedure retrieves all information regarding a workload capture present in the stipulated directory, imports the information into the DBA_WORKLOAD_CAPTURES and DBA_WORKLOAD_FILTERS views, and returns the appropriate DBA_WORKLOAD_CAPTURES.ID.

Syntax

```sql
DBMS_WORKLOAD_CAPTURE.GET_CAPTURE_INFO
    (dir IN VARCHAR2)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir</td>
<td>Name of the DIRECTORY object (case sensitive) where all the workload capture files are located (Mandatory)</td>
</tr>
</tbody>
</table>

Usage Notes

If an appropriate row describing the capture in the stipulated directory already exists in DBA_WORKLOAD_CAPTURES, the GET_CAPTURE_INFO Function simply returns that row's DBA_WORKLOAD_CAPTURES.ID. If no existing row matches the capture present in the stipulated directory a new row is inserted to DBA_WORKLOAD_CAPTURES and that row's ID is returned.

186.3.9 IMPORT_AWR Function

This procedure imports the AWR snapshots associated with a given capture ID provided those AWR snapshots were exported earlier from the original capture system using the EXPORT_AWR procedure.

Syntax

```sql
DBMS_WORKLOAD_CAPTURE.IMPORT_AWR (
    capture_id IN NUMBER,
    staging_schema IN VARCHAR2,
    force_cleanup IN BOOLEAN DEFAULT FALSE)
RETURN NUMBER;
```
Parameters

Table 186-10  IMPORT_AWR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_id</td>
<td>ID of the capture whose AWR snapshots should be imported. (Mandatory)</td>
</tr>
<tr>
<td>staging_schema</td>
<td>Name of a valid schema in the current database which can be used as a staging area while importing the AWR snapshots from the capture directory to the SYS_AWR schema. The SYS schema is not a valid input. (Mandatory, Case sensitive).</td>
</tr>
<tr>
<td>force_cleanup</td>
<td>Values:</td>
</tr>
<tr>
<td></td>
<td>• TRUE - any AWR data present in the given staging_schema are removed before the actual import operation. All tables with names that match any of the tables in AWR are dropped before the actual import. This typically is equivalent to dropping all tables returned by the following SQL:</td>
</tr>
<tr>
<td></td>
<td>SELECT table_name FROM dba_tables</td>
</tr>
<tr>
<td></td>
<td>WHERE owner = staging_schema</td>
</tr>
<tr>
<td></td>
<td>AND table_name like 'WR_$%';</td>
</tr>
<tr>
<td></td>
<td>Use this option only if you are sure that there are no important data in any such tables in the staging_schema.</td>
</tr>
<tr>
<td></td>
<td>• FALSE - (default) no tables dropped from the staging_schema prior to the import operation</td>
</tr>
</tbody>
</table>

Return Values

Returns the new randomly generated database ID that was used to import the AWR snapshots. The same value can be found in the AWR_DBID column in the DBA_WORKLOAD_CAPTURES view.

Usage Notes

IMPORT_AWR fails if the staging_schema provided as input contains any tables with the same name as any of the AWR tables, such as WRM$_SNAPSHOT or WRH$_PARAMETER. Please drop any such tables in the staging_schema before invoking IMPORT_AWR.

Related Topics

• EXPORT_AWR Procedure
  This procedure exports the AWR snapshots associated with a given capture ID.

186.3.10 REPORT Function

This function generates a report on the stipulated workload capture.

Syntax

```
DBMS_WORKLOAD_CAPTURE.REPORT (  
capture_id IN NUMBER,  
format IN VARCHAR2)  
RETURN CLOB;
```
Parameters

### Table 186-11 REPORT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_id</td>
<td>ID of the workload capture whose capture report is required. (Mandatory) This relates to the directory that contains the workload capture on which the Report needs to be generated. Should be a valid DIRECTORY object that points to a valid directory in the host system that contains a workload capture.</td>
</tr>
<tr>
<td>format</td>
<td>Specifies the report format. Valid values are DBMS_WORKLOAD_CAPTURE.TYPE_TEXT and DBMS_WORKLOAD_CAPTURE.TYPE_HTML. (Mandatory)</td>
</tr>
</tbody>
</table>

Return Values

The report body in the desired format returned as a CLOB.

### Table 186-12 Constants Used by Report Function

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_HTML</td>
<td>VARCHAR2(4)</td>
<td>’HTML’</td>
<td>Generates the HTML version of the report</td>
</tr>
<tr>
<td>TYPE_TEXT</td>
<td>VARCHAR2(4)</td>
<td>’TEXT’</td>
<td>Used as input to the format argument to generate the text version of the report</td>
</tr>
</tbody>
</table>

186.3.11 START_CAPTURE Procedure

This procedure initiates workload capture on all instances.

**Syntax**

```sql
DBMS_WORKLOAD_CAPTURE.START_CAPTURE (name IN VARCHAR2, dir IN VARCHAR2, duration IN NUMBER DEFAULT NULL, default_action IN VARCHAR2 DEFAULT 'INCLUDE', auto_unrestrict IN BOOLEAN DEFAULT TRUE, capture_sts IN BOOLEAN DEFAULT FALSE, sts_cap_interval IN NUMBER DEFAULT 300, plsql_mode IN VARCHAR2 DEFAULT 'TOP_LEVEL', encryption IN VARCHAR2 DEFAULT NULL);
```
### Parameters

#### Table 186-13  START_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the workload capture. Allows the workload capture to be given a label, such as “Thanksgiving weekend” or “Christmas peak workload” for future reference. The workload capture's name is preserved along with the captured workload actions. (Mandatory)</td>
</tr>
<tr>
<td>dir</td>
<td>Name of the DIRECTORY object (case sensitive) where all the workload capture files are stored. Should contain enough space to hold all the workload capture files. (Mandatory)</td>
</tr>
<tr>
<td>duration</td>
<td>Optional input to specify the duration (in seconds) for which the workload needs to be captured. DEFAULT is NULL which means that workload capture continues until the user executes DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE.</td>
</tr>
<tr>
<td>default_action</td>
<td>Can be either INCLUDE or EXCLUDE. Determines whether, by default, every user request should be captured or not. Also determines whether the workload filters specified should be considered as INCLUSION filters or EXCLUSION filters.</td>
</tr>
<tr>
<td></td>
<td>• If INCLUDE, by default all user requests to the database are captured, except for the part of the workload defined by the filters. In this case, all the filters specified using the ADD_FILTER Procedures are treated as EXCLUSION filters, determining the workload that is not captured. (DEFAULT, and so all the filters specified are assumed to be EXCLUSION filters.)</td>
</tr>
<tr>
<td></td>
<td>• If EXCLUDE, by default no user request to the database is captured, except for the part of the workload defined by the filters. In this case, all the filters specified using the ADD_FILTER Procedures are treated as INCLUSION filters, determining the workload that is captured.</td>
</tr>
<tr>
<td>auto_unrestrict</td>
<td>Can be either TRUE or FALSE.</td>
</tr>
<tr>
<td></td>
<td>• If TRUE, all instances started up in RESTRICTED mode using STARTUP RESTRICT are automatically unrestricted upon a successful START_CAPTURE. (DEFAULT)</td>
</tr>
<tr>
<td></td>
<td>• If FALSE, no database instance is automatically unrestricted.</td>
</tr>
<tr>
<td>capture_sts</td>
<td>If this parameter is TRUE, a SQL tuning set capture is also started in parallel with workload capture. The resulting SQL tuning set can be exported using the EXPORT_AWR Procedure along with the AWR data. Currently, parallel STS capture is not supported in an Oracle RAC environment, so this parameter has no effect if used in that context. Capture filters defined using the DBMS_WORKLOAD_REPLAY interface do not apply to the SQL tuning set capture. The calling user must have the appropriate privileges (ADMINISTER SQL TUNING SET).</td>
</tr>
<tr>
<td></td>
<td>If starting SQL set capture fails, workload capture is stopped. The reason is stored in DBA_WORKLOAD_CAPTURESERROR_MESSAGE. The default value is FALSE.</td>
</tr>
<tr>
<td>sts_cap_interval</td>
<td>Specifies the capture interval of the SQL set capture from the cursor cache in seconds. The default value is 300.</td>
</tr>
</tbody>
</table>
Table 186-13  (Cont.) START_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plsql_mode</td>
<td>Specifies the PL/SQL capture mode:</td>
</tr>
<tr>
<td></td>
<td>· TOP_LEVEL — only top-level PL/SQL calls are captured</td>
</tr>
<tr>
<td></td>
<td>· EXTENDED — both top-level PL/SQL calls and SQL called from PL/SQL are captured</td>
</tr>
<tr>
<td>encryption</td>
<td>Specify if sensitive data in workload capture is encrypted or not.</td>
</tr>
<tr>
<td></td>
<td>The possible values are:</td>
</tr>
<tr>
<td></td>
<td>· NULL — capture files are not encrypted</td>
</tr>
<tr>
<td></td>
<td>· AES128 — capture files are encrypted using AES128</td>
</tr>
<tr>
<td></td>
<td>· AES192 — capture files are encrypted using AES192</td>
</tr>
<tr>
<td></td>
<td>· AES256 — capture files is encrypted using AES256</td>
</tr>
<tr>
<td></td>
<td>The default value is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

- All user requests sent to database after a successful invocation of START_CAPTURE are recorded in the given dir directory for the given duration provided that one was specified. If no duration was specified, the capture lasts indefinitely until the FINISH_CAPTURE Procedure is executed.

- A workload capture once started continues to record user requests across database instance shutdowns and startups for the specified duration, or until FINISH_CAPTURE is executed, whichever occurs first.

- One can use workload filters (as described with regard to the ADD_FILTER Procedures) to capture only a subset of the user requests sent to the database. By default, when no workload filters are defined, all user requests are captured.

- Workload that is initiated from Oracle Database background processes (such as SMON, PMON, MMON) and Oracle Database Scheduler Jobs (as detailed in the DBMS_SCHEDULER package) is not captured, no matter how the workload filters are defined. These activities should happen automatically on an appropriately configured replay system.

- By default, all database instances that were started up in RESTRICTED mode using STARTUP RESTRICT are UNRESTRICTED upon a successful invocation of START_CAPTURE. Use FALSE for the auto_unrestrict input parameter, if you do not want this behavior.

- It is important to have a well-defined starting point for the workload so that the replay system can be restored to that point before initiating a replay of the captured workload. To have a well-defined starting point for the workload capture, it is preferable not to have any active user sessions when START_CAPTURE is executed. If ongoing sessions have ongoing transactions, those transactions are not replayed properly in subsequent database replays, since only that part of the transaction whose calls were executed after START_CAPTURE are replayed.

- For encrypted capture, the START_CAPTURE Procedure relies on a software keystore. The identifier is oracle.rat.database_replay.encryption (case-sensitive).

- You must configure a software keystore in auto-login mode. Otherwise, if the database is bounced during capture, the capture is automatically terminated.
DBMS_WORKLOAD_REPLAY

The DBMS_WORKLOAD_REPLAY package provides an interface to replay a workload capture. This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_WORKLOAD_REPLAY Subprograms

See Also:

Oracle Database Testing Guide for more information about database replay

187.1 DBMS_WORKLOAD_REPLAY Overview

The DBMS_WORKLOAD_REPLAY package provides an interface to replay a workload capture that was originally created by way of the DBMS_WORKLOAD_CAPTURE package.

Typically, the DBMS_WORKLOAD_CAPTURE package is used in the production system to capture a production workload, and the DBMS_WORKLOAD_REPLAY package is subsequently used in a test system to replay the captured production workload for testing purposes.

Related Topics

- DBMS_WORKLOAD_CAPTURE
  The DBMS_WORKLOAD_CAPTURE package configures the Workload Capture system and produce the workload capture data.

187.2 DBMS_WORKLOAD_REPLAY Security Model

The security model describes the privileges needed for using DBMS_WORKLOAD_REPLAY.

The following code sample shows the minimum set of privileges required to:

- Create directory objects
- Operate the interface provided by the DBMS_WORKLOAD_CAPTURE and DBMS_WORKLOAD_REPLAY packages
- Act as a replay client user (wrc someuser/somepassword or wrc USER=someuser PASSWORD=somepassword)

```sql
DROP USER rom1 CASCADE;
CREATE USER rom1 IDENTIFIED BY rom1;

GRANT EXECUTE ON DBMS_WORKLOAD_CAPTURE TO rom1;
GRANT EXECUTE ON DBMS_WORKLOAD_REPLAY TO rom1;
```
GRANT CREATE SESSION TO rom1;
GRANT CREATE ANY DIRECTORY TO rom1;
GRANT SELECT_CATALOG_ROLE TO rom1;
GRANT BECOME USER TO rom1;

Appropriate OS permissions are required to access and manipulate files and directories on both the capture and replay system. The Oracle process(es) and the OS user performing the capture or replay must be able to access and manipulate at least one common directory accessible from the host where the instance is running.

The replay client is a multithreaded program (an executable named `wrc` located in the `$ORACLE_HOME/bin` directory) where each thread submits a workload from a captured session. The OS user performing the replay must be able to execute `wrc` on hosts that are used for the replay clients and be able to access the file system appropriately to be able to copy the capture to the replay clients' hosts if required.

187.3 Summary of DBMS_WORKLOAD_REPLAY Subprograms

This table lists the DBMS_WORKLOAD_REPLAY package subprograms in alphabetical order.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_CAPTURE Function</td>
<td>Adds the given capture to the current schedule</td>
</tr>
<tr>
<td>ADD_FILTER Procedure</td>
<td>Adds a filter to replay only a subset of the captured workload</td>
</tr>
<tr>
<td>ADD_SCHEDULE_ORDERING Function</td>
<td>Adds a schedule order between two captures</td>
</tr>
<tr>
<td>ASSIGN_GROUP_TO_INSTANCE Procedure</td>
<td>Modifies the view <code>DBA_WORKLOAD_GROUP_ASSIGNMENTS</code></td>
</tr>
<tr>
<td>BEGIN_REPLAY_SCHEDULE Procedure</td>
<td>Initiates the creation of a reusable replay schedule</td>
</tr>
<tr>
<td>CALIBRATE Function</td>
<td>Operates on a processed workload capture directory to estimate the number of hosts and workload replay clients needed to faithfully replay the given workload</td>
</tr>
<tr>
<td>CANCEL_REPLAY Procedure</td>
<td>Cancels the workload replay in progress</td>
</tr>
<tr>
<td>COMPARE_PERIOD_REPORT Procedure</td>
<td>Generates a report comparing a replay to its capture or to another replay of the same capture</td>
</tr>
<tr>
<td>COMPARE_SQLSET_REPORT Function</td>
<td>Generates a report comparing a sqlset captured during replay to one captured during workload capture or to one captured during another replay of the same capture</td>
</tr>
<tr>
<td>CREATE_FILTER_SET Procedure</td>
<td>Uses the replay filters added to create a set of filters to use against the replay in <code>replay_dir</code></td>
</tr>
<tr>
<td>DELETE_FILTER Procedure</td>
<td>Deletes the named filter</td>
</tr>
<tr>
<td>DELETE_REPLAY_INFO Procedure</td>
<td>Deletes the rows in <code>DBA_WORKLOAD_REPLAYS</code> that corresponds to the given workload replay ID</td>
</tr>
<tr>
<td>END_REPLAY_SCHEDULE Procedure</td>
<td>Wraps up the creation of the current schedule</td>
</tr>
</tbody>
</table>
Table 187-1  (Cont.) DBMS_WORKLOAD_REPLAY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXPERIMENT_AWR Procedure</strong></td>
<td>Exports the Automatic Workload Repository (AWR) snapshots associated with a given replay ID</td>
</tr>
<tr>
<td><strong>GENERATE_CAPTURE_SUBSET Procedure</strong></td>
<td>Creates a new capture from an existing workload capture</td>
</tr>
<tr>
<td><strong>GET_DIVERGING_STATEMENT Function</strong></td>
<td>Exports the Automatic Workload Repository (AWR) snapshots associated with a given replay ID</td>
</tr>
<tr>
<td><strong>GET_REPLAY_DIRECTORY Function</strong></td>
<td>Returns the current replay directory set by the SET_REPLAY_DIRECTORY Procedure.</td>
</tr>
<tr>
<td><strong>GET_REPLAY_INFO Function</strong></td>
<td>Retrieves information about the workload capture and the history of all the workload replay attempts from the related directory</td>
</tr>
<tr>
<td><strong>GET_REPLAY_TIMEOUT Procedure</strong></td>
<td>Retrieves the replay timeout setting</td>
</tr>
<tr>
<td><strong>IMPORT_AWR Function</strong></td>
<td>Imports the Automatic Workload Repository (AWR) snapshots associated with a given replay ID</td>
</tr>
<tr>
<td><strong>INITIALIZE_CONSOLIDATED_REPLAY Procedure</strong></td>
<td>Puts the database state in INIT for a multiple-capture replay</td>
</tr>
<tr>
<td><strong>INITIALIZE_REPLAY Procedure</strong></td>
<td>Initializes replay, and loads specific data produced during processing into the database</td>
</tr>
<tr>
<td><strong>IS_REPLAY_PAUSED Function</strong></td>
<td>Reports whether the replay is currently paused</td>
</tr>
<tr>
<td><strong>LOAD_LONG_SQLTEXT Procedure</strong></td>
<td>Loads the captured SQL statements that are longer than 1000 characters to the <code>DBA_WORKLOAD_LONG_SQLTEXT</code> view</td>
</tr>
<tr>
<td><strong>PAUSE_REPLAY Procedure</strong></td>
<td>Pauses the in-progress workload replay</td>
</tr>
<tr>
<td><strong>POPULATE_DIVERGENCE Procedure</strong></td>
<td>Precomputes the divergence information for the given call, stream, or the whole replay so that the GET_DIVERGING_STATEMENT Function returns as quickly as possible for the precomputed calls</td>
</tr>
<tr>
<td><strong>PREPARE_CONSOLIDATED_REPLAY Procedure</strong></td>
<td>Puts the database in a special &quot;Prepare&quot; mode for a multiple-capture replay</td>
</tr>
<tr>
<td><strong>PREPARE_REPLAY Procedure</strong></td>
<td>Puts the database in a special &quot;Prepare&quot; mode</td>
</tr>
<tr>
<td><strong>PROCESS_CAPTURE Procedure</strong></td>
<td>Processes the workload capture found in <code>capture_dir</code> in place</td>
</tr>
<tr>
<td><strong>REMAP_CONNECTION Procedure</strong></td>
<td>Remaps the captured connection to a new one so that the user sessions can connect to the database in a desired way during workload replay</td>
</tr>
<tr>
<td><strong>REMOVE_CAPTURE Procedure</strong></td>
<td>Removes the given capture from the current schedule</td>
</tr>
<tr>
<td><strong>REMOVE_SCHEDULE_ORDERING Procedure</strong></td>
<td>Removes an existing schedule order from the current replay schedule</td>
</tr>
<tr>
<td><strong>REPORT Function</strong></td>
<td>Generates a report on the given workload replay</td>
</tr>
<tr>
<td><strong>RESUME_REPLAY Procedure</strong></td>
<td>Resumes a paused workload replay</td>
</tr>
<tr>
<td><strong>REUSE_REPLAY_FILTER_SET Procedure</strong></td>
<td>Reuses filters in the specified filter set as if each were added using the ADD_SCHEDULE_ORDERING Function</td>
</tr>
</tbody>
</table>
Table 187-1 (Cont.) DBMS_WORKLOAD_REPLAY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_ADVANCED_PARAMETER Procedure</td>
<td>Sets an advanced parameter for replay besides the ones used with the PREPARE_REPLAY Procedure</td>
</tr>
<tr>
<td>SET_REPLAY_DIRECTORY Procedure</td>
<td>Sets a directory that contains multiple workload captures as the current replay directory</td>
</tr>
<tr>
<td>SET_REPLAY_TIMEOUT Procedure</td>
<td>Sets the replay timeout setting</td>
</tr>
<tr>
<td>SET_SQL_MAPPING Procedure</td>
<td>Specifies SQL statements to be skipped or replaced during a database replay operation</td>
</tr>
<tr>
<td>SET_USER_MAPPING Procedure</td>
<td>Sets a new schema or user name to be used during replay instead of the captured user</td>
</tr>
<tr>
<td>START_CONSOLIDATED_REPLAY Procedure</td>
<td>Starts the replay of a multiple-capture capture</td>
</tr>
<tr>
<td>START_REPLAY Procedure</td>
<td>Starts the workload replay</td>
</tr>
<tr>
<td>USE_FILTER_SET Procedure</td>
<td>Uses the given filter set that has been created by calling the CREATE_FILTER_SET Procedure to filter the current replay</td>
</tr>
</tbody>
</table>

187.3.1 ADD_CAPTURE Function

This function adds the given capture to the current schedule. The directory has to be a valid capture processed in the current database’s version. It returns a unique ID that identifies this capture within this schedule.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.ADD_CAPTURE (  
capture_dir_name IN VARCHAR2,  
start_delay_seconds IN NUMBER DEFAULT 0,  
stop_replay IN BOOLEAN FALSE,  
take_begin_snapshot IN BOOLEAN TRUE,  
take_end_snapshot IN BOOLEAN TRUE,  
query_only IN BOOLEAN DEFAULT FALSE)  
RETURN NUMBER;
```

```sql
DBMS_WORKLOAD_REPLAY.ADD_CAPTURE (  
capture_dir_name IN VARCHAR2,  
start_delay_seconds IN NUMBER DEFAULT 0,  
stop_replay IN BOOLEAN FALSE,  
take_begin_snapshot IN BOOLEAN TRUE,  
take_end_snapshot IN BOOLEAN TRUE,  
query_only IN VARCHAR2 DEFAULT 'N')  
RETURN NUMBER;
```
### Parameters

#### Table 187-2 ADD_CAPTURE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_dir_name</td>
<td>Name of the OS directory containing the capture under the replay top-level directory</td>
</tr>
<tr>
<td>start_delay_seconds</td>
<td>Delay time in seconds before the replay of this capture starts</td>
</tr>
<tr>
<td>stop_replay</td>
<td>Stop the replay after it finishes</td>
</tr>
<tr>
<td>take_begin_snapshot</td>
<td>Take an AWR snapshot when the replay of this capture starts</td>
</tr>
<tr>
<td>take_end_snapshot</td>
<td>Take an AWR snapshot when the replay of this capture finishes</td>
</tr>
<tr>
<td>query_only</td>
<td>Replay only the read-only queries of this workload capture</td>
</tr>
</tbody>
</table>

**Usage Notes**

The `SET_REPLAY_DIRECTORY` Procedure must have already been called.

### 187.3.2 ADD_FILTER Procedure

This procedure adds a filter to replay only a subset of the captured workload.

The procedure adds a new filter that is used in the next replay filter set created using the `CREATE_FILTER_SET` Procedure. This filter will be considered an "INCLUSION" or "EXCLUSION" filter depending on the argument passed to `CREATE_FILTER_SET` when creating the filter set.

#### Syntax

```sql
DBMS_WORKLOAD_REPLAY.ADD_FILTER (  
  fname          IN VARCHAR2,  
  fattribute     IN VARCHAR2,  
  fvalue         IN VARCHAR2);
```

```sql
DBMS_WORKLOAD_REPLAY.ADD_FILTER (  
  fname          IN VARCHAR2,  
  fattribute     IN VARCHAR2,  
  fvalue         IN NUMBER);
```

#### Parameters

#### Table 187-3 ADD_FILTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>(Mandatory) Name of the filter. Can be used to delete the filter later if it is not required.</td>
</tr>
</tbody>
</table>
Table 187-3  (Cont.) ADD_FILTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| fattribute | (Mandatory) Specifies the attribute on which the filter is defined as one of the following values of type STRING:  
• USER  
• MODULE  
• ACTION  
• PROGRAM  
• SERVICE  
• CONNECTION_STRING |
| fvalue | (Mandatory) Specifies the value to which the given 'attribute' must be equal to for the filter to be considered active. Wildcards such as '%' are acceptable for all attributes that are of type STRING. Currently all the listed values of fattribute are of type STRING. INSTANCE_NUMBER is a NUMBER attribute. It is currently only supported for capture. |

187.3.3 ADD_SCHEDULE_ORDERING Function

This function adds a schedule order between two captures.

Together, schedule_capture_id and waitfor_capture_id form a schedule ordering that previously added by the ADD_SCHEDULE_ORDERING Function. The order is that replay of capture indicated by schedule_capture_id will not start unless the replay of capture indicated by waitfor_capture_id finishes.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.ADD_SCHEDULE_ORDERING (  
schedule_capture_id IN VARCHAR2,  
waitfor_capture_id IN VARCHAR2)  
RETURN NUMBER;
```

Parameters

Table 187-4  ADD_SCHEDULE_ORDERING Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_capture_id</td>
<td>Points to a capture that has been added to the current replay schedule. According to the new schedule ordering added by this subprogram, its replay will not start until the replay of another capture specified by waitfor_capture_id runs to completion.</td>
</tr>
<tr>
<td>waitfor_capture_id</td>
<td>Points to a capture that has been added to the current replay schedule. According to the new schedule ordering added by this subprogram, the replay of capture specified by schedule_capture_id will not start until the replay of this capture runs to completion.</td>
</tr>
</tbody>
</table>
Return Values
Returns a non-zero error code if the constraint cannot be added

Usage Notes
The two captures must have already been added to the replay schedule.

187.3.4 ASSIGN_GROUP_TO_INSTANCE Procedure

This procedure modifies the view DBA_WORKLOAD_GROUP_ASSIGNMENTS.

Syntax

```
DBMS_WORKLOAD_REPLAY.ASSIGN_GROUP_TO_INSTANCE (
    group_id          IN INTEGER,
    instance_number   IN INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>The identifier of the specified group of capture files</td>
</tr>
<tr>
<td>instance_number</td>
<td>The number used for instance registration. It is equivalent to the INSTANCE_NUMBER column in V$INSTANCE.</td>
</tr>
</tbody>
</table>

See Also:

- DBA_WORKLOAD_GROUP_ASSIGNMENTS in Oracle Database Reference
- V$INSTANCE in Oracle Database Reference

187.3.5 BEGIN_REPLAY_SCHEDULE Procedure

This procedure initiates the creation of a reusable replay schedule.

Syntax

```
DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE  (
    replay_dir_obj     IN     VARCHAR2,
    schedule_name      IN     VARCHAR2);
```
Parameters

Table 187-6 BEGIN_REPLAY_SCHEDULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_dir_obj</td>
<td>Directory object that points to the replay directory that contains all the capture directories involved in the schedule</td>
</tr>
<tr>
<td>schedule_name</td>
<td>Name of the schedule to be replayed</td>
</tr>
</tbody>
</table>

Usage Notes

- Only one schedule can be in creation mode at a time. Calling the subprogram again before end_replay_schedule will raise an error.
- Prerequisites:
  - The workload capture was already processed using the PROCESS_CAPTURE Procedure in the same database version.
  - The user must have copied the capture directory appropriately.
  - The database is not in replay mode.
  - The SET_REPLAY_DIRECTORY Procedure has already been called.

187.3.6 CALIBRATE Function

This function operates on a processed workload capture directory to estimate the number of hosts and workload replay clients needed to faithfully replay the given workload. This function returns the results as an XML CLOB.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.CALIBRATE (  
capture_dir          IN VARCHAR2,  
process_per_cpu      IN BINARY_INTEGER DEFAULT 4,  
threads_per_process  IN BINARY_INTEGER DEFAULT 50)  
RETURN CLOB;
```

Parameters

Table 187-7 CALIBRATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_dir</td>
<td>Name of the directory object that points to the (case sensitive) OS directory that contains processed capture data</td>
</tr>
<tr>
<td>process_per_cpu</td>
<td>Maximum number of processes allowed for each CPU (default is 4)</td>
</tr>
<tr>
<td>threads_per_process</td>
<td>Maximum number of threads allowed for each process (default is 50)</td>
</tr>
</tbody>
</table>

Return Values

Returns a CLOB formatted as XML that contains:
• Information about the capture
• Current database version
• Input parameters to this function
• Number of CPUs and replay clients needed to replay the given workload
• Information about the sessions captured (total number and maximum concurrency)

Usage Notes
• Prerequisite: The input workload capture was already processed using the PROCESS_CAPTURE Procedure in the same database version.
• This procedure will return the same results as the workload replay client in calibrate mode, which can be run as follows.

  $ wrc mode=calibrate replaydir=

187.3.7 CANCEL_REPLAY Procedure

This procedure cancels workload replay in progress. All the external replay clients (WRC) will automatically be notified to stop issuing the captured workload and exit.

Syntax

```
DBMS_WORKLOAD_REPLAY.CANCEL_REPLAY (  
    error_msg      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 187-8  CANCEL_REPLAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_msg</td>
<td>An optional reason for cancelling the replay can be passed which is recorded into DBA_WORKLOAD_REPLAYS.ERROR_MESSAGE.</td>
</tr>
</tbody>
</table>

Usage Notes

Prerequisite: A call to the INITIALIZE_REPLAY Procedure, or PREPARE_REPLAY Procedure, or START_REPLAY Procedure was already issued.

187.3.8 COMPARE_PERIOD_REPORT Procedure

This procedure generates a report comparing a replay to its capture or to another replay of the same capture.

Syntax

```
DBMS_WORKLOAD_REPLAY.COMpare_PERIOD_REPORT (  
    replay_id1   IN   NUMBER,  
    replay_id2   IN   NUMBER,  
    format       IN   VARCHAR2,  
    result       OUT  CLOB );
```
187.3.9 COMPARE_SQLSET_REPORT Function

This procedure generates a report comparing a sqlset captured during replay to one captured during workload capture or to one captured during another replay of the same capture.

Syntax

```sql
DBMS_WORKLOAD_REPLAYCOMPARE_SQLSET_REPORT (    replay_id1    IN NUMBER,
  replay_id2    IN NUMBER,
  format        IN VARCHAR2,
  r_level       IN VARCHAR2 DEFAULT 'ALL',
  r_sections    IN VARCHAR2 DEFAULT 'ALL',
  result        OUT CLOB )
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id1</td>
<td>First ID of the workload replay after a change</td>
</tr>
<tr>
<td>replay_id2</td>
<td>Second ID of the workload replay before a change. If this is NULL, then the comparison is done with the capture.</td>
</tr>
<tr>
<td>format</td>
<td>Specifies the report format. Valid values are DBMS_WORKLOAD_CAPTURE.TYPE_HTML, and DBMS_WORKLOAD_CAPTURE.TYPE_XML.</td>
</tr>
<tr>
<td>r_level</td>
<td>See level parameter in the REPORT_ANALYSIS_TASK Function in the DBMS_SQLPA package</td>
</tr>
<tr>
<td>r_sections</td>
<td>See section parameter in the REPORT_ANALYSIS_TASK Function in the DBMS_SQLPA package</td>
</tr>
<tr>
<td>result</td>
<td>Output of the report (CLOB)</td>
</tr>
</tbody>
</table>

187.3.9 COMPARE_SQLSET_REPORT Function

This procedure generates a report comparing a sqlset captured during replay to one captured during workload capture or to one captured during another replay of the same capture.

Syntax

```sql
DBMS_WORKLOAD_REPLAY_COMPARE_SQLSET_REPORT (    replay_id1    IN NUMBER,
  replay_id2    IN NUMBER,
  format        IN VARCHAR2,
  r_level       IN VARCHAR2 DEFAULT 'ALL',
  r_sections    IN VARCHAR2 DEFAULT 'ALL',
  result        OUT CLOB )
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id1</td>
<td>First ID of the workload replay after a change</td>
</tr>
<tr>
<td>replay_id2</td>
<td>Second ID of the workload replay before a change. If this is NULL, then the comparison is done with the capture.</td>
</tr>
<tr>
<td>format</td>
<td>Specifies the report format. Valid values are DBMS_WORKLOAD_CAPTURE.TYPE_HTML, and DBMS_WORKLOAD_CAPTURE.TYPE_XML.</td>
</tr>
<tr>
<td>r_level</td>
<td>See level parameter in the REPORT_ANALYSIS_TASK Function in the DBMS_SQLPA package</td>
</tr>
<tr>
<td>r_sections</td>
<td>See section parameter in the REPORT_ANALYSIS_TASK Function in the DBMS_SQLPA package</td>
</tr>
<tr>
<td>result</td>
<td>Output of the report (CLOB)</td>
</tr>
</tbody>
</table>
187.3.10 CREATE_FILTER_SET Procedure

This procedure creates a new filter set for the replays at `replay_dir`.

It includes all the replay filters that have already been added by the ADD_FILTER Procedure. After the procedure has completed and replay initiated, the newly-created filter set can be used to filter the replay in `replay_dir` by calling the USE_FILTER_SET Procedure.

Syntax

```
DBMS_WORKLOAD_REPLAY.CREATE_FILTER_SET(
    replay_dir       IN  VARCHAR2,
    filter_set       IN  VARCHAR2,
    default_action   IN  VARCHAR2 DEFAULT 'INCLUDE');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_dir</td>
<td>Object directory of the replay to be filtered</td>
</tr>
<tr>
<td>filter_set</td>
<td>Name of the filter set to create (to use in USE_FILTER_SET Procedure)</td>
</tr>
<tr>
<td>default_action</td>
<td>Can be either INCLUDE or EXCLUDE. Determines whether, by default, every captured call must be replayed or not. Also determines whether the workload filters specified must be considered as INCLUSION filters or EXCLUSION filters. If it is INCLUDE, then by default all captured calls are replayed, except for the part of the workload defined by the filters. In this case, all the filters that were specified using the ADD_SCHEDULE_ORDERING Function are treated as EXCLUSION filters, and will determine the workload that will not be replayed. If it is EXCLUDE, then by default no captured call to the database is replayed, except for the part of the workload defined by the filters. In this case, all the filters that were specified using the ADD_SCHEDULE_ORDERING Function are treated as INCLUSION filters, and will determine the workload that is replayed. Default: INCLUDE and all the filters specified are assumed to be EXCLUSION filters</td>
</tr>
</tbody>
</table>

Usage Notes

This operation must be invoked when no replay is initialized, prepared, or in progress.

187.3.11 DELETE_FILTER Procedure

This procedure deletes the named filter.

Syntax

```
DBMS_WORKLOAD_REPLAY.DELETE_FILTER(
    fname        IN  VARCHAR2);
```
Parameters

Table 187-12  DELETE_FILTER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>(Mandatory) Name of the filter that must be deleted</td>
</tr>
</tbody>
</table>

Usage Notes

The DELETE_FILTER Procedure only affects filters that have not been used by any previous capture. Consequently, filters can be deleted only if they have been added using the ADD_FILTER Procedures after any capture has been completed. Filters that have been added using ADD_FILTER before a START_CAPTURE and FINISH_CAPTURE cannot be deleted anymore using this subprogram.

187.3.12 DELETE_REPLAY_INFO Procedure

This procedure deletes the rows in DBA_WORKLOAD_REPLAYS that correspond to the given workload replay ID.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.DELETE_REPLAY_INFO (replay_id    IN  NUMBER);
```

Parameters

Table 187-13  DELETE_REPLAY_INFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id</td>
<td>(Mandatory) ID of the workload replay that must be deleted. Corresponds to DBA_WORKLOAD_REPLAYS.ID</td>
</tr>
</tbody>
</table>

187.3.13 END_REPLAY_SCHEDULE Procedure

This procedure wraps up the creation of the current schedule. The schedule is now saved and associated with the replay directory and can be used for a replay.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
```

Usage Notes

The BEGIN_REPLAY_SCHEDULE Procedure must have already been called.
187.3.14 EXPORT_AWR Procedure

This procedure exports the AWR snapshots associated with a stipulated replay ID.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.EXPORT_AWR (replay_id IN NUMBER);
```

Parameters

Table 187-14 EXPORT_AWR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id</td>
<td>(Mandatory) ID of the replay whose AWR snapshots are to be exported</td>
</tr>
</tbody>
</table>

Usage Notes

- At the end of each replay, the corresponding AWR snapshots are automatically exported. Consequently, there is no need to do this manually after a workload replay is complete, unless the automatic EXPORT_AWR invocation failed.
- This procedure will work only if the corresponding workload replay was performed in the current database (meaning that the corresponding row in DBA_WORKLOAD_REPLAYS was not created by calling the GET_REPLAY_INFO Function) and the AWR snapshots that correspond to that replay time period are still available.

187.3.15 GENERATE_CAPTURE_SUBSET Procedure

This procedure creates a new capture from an existing workload capture.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.GENERATE_CAPTURE_SUBSET (input_capture_dir IN VARCHAR2, output_capture_dir IN VARCHAR2, new_capture_name IN VARCHAR2, begin_time IN NUMBER, begin_include_incomplete IN BOOLEAN DEFAULT TRUE, end_time IN NUMBER, end_include_incomplete IN BOOLEAN DEFAULT FALSE, parallel_level IN NUMBER DEFAULT NULL);
```

Parameters

Table 187-15 GENERATE_CAPTURE_SUBSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_capture_dir</td>
<td>(Mandatory) Name of the directory object that points to an existing workload capture</td>
</tr>
<tr>
<td>output_capture_dir</td>
<td>(Mandatory) Name of the directory object that points to the new capture</td>
</tr>
</tbody>
</table>
Table 187-15  (Cont.) GENERATE_CAPTURE_SUBSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_capture_name</td>
<td>(Mandatory) Name of new capture</td>
</tr>
<tr>
<td>begin_time</td>
<td>Start of the time range - time offset in seconds from the start of a</td>
</tr>
<tr>
<td></td>
<td>workload capture</td>
</tr>
<tr>
<td>begin_include_incomplete</td>
<td>Column to include incomplete calls caused by begin_time</td>
</tr>
<tr>
<td>end_time</td>
<td>End of the time range - time offset in seconds from the start of a</td>
</tr>
<tr>
<td></td>
<td>workload capture. If end_time is zero or end_time is less or equal than</td>
</tr>
<tr>
<td></td>
<td>begin_time, the time range is invalid. The new capture will use the whole</td>
</tr>
<tr>
<td></td>
<td>duration of the input capture.</td>
</tr>
<tr>
<td>end_include_incomplete</td>
<td>Column to include incomplete calls caused by end_time</td>
</tr>
<tr>
<td>parallel_level</td>
<td>Number of Oracle processes used to process the input captures in a</td>
</tr>
<tr>
<td></td>
<td>parallel fashion. The NULL default value will auto-compute the parallelism</td>
</tr>
<tr>
<td></td>
<td>level based on number of CPUs, whereas a value of 1 will enforce serial</td>
</tr>
<tr>
<td></td>
<td>execution.</td>
</tr>
</tbody>
</table>

187.3.16 GET_DIVERGING_STATEMENT Function

This function retrieves information about a diverging call, including the statement text, the SQL ID, and the binds. If the replay of a recorded user call has data or error divergence, it is a diverging call.

Syntax

```
DBMS_WORKLOAD_REPLAY.GET_DIVERGING_STATEMENT (  
    replay_id    IN NUMBER,  
    stream_id    IN NUMBER,  
    call_counter IN NUMBER)  
RETURN CLOB;
```

Parameters

Table 187-16  GET_DIVERGING_STATEMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id</td>
<td>ID of the replay in which that call diverged</td>
</tr>
<tr>
<td>stream_id</td>
<td>Stream ID of the diverging call</td>
</tr>
<tr>
<td>call_counter</td>
<td>Call counter of the diverging call</td>
</tr>
</tbody>
</table>

Usage Notes

- Returns a CLOB formatted as XML that contains:
  - SQL ID
  - SQL Text
  - Bind information: position, name and value
- This function will silently invoke the POPULATE_DIVERGENCE Procedure to read the information from the capture files. Therefore, if divergence has not been
populated, then the first call to this function for a particular diverging call might take longer, especially in very large captures.

187.3.17 GET_REPLAY_DIRECTORY Function

This function returns the current replay directory set by the SET_REPLAY_DIRECTORY Procedure. It returns NULL if no replay directory has been set.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.GET_REPLAY_DIRECTORY
RETURN VARCHAR2;
```

Related Topics

- **SET_REPLAY_DIRECTORY Procedure**
  This procedure sets a directory that contains multiple workload captures as the current replay directory.

187.3.18 GET_REPLAY_INFO Function

This function retrieves information about the workload capture and the history of all the workload replay attempts from the stipulated directory.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.GET_REPLAY_INFO (
    replay_dir    IN VARCHAR2,
    load_details  IN BOOLEAN DFAULT FALSE)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_dir</td>
<td>(Mandatory) Name of the workload replay directory object (case sensitive).</td>
</tr>
<tr>
<td>load_details</td>
<td>Load the divergence and tracked commits data. The default value is FALSE.</td>
</tr>
</tbody>
</table>

Return Values

The procedure returns the `CAPTURE_ID`, which can be associated with both `DBA_WORKLOAD_CAPTURES.ID` and `DBA_WORKLOAD_REPLAYS.CAPTURE_ID` to access the imported information.

Usage Notes

- The procedure first imports a row into `DBA_WORKLOAD_CAPTURES` which will contain information about the capture. It then imports a row for every replay attempt retrieved from the given replay directory into `DBA_WORKLOAD_REPLAYS`. 
The procedure will not insert new rows to \texttt{DBA\_WORKLOAD\_CAPTURES} and \texttt{DBA\_WORKLOAD\_REPLAYS} if these views already contain rows describing the capture and replay history present in the given directory.

### 187.3.19 GET\_REPLAY\_TIMEOUT Procedure

This procedure gets the replay timeout setting.

**Syntax**

\begin{verbatim}
DBMS\_WORKLOAD\_REPLAY\_GET\_REPLAY\_TIMEOUT (  
enabled       OUT  BOOLEAN,  
min_delay     OUT  NUMBER,  
max_delay     OUT  NUMBER,  
delay_factor  OUT  NUMBER);  
\end{verbatim}

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>TRUE if the timeout action is enabled, FALSE otherwise.</td>
</tr>
<tr>
<td>min_delay</td>
<td>Lower bound of call delay in minutes. The replay action is activated only when the delay is equal to or more than min_delay.</td>
</tr>
<tr>
<td>max_delay</td>
<td>Upper bound of call delay in minutes. The timeout action throws ORA-15569 when the delay is more than max_delay.</td>
</tr>
<tr>
<td>delay_factor</td>
<td>Factor for the call delay that is between min_delay and max_delay. The timeout action throws ORA-15569 when the current replay elapsed time is more than the product of capture elapsed time and delay_factor.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure can be called anytime during replay.

### 187.3.20 IMPORT\_AWR Function

This procedure imports the AWR snapshots from a given replay.

**Syntax**

\begin{verbatim}
DBMS\_WORKLOAD\_REPLAY\_IMPORT\_AWR (  
    replay_id       IN   NUMBER,  
    staging_schema  IN   VARCHAR2)  
RETURN NUMBER;  
\end{verbatim}

Parameters

Table 187-19  IMPORT_AWR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id</td>
<td>(Mandatory) ID of the replay whose AWR snapshots must be imported</td>
</tr>
<tr>
<td>staging_schema</td>
<td>(Mandatory) Name of a valid schema in the current database which can be used as a staging area while importing the AWR snapshots from the replay directory to the SYS AWR schema. The SYS schema is not a valid input.</td>
</tr>
</tbody>
</table>

Return Values

Returns the new randomly generated database ID that was used to import the AWR snapshots. The same value can be found in the AWR_DBID column in the DBA_WORKLOAD_REPLAYS view.

Usage Notes

- This procedure will work provided those AWR snapshots were exported earlier from the original replay system using the EXPORT_AWR Procedure.
- IMPORT_AWR will fail if the staging_schema provided as input contains any tables with the same name as any of the AWR tables, such as WRM$_SNAPSHOT or WRH$_PARAMETER. Drop any such tables in the staging_schema before invoking IMPORT_AWR.

187.3.21 INITIALIZE_CONSOLIDATED_REPLAY Procedure

This procedure puts the database state in INIT for a multiple-capture replay.

It uses the replay_dir which has already been defined by the SET_REPLAY_DIRECTORY Procedure, pointing to a directory that contains all the capture directories involved in the schedule. It reads data about schedule schedule_name from the directory, and loads required connection data into the replay system.

Syntax

```
DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY (replay_name IN VARCHAR2,
                                           schedule_name   IN VARCHAR2,
                                           plsql_mode      IN VARCHAR2 DEFAULT 'TOP_LEVEL');
```

Parameters

Table 187-20  INITIALIZE_CONSOLIDATED_REPLAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_name</td>
<td>(Mandatory) Name of the workload replay. Every replay of a processed workload capture can be given a name.</td>
</tr>
<tr>
<td>schedule_name</td>
<td>Name of the schedule to be replayed. It must have been created through the BEGIN_REPLAY_SCHEDULE Procedure for the replay directory replay_dir.</td>
</tr>
</tbody>
</table>
### Table 187-20  (Cont.) INITIALIZE_CONSOLIDATED_REPLAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plsql_mode</td>
<td>Specifies the replay options for PL/SQL calls:</td>
</tr>
<tr>
<td></td>
<td>• TOP_LEVEL — only top-level PL/SQL calls are replayed</td>
</tr>
<tr>
<td></td>
<td>• EXTENDED — SQL executed from PL/SQL or top-level SQL PL/SQL if there is no SQL recorded inside the PL/SQL are replayed. All captures must be done in 'EXTENDED' PL/SQL mode.</td>
</tr>
</tbody>
</table>

### Usage Notes

**Prerequisites:**

- Workload capture was already processed using the `PROCESS_CAPTURE Procedure` in the same database version.
- Database state has been logically restored to what it was at the beginning of the original workload capture.
- The `SET_REPLAY_DIRECTORY Procedure` has been called.

### 187.3.22 INITIALIZE_REPLAY Procedure

This procedure puts the database state in `INIR` for `REPLAY` mode, and loads data into the replay system that is required before preparing for the replay (by executing the `PAUSE_REPLAY Procedure`).

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY (
    replay_name IN VARCHAR2,
    replay_dir IN VARCHAR2,
    plsql_mode IN VARCHAR2 DEFAULT 'TOP_LEVEL',
    rac_inst_list IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

### Table 187-21  INITIALIZE_REPLAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_name</td>
<td>(Mandatory) Name of the workload replay. Every replay of a processed workload capture can be given a name.</td>
</tr>
<tr>
<td>replay_dir</td>
<td>Name of the directory object that points to the OS directory (case sensitive) that contains processed capture data</td>
</tr>
<tr>
<td>plsql_mode</td>
<td>Specifies the replay options for PL/SQL calls:</td>
</tr>
<tr>
<td></td>
<td>• TOP_LEVEL — only top-level PL/SQL calls are replayed</td>
</tr>
<tr>
<td></td>
<td>• EXTENDED — SQL executed from PL/SQL or top-level SQL PL/SQL if there is no SQL recorded inside the PL/SQL are replayed. All captures must be done in 'EXTENDED' PL/SQL mode.</td>
</tr>
</tbody>
</table>
### Table 187-21  (Cont.) INITIALIZE_REPLAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rac_inst_list</td>
<td>Specifies a list of Oracle Real Application Clusters (Oracle RAC) instances</td>
</tr>
<tr>
<td></td>
<td>that will be used for replay. The parameter is a string of instance numbers</td>
</tr>
<tr>
<td></td>
<td>that are separated by commas. For example:</td>
</tr>
<tr>
<td></td>
<td><code>rac_inst_list='1,3,5'</code></td>
</tr>
</tbody>
</table>

#### Usage Notes

- **Prerequisites:**
  - Workload capture was already processed using the `PROCESS_CAPTURE Procedure` in the same database version.
  - Database state has been logically restored to what it was at the beginning of the original workload capture.
- The subprogram loads data into the replay system that is required before preparing for the replay by calling the `PAUSE_REPLAY Procedure`. For instance, during capture the user may record the connection string each session used to connect to the server. The INITIALIZE_REPLAY Procedure loads this data and allows the user to re-map the recorded connection string to new connection strings or service points.

  Elaborating on the example described in the `PROCESS_CAPTURE Procedure`, the user could invoke the following:

  ```sql
  DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY('replay foo #1', 'rec_dir');
  ```

  This command will load up the connection map and by default will set all replay time connection strings to be equal to `NULL`. A `NULL` replay time connection string means that the workload replay clients (WRCs) will connect to the default host as determined by the replay client's runtime environment settings. The user can change a particular connection string to a new one (or a new service point) for replay by using the `REMAP_CONNECTION Procedure`.

- **For encrypted capture, the INITIALIZE_REPLAY Procedure relies on Oracle wallet. The identifier is `oracle.rat.database_replay.encryption` (case-sensitive).**

#### Related Topics

- **PAUSE_REPLAY Procedure**
  This procedure pauses the in-progress workload replay.

- **INITIALIZE_REPLAY Procedure**
  This procedure puts the database state in `INIT` for `REPLAY mode`, and loads data into the replay system that is required before preparing for the replay (by executing the `PAUSE_REPLAY Procedure`).

- **PROCESS_CAPTURE Procedure**
  This procedure processes the workload capture found in `capture_dir in place`.

- **REMAP_CONNECTION Procedure**
  This procedure remaps the captured connection to a new one so that the user sessions can connect to the database in a desired way during workload replay.
187.3.23 IS_REPLAY_PAUSED Function

This function reports whether the replay is currently paused.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.IS_REPLAY_PAUSED
   RETURN BOOLEAN;
```

Return Values

Returns `TRUE` if the `PAUSE_REPLAY Procedure` has been called successfully and the `RESUME_REPLAY Procedure` has not been called yet.

Usage Notes

A call to the `START_REPLAY Procedure` must have already been issued as a prerequisite.

187.3.24 LOAD_LONG_SQLTEXT Procedure

This procedure loads the captured SQL statements that are longer than 1000 characters to the `DBA_WORKLOAD_LONG_SQLTEXT` view.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.LOAD_LONG_SQLTEXT (capture_id IN NUMBER);
```

Parameters

Table 187-22  LOAD_LONG_SQLTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_id</td>
<td>Internal key for the workload capture</td>
</tr>
</tbody>
</table>

**Note:**

This procedure is available starting with Oracle Database Release 18c.

**See Also:**

`DBA_WORKLOAD_LONG_SQLTEXT` in *Oracle Database Reference*
187.3.25 PAUSE_REPLAY Procedure

This procedure pauses the in-progress workload replay.

All subsequent user calls from the replay clients will be stalled until either a call to the RESUME_REPLAY Procedure is issued or the replay is cancelled.

Syntax

DBMS_WORKLOAD_REPLAY.PAUSE_REPLAY;

Usage Notes

• Prerequisite: A call to the START_REPLAY Procedure must have already been issued.
• User calls that were already in-progress when this procedure was invoked are allowed to run to completion. Only subsequent user calls, when issued, are paused.

187.3.26 POPULATE_DIVERGENCE Procedure

This procedure precomputes the divergence information for the given call, stream, or the whole replay so that the GET_DIVERGING_STATEMENT Function returns as quickly as possible for the precomputed calls.

Syntax

DBMS_WORKLOAD_REPLAY.POPULATE_DIVERGENCE (
    replay_id    IN   NUMBER,
    stream_id    IN   NUMBER DEFAULT NULL,
    call_counter IN   NUMBER DEFAULT NULL);

Parameters

Table 187-23  POPULATE_DIVERGENCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id</td>
<td>ID of the replay</td>
</tr>
<tr>
<td>stream_id</td>
<td>Stream ID of the diverging call. If NULL is provided, then divergence information is precomputed for all diverging calls in the given replay.</td>
</tr>
<tr>
<td>call_counter</td>
<td>Call counter of the diverging call. If NULL is provided, then divergence information is precomputed for all diverging calls in the given stream.</td>
</tr>
</tbody>
</table>

Related Topics

• GET_DIVERGING_STATEMENT Function
  This function retrieves information about a diverging call, including the statement text, the SQL ID, and the binds. If the replay of a recorded user call has data or error divergence, it is a diverging call.
187.3.27 PREPARE_CONSOLIDATED_REPLAY Procedure

Similar to the PREPARE REPLAY Procedure, this procedure puts the database in a special "Prepare" mode for a multiple-capture replay. The difference is that this subprogram should be used only for consolidated replays.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY (
  synchronization           IN BOOLEAN,
  connect_time_scale        IN NUMBER    DEFAULT 100,
  think_time_scale          IN NUMBER    DEFAULT 100,
  think_time_auto_correct   IN BOOLEAN   DEFAULT TRUE,
  capture_sts               IN BOOLEAN   DEFAULT FALSE,
  sts_cap_interval          IN NUMBER    DEFAULT 300);
```

Parameters

Table 187-24  PREPARE_CONSOLIDATED_REPLAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronization</td>
<td>Sets the synchronization mode for replay:</td>
</tr>
<tr>
<td></td>
<td>• 'TIME' — The synchronization will be based on the time the action took place during capture (clock-based time).</td>
</tr>
<tr>
<td></td>
<td>• 'SCN' — The synchronization will be based on the capture-time commits; the commit order will be preserved during replay. This is the default mode.</td>
</tr>
<tr>
<td></td>
<td>• 'OBJECT_ID' — Every replayed action will be executed only after the relevant commits have finished execution. The relevant commits are those that were issued before the given action in the captured workload and that modified at least one of the database objects the given action is referencing (either implicitly or explicitly). This synchronization mode makes sure that any replay action will see the same data that the action saw during capture, but allows greater concurrency for the actions that do not touch the same objects/tables.</td>
</tr>
<tr>
<td></td>
<td>This value is deprecated. For compatibility, an overloaded version of this procedure uses BOOLEAN for this parameter:</td>
</tr>
<tr>
<td></td>
<td>• TRUE means 'SCN'</td>
</tr>
<tr>
<td></td>
<td>• FALSE means 'TIME'</td>
</tr>
</tbody>
</table>

For compatibility, an overloaded version of this procedure uses BOOLEAN for this parameter:

```sql
DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY (
  synchronization           IN VARCHAR2 DEFAULT 'SCN',
  connect_time_scale        IN NUMBER    DEFAULT 100,
  think_time_scale          IN NUMBER    DEFAULT 100,
  think_time_auto_correct   IN BOOLEAN   DEFAULT TRUE,
  capture_sts               IN BOOLEAN   DEFAULT FALSE,
  sts_cap_interval          IN NUMBER    DEFAULT 300);
```
Table 187-24 (Cont.) PREPARE_CONSOLIDATED_REPLAY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect_time_scale</td>
<td>Scales the time elapsed between the instant the workload capture was started and the session connects with the given value. The input is interpreted as a % value. Can potentially be used to increase or decrease the number of concurrent users during the workload replay. DEFAULT VALUE is 100. See &quot;Example 187-1&quot;.</td>
</tr>
<tr>
<td>think_time_scale</td>
<td>Scales the time elapsed between two successive user calls from the same session. The input is interpreted as a % value. Can potentially be used to increase or decrease the number of concurrent users during the workload replay. DEFAULT VALUE is 100. See &quot;Example 187-2&quot;.</td>
</tr>
<tr>
<td>think_time_auto_correct</td>
<td>Auto corrects the think time between calls appropriately when user calls takes longer to complete during replay than during the original capture. DEFAULT is TRUE which is to reduce think time if replay goes slower than capture. See &quot;Example 187-3&quot;.</td>
</tr>
<tr>
<td>capture_sts</td>
<td>If this parameter is TRUE, then a SQL tuning set capture is also started in parallel with workload replay. The resulting SQL tuning set can be exported using the EXPORT_AWR Procedure along with the Automatic Workload Repository (AWR) data. Currently, parallel SQL tuning set (STS) capture is not supported in an Oracle RAC environment. So, this parameter has no effect in that context. The calling user must have the appropriate privileges (ADMINISTER SQL TUNING SET). The default value is FALSE.</td>
</tr>
<tr>
<td>sts_cap_interval</td>
<td>Specifies the capture interval of the SQL set capture from the cursor cache in seconds. The default value is 300.</td>
</tr>
</tbody>
</table>

Usage Notes

A consolidated replay replays multiple captures in one replay. Each capture records different system change number (SCN) values. For this reason SCN-based sync is not supported for consolidated replays. Consolidated replays only support non-sync mode and the Object-ID based synchronization, and SCN-based synchronization is currently not supported.

Related Topics
- PREPARE_REPLAY Procedure

This procedure puts the database state in PREPARE FOR REPLAY mode.

187.3.28 PREPARE_REPLAY Procedure

This procedure puts the database state in PREPARE FOR REPLAY mode.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY (  
  synchronization IN BOOLEAN,  
  connect_time_scale IN NUMBER DEFAULT 100,  
  think_time_scale IN NUMBER DEFAULT 100,  
  think_time_auto_correct IN BOOLEAN DEFAULT TRUE,  
  scale_up_multiplier IN NUMBER DEFAULT 1,  
)  
```
Parameters

**Table 187-25  PREPARE_REPLAY Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronization</td>
<td>Sets the synchronization mode for replay:</td>
</tr>
<tr>
<td></td>
<td>• 'TIME' — The synchronization will be based on the time the action took place during capture (clock-based time).</td>
</tr>
<tr>
<td></td>
<td>• 'SCN' — The synchronization will be based on the capture-time commits; the commit order will be preserved during replay. This is the default mode.</td>
</tr>
<tr>
<td></td>
<td>• 'OBJECT_ID' — Every replayed action will be executed only after the relevant commits have finished execution. The relevant commits are those that were issued before the given action in the captured workload and that modified at least one of the database objects the given action is referencing (either implicitly or explicitly). This synchronization mode makes sure that any replay action will see the same data that the action saw during capture, but allows greater concurrency for the actions that do not touch the same objects/tables.</td>
</tr>
<tr>
<td></td>
<td>This value is deprecated.</td>
</tr>
<tr>
<td></td>
<td>For compatibility, an overloaded version of this procedure uses BOOLEAN for this parameter:</td>
</tr>
<tr>
<td></td>
<td>• TRUE means 'SCN'</td>
</tr>
<tr>
<td></td>
<td>• FALSE means 'TIME'</td>
</tr>
<tr>
<td>connect_time_scale</td>
<td>Scales the time elapsed between the instant the workload capture was started and the session connects with the given value. The input is interpreted as a % value. Can potentially be used to increase or decrease the number of concurrent users during the workload replay. DEFAULT VALUE is 100. See &quot;Example 187-1&quot;.</td>
</tr>
<tr>
<td>think_time_scale</td>
<td>Scales the time elapsed between two successive user calls from the same session. The input is interpreted as a % value. Can potentially be used to increase or decrease the number of concurrent users during the workload replay. DEFAULT VALUE is 100. See &quot;Example 187-2&quot;.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>think_time_auto_correct</td>
<td>Auto corrects the think time between calls appropriately when a user call takes longer to complete during replay than during the original capture. DEFAULT is TRUE which is to reduce think time if replay goes slower than capture. See &quot;Example 187-3&quot;.</td>
</tr>
<tr>
<td>scale_up_multiplier</td>
<td>Defines the number of times the query workload is scaled up during replay. Each captured session is replayed concurrently as many times as the value of the scale_up_multiplier. However, only one of the sessions in each set of identical replay sessions executes both queries and updates. The remaining sessions only execute queries.</td>
</tr>
<tr>
<td>capture_sts</td>
<td>If this parameter is TRUE, then a SQL tuning set capture is also started in parallel with workload replay. The resulting SQL tuning set can be exported using the EXPORT_AWR Procedure along with the AWR data. Currently, parallel SQL tuning set (STS) capture is not supported in an Oracle RAC environment. So, this parameter has no effect in that context. The calling user must have the appropriate privileges ('ADMINISTER SQL TUNING SET'). The default value is FALSE.</td>
</tr>
<tr>
<td>sts_cap_interval</td>
<td>Specifies the capture interval of the SQL set capture from the cursor cache in seconds. The default value is 300.</td>
</tr>
<tr>
<td>rac_mode</td>
<td>Specifies replay options in an Oracle Real Application Cluster (Oracle RAC) environment. This parameter accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>• GLOBAL_SYNC — Synchronization across all Oracle RAC instances. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>• PER_INSTANCE_CLIENT — Synchronization is global and each WRC client handles part of the workload that is sent to only one instance.</td>
</tr>
<tr>
<td></td>
<td>• PER_INSTANCE_SYNC — Synchronization is local (within each instance only) and each WRC client handles part of the workload that is sent to only one instance.</td>
</tr>
<tr>
<td>query_only</td>
<td>Replays only the read-only queries of the workload capture. The default value is FALSE.</td>
</tr>
</tbody>
</table>

### Usage Notes

- **Prerequisites:**
  - The database has been initialized for replay using the INITIALIZE_REPLAY Procedure.
  - Any capture time connection strings that require remapping have been already done using the REMAP_CONNECTION Procedure.
- One or more external replay clients (WRC) can be started once the PREPARE_REPLAY procedure has been executed.
- With regard to scale_up_multiplier:
  - One replay session (base session) of each set of identical sessions will replay every call from the capture as usual.
The remaining sessions (scale-up sessions) will only replay calls that are read-only. Thus, DDL, DML, and PL/SQL calls that modified the database is skipped. SELECT FOR UPDATE statements are also skipped.

Read-only calls from the scale-up are synchronized appropriately and obey the timings defined by think_time_scale, connect_time_scale, and think_time_auto_correct. Also, the queries are made to wait for the appropriate commits.

No replay data or error divergence records are generated for the scale-up sessions.

All base or scale-up sessions that replay the same capture file will connect from the same workload replay client.

Example 187-1  Application of the connect_time_scale Parameter

If the following was observed during the original workload capture:

12:00 : Capture was started
12:10 : First session connect (10m after)
12:30 : Second session connect (30m after)
12:42 : Third session connect (42m after)

If the connect_time_scale is 50, then the session connects will happen as follows:

12:00 : Replay was started with 50% connect time scale
12:05 : First session connect ( 5m after)
12:15 : Second session connect (15m after)
12:21 : Third session connect (21m after)

If the connect_time_scale is 200, then the session connects will happen as follows:

12:00 : Replay was started with 200% connect time scale
12:20 : First session connect (20m after)
13:00 : Second session connect (60m after)
13:24 : Third session connect (84m after)

Example 187-2  Application of the think_time_scale Parameter

If the following was observed during the original workload capture:

12:00 : User SCOTT connects
12:10 : First user call issued (10m after completion of prevcall)
12:14 : First user call completes in 4mins
12:30 : Second user call issued (16m after completion of prevcall)
12:40 : Second user call completes in 10m
12:42 : Third user call issued ( 2m after completion of prevcall)
12:50 : Third user call completes in 8m

If the think_time_scale is 50 during the workload replay, then the user calls will look something like below:

12:00 : User SCOTT connects
12:05 : First user call issued 5 mins (50% of 10m) after the completion of previous call
12:10 : First user call completes in 5m (takes a minute longer)
12:18 : Second user call issued 8 mins (50% of 16m) after the completion of prev call
12:25 : Second user call completes in 7m (takes 3 minutes less)
12:26 : Third user call issued 1 min (50% of 2m) after the completion of prev
call
12:35 : Third user call completes in 9m (takes a minute longer)

**Example 187-3 Application of the think_time_auto_correct Parameter**

If the following was observed during the original workload capture:

12:00 : User SCOTT connects
12:10 : First user call issued (10m after completion of prev call)
12:14 : First user call completes in 4m
12:30 : Second user call issued (16m after completion of prev call)
12:40 : Second user call completes in 10m
12:42 : Third user call issued (2m after completion of prev call)
12:50 : Third user call completes in 8m

If the think_time_scale is 100 and the think_time_auto_correct is TRUE during the workload replay, then the user calls will look something like below:

12:00 : User SCOTT connects
12:10 : First user call issued 10 mins after the completion of prev call
12:15 : First user call completes in 5m (takes 1 minute longer)
12:30 : Second user call issued 15 mins (16m minus the extra time of 1m the prev call took) after the completion of prev call
12:44 : Second user call completes in 14m (takes 4 minutes longer)
12:44 : Third user call issued immediately (2m minus the extra time of 4m the prev call took) after the completion of prev call
12:52 : Third user call completes in 8m

### 187.3.29 PROCESS_CAPTURE Procedure

This procedure processes the workload capture found in `capture_dir` in place.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE (  
capture_dir            IN   VARCHAR2,  
parallel_level         IN   NUMBER DEFAULT NULL,  
synchronization        IN   VARCHAR2 DEFAULT 'SCN',  
plsql_mode             IN   VARCHAR2 DEFAULT 'TOP_LEVEL');
```

**Parameters**

**Table 187-26 PROCESS_CAPTURE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_dir</td>
<td>(Mandatory) Name of the workload capture directory object (case sensitive). The directory object must point to a valid OS directory that has the appropriate permissions. New files are added to this directory.</td>
</tr>
<tr>
<td>parallel_level</td>
<td>Number of Oracle processes used to process the capture in parallel. The NULL default value will auto-compute the parallelism level, whereas a value of 1 will enforce serial execution.</td>
</tr>
</tbody>
</table>
Table 187-26  (Cont.) PROCESS_CAPTURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronization</td>
<td>Determines the synchronization mode that the user will be able to use for replay:</td>
</tr>
<tr>
<td></td>
<td>· 'TIME' — When 'TIME' is selected, the replay can use 'TIME' synchronization mode only.</td>
</tr>
<tr>
<td></td>
<td>When 'TIME' synchronization mode is used for replay, the synchronization will be based on the time the action took place during capture (clock-based time).</td>
</tr>
<tr>
<td></td>
<td>· 'SCN' — When 'SCN' is selected, the replay can use the 'TIME' or 'SCN' synchronization mode. This is the default.</td>
</tr>
<tr>
<td></td>
<td>When 'SCN' synchronization mode is used for replay, the synchronization will be based on the capture-time commits; the commit order will be preserved during replay. This is the default mode.</td>
</tr>
<tr>
<td></td>
<td>· 'OBJECT_ID' — When 'OBJECT_ID' is selected, replay can use the 'TIME', 'SCN', or 'OBJECT_ID' synchronization mode.</td>
</tr>
<tr>
<td></td>
<td>When 'OBJECT_ID' synchronization mode is used for replay, every replayed action will be executed only after the relevant commits have finished execution. These commits are those that were issued before the given action in the captured workload and that modified at least one of the database objects the given action is referencing (either implicitly or explicitly).</td>
</tr>
<tr>
<td></td>
<td>This synchronization mode makes sure that any replay action will see the same data that the action saw during capture, but allows greater concurrency for the actions that do not touch the same objects/tables.</td>
</tr>
<tr>
<td></td>
<td>This synchronization mode is deprecated.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>plsql_mode</th>
<th>Specifies the processing mode for PL/SQL:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>· 'TOP_LEVEL' — metadata is generated for top-level PL/SQL calls only; 'TOP_LEVEL' will be the only option for replay.</td>
</tr>
<tr>
<td></td>
<td>· 'EXTENDED' — metadata is generated for both top-level PL/SQL calls and the SQL called from PL/SQL. A new directoryppe_X.X.X.X (where the Xs represent the current Oracle version) is created under the capture root directory. Capture must have been done with this same value for the plsql_mode parameter. Replay can use either 'TOP_LEVEL' or 'EXTENDED'.</td>
</tr>
</tbody>
</table>

Usage Notes

- This subprogram analyzes the workload capture found in the capture_dir and creates new workload replay specific metadata files that are required to replay the given workload capture. It only creates new files and does not modify any files that were originally created during the workload capture. Therefore, this procedure can be run multiple times on the same capture directory, such as when the procedure encounters unexpected errors or is cancelled by the user.

- Once this procedure runs successfully, the capture_dir can be used as input to the INITIALIZE_REPLAY Procedure in order to replay the captured workload present in capture_dir.
• Before a workload capture can be replayed in a particular database version, the capture must be processed using `PROCESS_CAPTURE` in the same database version. Once created, a processed workload capture can be used to replay the captured workload multiple times in the same database version.

For example, suppose workload "foo" was captured in `rec_dir` in Oracle database version 10.2.0.5. In order to replay the workload "foo" in version 11.1.0.1 the workload must be processed in version 11.1.0.1. The following procedure must be executed in an 11.1.0.1 database in order to process the capture directory `rec_dir`:

```sql
DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE('rec_dir');
```

Now, `rec_dir` contains a valid 11.1.0.1 processed workload capture that can be used to replay the workload "foo" in 11.1.0.1 databases as many times as required.

• For encrypted capture, the `PROCESS_CAPTURE` procedure relies on Oracle wallet. The identifier is `oracle.rat.database_replay.encryption` (case-sensitive).

### 187.3.30 REMAP_CONNECTION Procedure

This procedure remaps the captured connection to a new one so that the user sessions can connect to the database in a desired way during workload replay.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (  
  connection_id         IN  NUMBER,  
  replay_connection     IN  VARCHAR2);
```

```sql
DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (  
  capture_number        IN  VARCHAR2,  
  connection_id         IN  NUMBER,  
  replay_connection     IN  VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_number</td>
<td>Pointing to a capture of the current replay schedule</td>
</tr>
<tr>
<td>connection_id</td>
<td>ID of the connection to be remapped. Corresponds to <code>DBA_WORKLOAD_CONNECTION_MAP.CONN_ID</code>.</td>
</tr>
<tr>
<td>replay_connection</td>
<td>New connection string to be used during replay</td>
</tr>
</tbody>
</table>

**Usage Notes**

• Prior to calling `REMAP_CONNECTION` all replay connection strings are set to `NULL` by default. If a `replay_connection` is `NULL`, then the replay sessions will connect as determined by the replay client's runtime environment. For example, if the environment variable `TNS_ADMIN` is defined and the user does not call the `REMAP_CONNECTION Procedure`, then the `wrc` executable will connect to the server specified in the `tnsnames.ora` file pointed to by `TNS_ADMIN`.

• A valid `replay_connection` must specify a connect identifier or a service point. See the `Oracle Database Net Services Reference` for ways to specify connect identifiers (such as...
net service names, database service names, and net service aliases) and naming methods that can be used to resolve a connect identifier to a connect descriptor.

- An error is returned if no row matches the given `connection_id`.
- Use the `DBA_WORKLOAD_CONNECTION_MAP` view to review all the connection strings that are used by the subsequent workload replay, and also to examine connection string remappings used for previous workload replays.

### 187.3.31 REMOVE_CAPTURE Procedure

This procedure removes the given capture from the current schedule.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.REMOVE_CAPTURE (
    schedule_capture_number    IN     NUMBER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_capture_number</td>
<td>Unique ID that identifies this capture within this schedule</td>
</tr>
</tbody>
</table>

### 187.3.32 REMOVE_SCHEDULE_ORDERING Procedure

This procedure removes an existing schedule order from the current replay schedule.

Together, `schedule_capture_id` and `waitfor_capture_id` form a schedule ordering that previously added by the `ADD_SCHEDULE_ORDERING Function` (`schedule_capture_id`, `waitfor_capture_id`). The order is that replay of capture indicated by `schedule_capture_id` will not start unless the replay of capture indicated by `waitfor_capture_id` finishes.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.REMOVE_SCHEDULE_ORDERING (
    schedule_capture_id     IN         NUMBER,
    waitfor_capture_id      IN         NUMBER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_capture_id</td>
<td>Points to a capture that has been added to the current replay schedule (see procedure description).</td>
</tr>
<tr>
<td>waitfor_capture_id</td>
<td>Points to a capture that has been added to the current replay schedule.</td>
</tr>
</tbody>
</table>
• The `BEGIN_REPLAY_SCHEDULE Procedure` must have been called.
• The replay schedule order should have already been added using the `ADD_SCHEDULE_ORDERING Function`.

187.3.33 REPORT Function

This function generates a report on the stipulated workload replay.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.REPORT (
    replay_id      IN NUMBER,
    format         IN VARCHAR2)
RETURN CLOB;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_id</td>
<td>(Mandatory) Specifies the ID of the workload replay whose report is requested.</td>
</tr>
<tr>
<td>format</td>
<td>(Mandatory) Specifies the report format. Valid values:</td>
</tr>
<tr>
<td></td>
<td>• HTML - Generates the HTML version of the report</td>
</tr>
<tr>
<td></td>
<td>• XML - Generates the XML version of the report</td>
</tr>
<tr>
<td></td>
<td>• TEXT - Generates the text version of the report</td>
</tr>
</tbody>
</table>

**Return Values**

The report body in the desired format returned as a CLOB.

187.3.34 RESUME_REPLAY Procedure

This procedure resumes a paused workload replay.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.RESUME_REPLAY;
```

**Usage Notes**

Prerequisite: A call to the `PAUSE_REPLAY Procedure` must have already been issued.

187.3.35 REUSE_REPLAY_FILTER_SET Procedure

This procedure reuses filters in the specified filter set as if each were added using the `ADD_SCHEDULE_ORDERING Function`.

Each call adds one filter set, which is a collection of individual filters on various attributes. Also, a new filter rule can be added, and an existing filter can be deleted before invoking the `CREATE_FILTER_SET Procedure` to create a new filter set.
Syntax

```
DBMS_WORKLOAD_REPLAY.REUSE_REPLAY_FILTER_SET(
    replay_dir  IN VARCHAR2,
    filter_set  IN VARCHAR2);
```

Parameters

Table 187-31  REUSE_REPLAY_FILTER_SET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_dir</td>
<td>Capture ID of the existing filter set with which it is associated</td>
</tr>
<tr>
<td>filter_set</td>
<td>Name of the filter set to be reused</td>
</tr>
</tbody>
</table>

Related Topics

- **ADD_SCHEDULE_ORDERING Function**
  This function adds a schedule order between two captures.

187.3.36 SET_ADVANCED_PARAMETER Procedure

This procedure sets an advanced parameter for replay besides the ones used with the PREPARE_REPLAY Procedure.

The advanced parameters control aspects of the replay that are more specialized. The advanced parameters are reset to their default values after the replay has finished.

Syntax

```
DBMS_WORKLOAD_REPLAY.SET_ADVANCED_PARAMETER(
    pname    IN   VARCHAR2,
    pvalue   IN   VARCHAR2);
```

```
DBMS_WORKLOAD_REPLAY.SET_ADVANCED_PARAMETER(
    pname    IN   VARCHAR2,
    pvalue   IN   NUMBER);
```

```
DBMS_WORKLOAD_REPLAY.SET_ADVANCED_PARAMETER(
    pname    IN   VARCHAR2,
    pvalue   IN   BOOLEAN);
```

Parameters

Table 187-32  SET_ADVANCED_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>Name of the parameter (case insensitive)</td>
</tr>
<tr>
<td>pvalue</td>
<td>Value of the parameter</td>
</tr>
</tbody>
</table>

Usage Notes

The current parameters and values that can be used are:

'DO_NO_WAIT_COMMITS': (default: FALSE)
This parameter controls whether the COMMIT issued by replay sessions is NOWAIT. The default value for this parameter is FALSE. In this case all the COMMITS are issued with the mode they were captured (wait, no-wait, batch, no-batch). If the parameter is set to TRUE, then all COMMITS are issued in no-wait mode. This is useful in cases where the replay is becoming noticeably slow because of a high volume of concurrent COMMITS. Setting the parameter to TRUE will significantly decrease the waits on the 'log file sync' event during the replay with respect to capture.

Related Topics

• PREPARE_REPLAY Procedure
  This procedure puts the database state in PREPARE FOR REPLAY mode.

187.3.37 SET_REPLAY_DIRECTORY Procedure

This procedure sets a directory that contains multiple workload captures as the current replay directory.

Syntax

```
DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY (    
   replay_dir   IN   VARCHAR2);
```

Parameters

Table 187-33  SET_REPLAY_DIRECTORY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay_dir</td>
<td>Name of the OS directory containing the captures for a workload consolidation</td>
</tr>
</tbody>
</table>

187.3.38 SET_REPLAY_TIMEOUT Procedure

This procedure sets the replay timeout setting. The purpose is to abort user calls that might make the replay much slower or even cause a replay hang.

Syntax

```
DBMS_WORKLOAD_REPLAY.SET_REPLAY_TIMEOUT (    
   enabled       OUT  BOOLEAN DEFAULT TRUE,    
   min_delay     OUT  NUMBER DEFAULT 10,       
   max_delay     OUT  NUMBER DEFAULT 120,       
   delay_factor  OUT  NUMBER DEFAULT 8);        
```

Parameters

Table 187-34  SET_REPLAY_TIMEOUT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>TRUE to enable the timeout action, and FALSE to disable.</td>
</tr>
<tr>
<td>min_delay</td>
<td>Lower bound of call delay in minutes. The replay action is activated only when the delay is equal to or more than min_delay. Default = 10.</td>
</tr>
</tbody>
</table>
### Table 187-34  (Cont.) SET_REPLAY_TIMEOUT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_delay</td>
<td>Upper bound of call delay in minutes. The timeout action throws ORA-15569 when the delay is more than max_delay. Default = 120.</td>
</tr>
<tr>
<td>delay_factor</td>
<td>Factor for the call delay that is between min_delay and max_delay. The timeout action throws ORA-15569 when the current replay elapsed time is more than the product of capture elapsed time and delay_factor. Default = 8.</td>
</tr>
</tbody>
</table>

### Usage Notes

- This procedure can be called anytime during replay.
- Call delay is defined as the difference between replay and capture if replay elapsed time is longer than call elapsed time.
- Once a replay timeout action is enabled, a user call will exit with ORA-15569 if it has been delayed more than the condition specified by the replay action. The call and its error are reported as error divergence.
- Replay timeout operates as follows:
  - The timeout action has no effect if it is not enabled.
  - If the call delay in minutes is less than a lower bound specified by parameter min_delay, then the timeout action is non-operational.
  - If the delay in minutes is more than a upper bound specified by parameter max_delay, the timeout action will abort the user call and throw ORA-15569.
  - For delay that is between the lower bound and upper bound, the user call will abort with ORA-15569 only when the current replay elapsed time is more than the product of capture elapsed time and parameter delay_factor.

### 187.3.39 SET_SQL_MAPPING Procedure

This procedure specifies SQL statements to be skipped or replaced during a database replay operation.

#### Syntax

```sql
PROCEDURE SET_SQL_MAPPING (  
schedule_cap_id      IN NUMBER,  
sql_id               IN VARCHAR2,  
operation            IN VARCHAR2,  
replacement_sql_text IN VARCHAR2 DEFAULT NULL);  

PROCEDURE SET_SQL_MAPPING (  
sql_id               IN VARCHAR2,  
operation            IN VARCHAR2,  
replacement_sql_text IN VARCHAR2 DEFAULT NULL);  
```
### Parameters

#### Table 187-35  SET_SQL_MAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_cap_id</td>
<td>ID of a capture in the schedule</td>
</tr>
<tr>
<td>sql_id</td>
<td>SQL identifier of the SQL statement at the time of capture</td>
</tr>
<tr>
<td>operation</td>
<td>Directs that one of the following actions be performed for the specified statement during database replay:</td>
</tr>
<tr>
<td></td>
<td>· ‘SKIP’ — Skip the SQL statement identified by sql_id</td>
</tr>
<tr>
<td></td>
<td>· ‘REPLACE’ — Replace this SQL statement identified by sql_id with the SQL statement in the replacement_sql_text parameter</td>
</tr>
</tbody>
</table>

**Usage Notes**

- replacement_sql_text: When ‘SKIP’ is specified for the operation parameter, this parameter is NULL. When ‘REPLACE’ is specified for the operation parameter, this parameter’s value is the SQL statement to be used.
- schedule_cap_id is used for consolidated replay.

### 187.3.40 SET_USER_MAPPING Procedure

This procedure sets a new schema or user name to be used during replay instead of the captured user.

#### Syntax

```sql
DBMS_WORKLOAD_REPLAY.SET_USER_MAPPING (  
schedule_cap_id      IN NUMBER,  
capture_user         IN VARCHAR2,  
replay_user          IN VARCHAR2);  
```

```sql
DBMS_WORKLOAD_REPLAY.SET_USER_MAPPING (  
capture_user         IN VARCHAR2,  
replay_user          IN VARCHAR2);  
```

**Parameters**

#### Table 187-36  SET_USER_MAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_cap_id</td>
<td>ID of the a capture in the schedule</td>
</tr>
<tr>
<td>capture_user</td>
<td>User name during the time of the workload capture</td>
</tr>
<tr>
<td>replay_user</td>
<td>User name to which captured user is remapped during replay.</td>
</tr>
</tbody>
</table>

**Usage Notes**

- A schedule_cap_id of NULL is used for regular non-consolidate replay.
- The replay must be initialized but not prepared in order to use this subprogram.
• If `replay_user` is set to `NULL`, then the mapping is disabled.
• After multiple calls with the same `capture_user`, the last call always takes effect.
• To list all the mappings that will be in effect during the subsequent replay execute the following:
  ```sql
  SELECT * FROM DBA_WORKLOAD_ACTIVE_USER_MAP
  ```
• The overloaded version without the `schedule_cap_id` calls the one with the `schedule_cap_id` argument by passing in `NULL`.
• Mappings are stored in a table made public through the view `DBA_WORKLOAD_USER_MAP`. To remove old mappings execute
  ```sql
  DELETE * FROM DBA_WORKLOAD_USER_MAP
  ```

### 187.3.41 START_CONSOLIDATED_REPLAY Procedure

This procedure starts the replay of a multiple-capture capture. It should be used only for consolidated replays.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY;
```

**Usage Notes**

Prerequisites:

• The call to the `PREPARE_REPLAY Procedure` was already issued.
• A sufficient number of external replay clients (WRC) that can faithfully replay the captured workload already started. The status of such external replay clients can be monitored using `V$WORKLOAD_REPLAY_CLIENTS`.

### 187.3.42 START_REPLAY Procedure

This procedure starts the workload replay.

All the external replay clients (WRC) that are currently connected to the replay database will automatically be notified, and those replay clients (WRC) will begin issuing the captured workload. It should only be used for consolidated replays.

**Syntax**

```sql
DBMS_WORKLOAD_REPLAY.START_REPLAY;
```

**Usage Notes**

• Prerequisites:
  – The call to the `PREPARE_REPLAY Procedure` was already issued.
  – A sufficient number of external replay clients (WRC) that can faithfully replay the captured workload already started. The status of such external replay clients can be monitored using `V$WORKLOAD_REPLAY_CLIENTS`.
• Use the WRC's `CALIBRATE` mode to determine the number of replay clients that might be required to faithfully replay the captured workload. For example:
187.3.43 USE_FILTER_SET Procedure

This procedure applies a filter set to a capture in the current replay schedule.

The filter set must have been created by calling the CREATE_FILTER_SET Procedure.

Syntax

```sql
DBMS_WORKLOAD_REPLAY.USE_FILTER_SET(
    capture_number   IN    VARCHAR2,
    filter_set       IN    VARCHAR2);
```

```sql
DBMS_WORKLOAD_REPLAY.USE_FILTER_SET(
    filter_set       IN    VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_number</td>
<td>Pointing to a capture of the current replay schedule</td>
</tr>
<tr>
<td>filter_set</td>
<td>Name of the filter set</td>
</tr>
</tbody>
</table>

Usage Notes

The filter set must have been created by calling the CREATE_FILTER_SET Procedure.
The `DBMS_WORKLOAD_REPOSITORY` package lets you manage the Automatic Workload Repository (AWR) by performing operations, such as, managing snapshots and baselines.

The chapter contains the following topics:

- Examples
- Data Structures
- Summary of `DBMS_WORKLOAD_REPOSITORY` Subprograms

See Also:

*Oracle Database Performance Tuning Guide* for more information about the Automatic Workload Repository

### 188.1 DBMS_WORKLOAD_REPOSITORY Examples

This example shows how to generate an AWR text report with the `DBMS_WORKLOAD_REPOSITORY` package for database ID 1557521192, instance ID 1, snapshot IDs 5390 and 5391, and with default options.

```sql
-- make sure to set line size appropriately
-- set linesize 152
SELECT output FROM TABLE(
    DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
        1557521192, 1, 5390, 5391)
) ;
```

You can call the `DBMS_WORKLOAD_REPOSITORY` packaged functions directly as in the example, but Oracle recommends you use the corresponding supplied SQL script (`awrrpt.sql` in this case) for the packaged function, which prompts the user for required information.

### 188.2 DBMS_WORKLOAD_REPOSITORY Data Structures

The `DBMS_WORKLOAD_REPOSITORY` package defines an object and associated table types.

**OBJECT Types**

- `AWR_BASELINE_METRIC_TYPE` Object Type

**TABLE Types**

- `AWR_BASELINE_METRIC_TYPE_TABLE` Table Type
- `AWRRPT_INSTANCE_LIST_TYPE` Table Type
188.2.1 DBMS_WORKLOAD_REPOSITORY
AWR_BASELINE_METRIC_TYPE Object Type

This type shows the values of the metrics corresponding to a baseline.

Syntax

```
TYPE awr_baseline_metric_type AS OBJECT (
  baseline_name      VARCHAR2(64),
  dbid               NUMBER NOT NULL,
  instance_number    NUMBER NOT NULL,
  beg_time           DATE NOT NULL,
  end_time           DATE NOT NULL,
  metric_id          NUMBER NOT NULL,
  metric_name        VARCHAR2(64) NOT NULL,
  metric_unit        VARCHAR2(64) NOT NULL,
  num_interval       NUMBER NOT NULL,
  interval_size      NUMBER NOT NULL,
  average            NUMBER NOT NULL,
  minimum            NUMBER NOT NULL,
  maximum            NUMBER NOT NULL);
```

Fields

Table 188-1  AWR_BASELINE_METRIC_TYPE Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline_name</td>
<td>Name of the Baseline</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID for the snapshot</td>
</tr>
<tr>
<td>instance_number</td>
<td>Instance number for the snapshot</td>
</tr>
<tr>
<td>beg_time</td>
<td>Begin time of the interval</td>
</tr>
<tr>
<td>end_time</td>
<td>End time of the interval</td>
</tr>
<tr>
<td>metric_id</td>
<td>Metric ID</td>
</tr>
<tr>
<td>metric_name</td>
<td>Metric name</td>
</tr>
<tr>
<td>metric_unit</td>
<td>Unit of measurement</td>
</tr>
<tr>
<td>num_interval</td>
<td>Number of intervals observed</td>
</tr>
<tr>
<td>interval_size</td>
<td>Interval size (in hundredths of a second)</td>
</tr>
<tr>
<td>average</td>
<td>Average over the period</td>
</tr>
<tr>
<td>minimum</td>
<td>Minimum value observed</td>
</tr>
<tr>
<td>maximum</td>
<td>Maximum value observed</td>
</tr>
</tbody>
</table>
188.2.2 DBMS_WORKLOAD_REPOSITORY
AWR_BASELINE_METRIC_TYPE_TABLE Table Type

This type is used by the SELECT_BASELINE_METRIC Function.

Syntax

CREATE TYPE awr_baseline_metric_type_table AS TABLE OF awr_baseline_metric_type;

Related Topics

• SELECT_BASELINE_METRIC Function
  This table function shows the values of the metrics corresponding to a baseline for all the snapshots.

188.2.3 DBMS_WORKLOAD_REPOSITORY
AWRRPT_INSTANCE_LIST_TYPE Table Type

This type provides an alternative to a comma-separated list.

Syntax

CREATE TYPE awrrpt_instance_list_type AS TABLE OF NUMBER;

188.3 Summary of DBMS_WORKLOAD_REPOSITORY Subprograms

This table lists the DBMS_WORKLOAD_REPOSITORY subprograms and briefly describes them.

Table 188-2 DBMS_WORKLOAD_REPOSITORY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_COLORED_SQL Procedure</td>
<td>Adds a colored SQL ID</td>
</tr>
<tr>
<td>ASH_GLOBAL_REPORT_HTML Function</td>
<td>Displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in HTML format.</td>
</tr>
<tr>
<td>ASH_GLOBAL_REPORT_TEXT Function</td>
<td>Displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in Text format.</td>
</tr>
<tr>
<td>ASH_REPORT_ANALYTICS Function</td>
<td>Displays the ASH Analytics active report</td>
</tr>
<tr>
<td>ASH_REPORT_HTML Function</td>
<td>Displays the ASH report in HTML</td>
</tr>
<tr>
<td>ASH_REPORT_TEXT Function</td>
<td>Displays the ASH report in text</td>
</tr>
<tr>
<td>AWR_DIFF_REPORT_HTML Function</td>
<td>Displays the AWR Diff-Diff report in HTML</td>
</tr>
<tr>
<td>AWR_DIFF_REPORT_TEXT Function</td>
<td>Displays the AWR Diff-Diff report in text</td>
</tr>
<tr>
<td>AWR_GLOBAL_DIFF_REPORT_HTML Functions</td>
<td>Displays the Global AWR Compare Periods Report in HTML</td>
</tr>
</tbody>
</table>
### Table 188-2  (Cont.) DBMS_WORKLOAD_REPOSITORY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWR_GLOBAL_DIFF_REPORT_TEXT Functions</td>
<td>Displays the Global AWR Compare Periods Report in text</td>
</tr>
<tr>
<td>AWR_GLOBAL_REPORT_HTML Functions</td>
<td>Displays the Global AWR report in HTML</td>
</tr>
<tr>
<td>AWR_GLOBAL_REPORT_TEXT Functions</td>
<td>Displays the Global AWR report in text</td>
</tr>
<tr>
<td>AWR_REPORT_HTML Function</td>
<td>Displays the AWR report in HTML</td>
</tr>
<tr>
<td>AWR_REPORT_TEXT Function</td>
<td>Displays the AWR report in text</td>
</tr>
<tr>
<td>AWR_SET_REPORT_THRESHOLD S Procedure</td>
<td>Configures specified report thresholds, including the number of rows in the report</td>
</tr>
<tr>
<td>AWR_SQL_REPORT_HTML Function</td>
<td>Displays the AWR SQL Report in HTML format</td>
</tr>
<tr>
<td>AWR_SQL_REPORT_TEXT Function</td>
<td>Displays the AWR SQL Report in text format</td>
</tr>
<tr>
<td>CONTROL_RESTRICTED_SNAPSHOT Procedure</td>
<td>Controls the AWR snapshot creation for a database in the restricted session mode.</td>
</tr>
<tr>
<td>CREATE_BASELINE Functions &amp; Procedures</td>
<td>Creates a single baseline</td>
</tr>
<tr>
<td>CREATE_BASELINE_TEMPLATE Procedures</td>
<td>Creates a baseline template</td>
</tr>
<tr>
<td>CREATE_REMOTE_SNAPSHOT Function and Procedure</td>
<td>Creates a manual remote snapshot immediately using the Remote Management Framework (RMF)</td>
</tr>
<tr>
<td>CREATE_SNAPSHOT Function and Procedure</td>
<td>Creates a manual local snapshot immediately</td>
</tr>
<tr>
<td>DROP_BASELINE Procedure</td>
<td>Drops a previously-defined baseline</td>
</tr>
<tr>
<td>DROP_BASELINE_TEMPLATE Procedure</td>
<td>Removes a baseline template that is no longer needed</td>
</tr>
<tr>
<td>DROP_SNAPSHOT_RANGE Procedure</td>
<td>Drops a range of snapshots</td>
</tr>
<tr>
<td>LOCAL_AWR_DBID Function</td>
<td>Returns the database identifier for the local AWR database</td>
</tr>
<tr>
<td>MODIFY_BASELINE_WINDOW_SIZE Procedure</td>
<td>Modifies the window size for the Default Moving Window Baseline</td>
</tr>
<tr>
<td>MODIFY_SNAPSHOT_SETTINGS Procedures</td>
<td>Modifies the snapshot settings</td>
</tr>
<tr>
<td>PURGE_SQL_DETAILS Procedure</td>
<td>Purges SQL details, specifically rows from WRH$$_SQLTEXT and WRH$$_SQL_PLAN that do not have corresponding rows (DBID, SQL_ID) in WRH$$_SQLSTAT.</td>
</tr>
<tr>
<td>REGISTER_REMOTE_DATABASE Procedures</td>
<td>Registers a remote database in the AWR using the Remote Management Framework (RMF)</td>
</tr>
<tr>
<td>REMOVE_COLORED_SQL Procedure</td>
<td>Removes a colored SQL ID</td>
</tr>
<tr>
<td>RENAME_BASELINE Procedure</td>
<td>Renames a baseline</td>
</tr>
<tr>
<td>SELECT_BASELINE_DETAILS Function</td>
<td>Shows the values of the metrics corresponding to a baseline for a range of snapshots</td>
</tr>
</tbody>
</table>
Table 188-2 (Cont.) DBMS_WORKLOAD_REPOSITORY Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT_BASELINE_METRIC Function</td>
<td>Shows the values of the metrics corresponding to a baseline for all the snapshots</td>
</tr>
<tr>
<td>UNREGISTER_REMOTE_DATABASE Procedures</td>
<td>Removes all the statistics, metadata, and partitions of a remote database from the AWR using the Remote Management Framework (RMF)</td>
</tr>
<tr>
<td>UPDATE_DATAFILE_INFO Procedure</td>
<td>Updates the data file and tablespace information stored in the AWR with the current information in the database</td>
</tr>
<tr>
<td>UPDATE_OBJECT_INFO Procedure</td>
<td>Updates rows of WRH$SEG_STAT_OBJ table that represent objects in the local database</td>
</tr>
</tbody>
</table>

188.3.1 ADD_COLORED_SQL Procedure

This procedure adds a colored SQL ID.

If an SQL ID is colored, it will be captured in every snapshot, independent of its level of activities (so that it does not have to be a TOP SQL). Capture occurs if the SQL is found in the cursor cache at snapshot time. To uncolor the SQL, invoke the REMOVE_COLORED_SQL Procedure.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.ADD_COLORED_SQL(
    sql_id         IN VARCHAR2,
    dbid           IN NUMBER DEFAULT NULL);
```

Parameters

Table 188-3 ADD_COLORED_SQL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>13-character external SQL ID</td>
</tr>
<tr>
<td>dbid</td>
<td>Optional DBID, defaults to Local DBID</td>
</tr>
</tbody>
</table>

188.3.2 ASH_GLOBAL_REPORT_HTML Function

This table function displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in HTML format.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.ASH_GLOBAL_REPORT_HTML(
    l_dbid          IN NUMBER,
    l_inst_num      IN VARCHAR2((1023),
    l_btime         IN DATE,
    l_etime         IN DATE,
    l_options       IN NUMBER    DEFAULT 0,
    l_slot_width    IN NUMBER    DEFAULT 0,
    l_sid           IN NUMBER    DEFAULT NULL,
    l_sql_id        IN VARCHAR2  DEFAULT NULL,
```


Parameters

Table 188-4  ASH_GLOBAL_REPORT_HTML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>List of instances (such as '1,2,3'), or NULL to report on all instances in the database</td>
</tr>
<tr>
<td>l_btime</td>
<td>The 'begin time'</td>
</tr>
<tr>
<td>l_etime</td>
<td>The 'end time'</td>
</tr>
<tr>
<td>l_options</td>
<td>Report level (currently not used)</td>
</tr>
<tr>
<td>l_slot_width</td>
<td>Specifies (in seconds) how wide the slots used in the &quot;Top Activity&quot; section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.</td>
</tr>
<tr>
<td>l_sid</td>
<td>Session ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_sql_id</td>
<td>SQL ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_wait_class</td>
<td>Wait class name (see Usage Notes)</td>
</tr>
<tr>
<td>l_service_hash</td>
<td>Service name hash (see Usage Notes)</td>
</tr>
<tr>
<td>l_module</td>
<td>Module name (see Usage Notes)</td>
</tr>
<tr>
<td>l_action</td>
<td>Action name (see Usage Notes)</td>
</tr>
<tr>
<td>l_client_id</td>
<td>Client ID for end-to-end backtracing (see Usage Notes)</td>
</tr>
<tr>
<td>l_plsql_entry</td>
<td>PL/SQL entry point (see Usage Notes)</td>
</tr>
<tr>
<td>l_data_src</td>
<td>Ignored since the report works off of data on disk only</td>
</tr>
</tbody>
</table>
| l_container    | Name of the container for which report activity is limited. Valid values other than NULL (default) should be taken from container names in V$CONTAINERS. Behavior is as follows:  
  - If NULL: When connected to a root container the report is on all containers. When connected to a PDB the report is on only that PDB.  
  - If not NULL: When connected to a root container the report is on activity from the specified container. When connected to a PDB the report is the same as NULL value for l_container regarding the connected PDB.  
  Note: If while connected to a PDB you request information from another PDB this produces an empty report. |
Return Values

The output will be one column of VARCHAR2 (1500).

Usage Notes

- You can call the function directly but Oracle recommends you use the ashrpti.sql script which prompts users for the required information.

- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as SQL_ID 'abcdefghijkl23' pass that sql_id value to the l_sql_id argument:

  l_sql_id => 'abcdefghijkl23'

Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, AND conditional logic is used to connect them. For example, to generate an ASH report on MODULE "PAYROLL" and ACTION "PROCESS", use the following predicate:

  l_module => 'PAYROLL', l_action => 'PROCESS'

Valid SQL wildcards can be used in all the arguments that are of type VARCHAR2.

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Comment</th>
<th>Wildcard Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_sid</td>
<td>Session ID (for example, VARCHAR2(1023))</td>
<td>No</td>
</tr>
<tr>
<td>l_sql_id</td>
<td>SQL ID (for example, VARCHAR2(1023))</td>
<td>Yes</td>
</tr>
<tr>
<td>l_wait_class</td>
<td>Wait class name (for example, VARCHAR2(1023))</td>
<td>Yes</td>
</tr>
<tr>
<td>l_service_hash</td>
<td>Service name hash (for example, VARCHAR2(1023))</td>
<td>No</td>
</tr>
<tr>
<td>l_module</td>
<td>Module name (for example, VARCHAR2(1023))</td>
<td>Yes</td>
</tr>
<tr>
<td>l_action</td>
<td>Action name (for example, VARCHAR2(1023))</td>
<td>Yes</td>
</tr>
<tr>
<td>l_client_id</td>
<td>Client ID for end-to-end backtracing (for example, VARCHAR2(1023))</td>
<td>Yes</td>
</tr>
<tr>
<td>l_data_src</td>
<td>Wildcards are not allowed for VARCHAR2(1023) as it is of numeric datatype</td>
<td>No</td>
</tr>
</tbody>
</table>

188.3.3 ASH_GLOBAL_REPORT_TEXT Function

This table function displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in text format.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.ASH_GLOBAL_REPORT_TEXT(
  l_dbid               IN VARCHAR2(1023),
  l_inst_num           IN NUMBER,
)```

188-7
l_btime IN DATE,
l_etime IN DATE,
l_options IN NUMBER DEFAULT 0,
l_slot_width IN NUMBER DEFAULT 0,
l_sid IN NUMBER DEFAULT NULL,
l_sql_id IN VARCHAR2 DEFAULT NULL,
l_wait_class IN VARCHAR2 DEFAULT NULL,
l_service_hash IN NUMBER DEFAULT NULL,
l_module IN VARCHAR2 DEFAULT NULL,
l_action IN VARCHAR2 DEFAULT NULL,
l_client_id IN VARCHAR2 DEFAULT NULL,
l_plsql_entry IN VARCHAR2 DEFAULT NULL,
l_data_src IN NUMBER DEFAULT 0,
l_container IN VARCHAR2 DEFAULT NULL)
RETURN awrpt_text_type_table PIPELINED;

Parameters

Table 188-6  ASH_GLOBAL_REPORT_TEXT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>List of instances (such as '1, 2, 3'), or NULL to report on all instances in the database</td>
</tr>
<tr>
<td>l_btime</td>
<td>The 'begin time'</td>
</tr>
<tr>
<td>l_etime</td>
<td>The 'end time'</td>
</tr>
<tr>
<td>l_options</td>
<td>Report level (currently not used)</td>
</tr>
<tr>
<td>l_slot_width</td>
<td>Specifies (in seconds) how wide the slots used in the &quot;Top Activity&quot; section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.</td>
</tr>
<tr>
<td>l_sid</td>
<td>Session ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_sql_id</td>
<td>SQL ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_wait_class</td>
<td>Wait class name (see Usage Notes)</td>
</tr>
<tr>
<td>l_service_hash</td>
<td>Service name hash (see Usage Notes)</td>
</tr>
<tr>
<td>l_module</td>
<td>Module name (see Usage Notes)</td>
</tr>
<tr>
<td>l_action</td>
<td>Action name (see Usage Notes)</td>
</tr>
<tr>
<td>l_client_id</td>
<td>Client ID for end-to-end backtracing (see Usage Notes)</td>
</tr>
<tr>
<td>l_plsql_entry</td>
<td>PL/SQL entry point (see Usage Notes)</td>
</tr>
<tr>
<td>l_data_src</td>
<td>Ignored since the report works off of data on disk only</td>
</tr>
</tbody>
</table>
### Table 188-6 (Cont.) ASH_GLOBAL_REPORT_TEXT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_container</td>
<td>Name of the container for which report activity is limited. Valid values other than NULL (default) should be taken from container names in V$CONTAINERS. Behavior is as follows:</td>
</tr>
<tr>
<td></td>
<td>· If NULL: When connected to a root container the report is on all containers. When connected to a PDB the report is on only that PDB.</td>
</tr>
<tr>
<td></td>
<td>· If not NULL: When connected to a root container the report is on activity from the specified container. When connected to a PDB the report is the same as NULL value for l_container regarding the connected PDB.</td>
</tr>
<tr>
<td></td>
<td>Note: If while connected to a PDB you request information from another PDB this produces an empty report.</td>
</tr>
</tbody>
</table>

### Return Values

The output will be one column of VARCHAR2 (320).

### Usage Notes

- You can call the function directly but Oracle recommends you use the ashrpti.sql script which prompts users for the required information.
- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as SQL_ID 'abcdefghij123' pass that SQL_ID value to the l_sql_id argument:

  l_sql_id => 'abcdefghij123'

### Table 188-7 ASH_GLOBAL_REPORT_TEXT: Wildcards Allowed (or Not) in Arguments

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Comment</th>
<th>Wildcard Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_sid</td>
<td>Session ID (for example, V$SESSION.SID)</td>
<td>No</td>
</tr>
<tr>
<td>l_sql_id</td>
<td>SQL ID (for example, V$SQL.SQL_ID)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_wait_class</td>
<td>Wait class name (for example, V$EVENT_NAME.WAIT_CLASS)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_service_hash</td>
<td>Service name hash (for example, V$ACTIVE_SERVICES.NAME_HASH)</td>
<td>No</td>
</tr>
<tr>
<td>l_module</td>
<td>Module name (for example, V$SESSION.MODULE)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_action</td>
<td>Action name (for example, V$SESSION.ACTION)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_client_id</td>
<td>Client ID for end-to-end backtracing (for example, V$SESSION.CLIENT_IDENTIFIER)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Table 188-7** ASH_GLOBAL_REPORT_TEXT: Wildcards Allowed (or Not) in Arguments

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Comment</th>
<th>Wildcard Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_plsql_entry</td>
<td>PL/SQL entry point (for example, &quot;SYS.DBMS_LOB.&quot;*)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_data_src</td>
<td>Wildcards are not allowed for l_data_src as it is of numeric datatype</td>
<td>No</td>
</tr>
</tbody>
</table>

- Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, **AND** conditional logic is used to connect them. For example, to generate an ASH report on **MODULE** "PAYROLL" and **ACTION** "PROCESS", use the following predicate:

  l_module => 'PAYROLL', l_action => 'PROCESS'

- Valid SQL wildcards can be used in all the arguments that are of type **VARCHAR2**.

### 188.3.4 ASH_REPORT_ANALYTICS Function

This function returns the ASH Analytics active report.

**Syntax**

```sql
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_ANALYTICS(
  dbid           IN NUMBER DEFAULT NULL,
  inst_id        IN NUMBER DEFAULT NULL,
  begin_time     IN DATE,
  end_time       IN DATE,
  report_level   IN VARCHAR2 DEFAULT NULL,
  filter_list    IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;
```

**Parameters**

**Table 188-8** ASH_REPORT_ANALYTICS Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid</td>
<td>Database identifier. If its value is set to NULL, then the database identifier for the local database is used. Its default value is NULL.</td>
</tr>
<tr>
<td>inst_id</td>
<td>Instance number of the database for which the statistics are required. If its value is set to NULL, then the statistics for the local database are returned. Its default value is NULL.</td>
</tr>
<tr>
<td>begin_time</td>
<td>The start time of the interval for which the ASH report is required.</td>
</tr>
<tr>
<td>end_time</td>
<td>The end time of the interval for which the ASH report is required.</td>
</tr>
<tr>
<td>report_level</td>
<td>Describes the list of components to build.</td>
</tr>
<tr>
<td>filter_list</td>
<td>Describes the list of filters to apply. Its default value is NULL (no filters to apply).</td>
</tr>
</tbody>
</table>

**Return Values**

Returns the ASH Analytics active report.
188.3.5 ASH_REPORT_HTML Function

This table function displays the ASH Spot report in HTML.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(
    l_dbid          IN NUMBER,
    l_inst_num      IN NUMBER,
    l_btime         IN DATE,
    l_etime         IN DATE,
    l_options       IN NUMBER  DEFAULT 0,
    l_slot_width    IN NUMBER  DEFAULT 0,
    l_sid           IN NUMBER  DEFAULT NULL,
    l_sql_id        IN VARCHAR2 DEFAULT NULL,
    l_wait_class    IN VARCHAR2 DEFAULT NULL,
    l_service_hash  IN NUMBER  DEFAULT NULL,
    l_module        IN VARCHAR2 DEFAULT NULL,
    l_action        IN VARCHAR2 DEFAULT NULL,
    l_client_id     IN VARCHAR2 DEFAULT NULL,
    l_plsql_entry   IN VARCHAR2 DEFAULT NULL,
    l_data_src      IN NUMBER  DEFAULT 0,
    l_container     IN VARCHAR2 DEFAULT NULL)
RETURN awrrpt_html_type_table PIPELINED;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>Instance number</td>
</tr>
<tr>
<td>l_btime</td>
<td>The 'begin time'</td>
</tr>
<tr>
<td>l_etime</td>
<td>The 'end time'</td>
</tr>
<tr>
<td>l_options</td>
<td>Report level (currently not used)</td>
</tr>
<tr>
<td>l_slot_width</td>
<td>Specifies (in seconds) how wide the slots used in the &quot;Top Activity&quot; section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.</td>
</tr>
<tr>
<td>l_sid</td>
<td>Session ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_sql_id</td>
<td>SQL ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_wait_class</td>
<td>Wait class name (see Usage Notes)</td>
</tr>
<tr>
<td>l_service_hash</td>
<td>Service name hash (see Usage Notes)</td>
</tr>
<tr>
<td>l_module</td>
<td>Module name (see Usage Notes)</td>
</tr>
<tr>
<td>l_action</td>
<td>Action name (see Usage Notes)</td>
</tr>
<tr>
<td>l_client_id</td>
<td>Client ID for end-to-end backtracing (see Usage Notes)</td>
</tr>
<tr>
<td>l_plsql_entry</td>
<td>PL/SQL entry point (see Usage Notes)</td>
</tr>
</tbody>
</table>
### Table 188-9  (Cont.) ASH_REPORT_HTML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_data_src</td>
<td>Can be used to specify a data source (see Usage Notes)</td>
</tr>
<tr>
<td></td>
<td>1 =&gt; memory (V$ACTIVE_SESSION_HISTORY)</td>
</tr>
<tr>
<td></td>
<td>2 =&gt; disk (DBA_HIST_ACTIVE_SESS_HISTORY)</td>
</tr>
<tr>
<td></td>
<td>0 =&gt; both. This is the default value. Here, the begin and end time parameters are used to get the samples from the appropriate data source, which can be memory, disk, or both.</td>
</tr>
<tr>
<td>l_container</td>
<td>Name of the container for which report activity is limited. Valid values other than NULL (default) should be taken from container names in V$CONTAINERS. Behavior is as follows:</td>
</tr>
<tr>
<td></td>
<td>If NULL: When connected to a root container the report is on all containers. When connected to a PDB the report is on only that PDB.</td>
</tr>
<tr>
<td></td>
<td>If not NULL: When connected to a root container the report is on activity from the specified container. When connected to a PDB the report is the same as NULL value for l_container regarding the connected PDB.</td>
</tr>
</tbody>
</table>

Note: If while connected to a PDB you request information from another PDB this produces an empty report.

### Return Values

The output will be one column of VARCHAR2 (500).

### Usage Notes

- You can call the function directly but Oracle recommends you use the ashrpti.sql script which prompts users for the required information.

- By default, the report uses the begin and end time parameters (l_btime and l_etime, respectively) to find all rows in that time range either from memory, or disk, or both. However, using l_data_src, one can explicitly specify one of those data sources. For example, to generate an ASH report on all rows between l_btime and l_etime found in memory, use

\[ l_data_src = 1 \]

Similarly, to generate a report on samples found only on disk, use

\[ l_data_src = 2 \]

- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as SQL_ID 'abcdefghijkl23' pass that sql_id value to the l_sql_id argument:

\[ l_sql_id = 'abcdefghijkl23' \]

Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, AND conditional logic is used to connect them. For example, to
generate an ASH report on **MODULE "PAYROLL" and ACTION "PROCESS"**, use the following predicate:

\[
l\_module \Rightarrow 'PAYROLL', \ l\_action \Rightarrow 'PROCESS'
\]

Valid SQL wildcards can be used in all the arguments that are of type `VARCHAR2`.

### Table 188-10 ASH_REPORT_HTML: Wildcards Allowed (or Not) in Arguments

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Comment</th>
<th>Wildcard Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>l\_sid</code></td>
<td>Session ID (for example, <code>V$SESSION.SID</code>)</td>
<td>No</td>
</tr>
<tr>
<td><code>l\_sql\_id</code></td>
<td>SQL ID (for example, <code>V$SQL.SQL_ID</code>)</td>
<td>Yes</td>
</tr>
<tr>
<td><code>l\_wait\_class</code></td>
<td>Wait class name (for example, <code>V$EVENT\_NAME.WAIT\_CLASS</code>)</td>
<td>Yes</td>
</tr>
<tr>
<td><code>l\_service\_hash</code></td>
<td>Service name hash (for example, <code>V$ACTIVE\_SERVICES.NAME\_HASH</code>)</td>
<td>No</td>
</tr>
<tr>
<td><code>l\_module</code></td>
<td>Module name (for example, <code>V$SESSION.MODULE</code>)</td>
<td>Yes</td>
</tr>
<tr>
<td><code>l\_action</code></td>
<td>Action name (for example, <code>V$SESSION.ACTION</code>)</td>
<td>Yes</td>
</tr>
<tr>
<td><code>l\_client\_id</code></td>
<td>Client ID for end-to-end backtracing (for example, <code>V$SESSION.CLIENT\_IDENTIFIER</code>)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 188.3.6 ASH_REPORT_TEXT Function

This table function displays the ASH Spot report in text.

#### Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_TEXT(
    l\_dbid          IN NUMBER,
    l\_inst\_num      IN NUMBER,
    l\_btime         IN DATE,
    l\_etime         IN DATE,
    l\_options       IN NUMBER  DEFAULT 0,
    l\_slot\_width    IN NUMBER  DEFAULT 0,
    l\_sid           IN NUMBER  DEFAULT NULL,
    l\_sql\_id        IN VARCHAR2 DEFAULT NULL,
    l\_wait\_class    IN VARCHAR2 DEFAULT NULL,
    l\_service\_hash  IN NUMBER  DEFAULT NULL,
    l\_module        IN VARCHAR2 DEFAULT NULL,
    l\_action        IN VARCHAR2 DEFAULT NULL,
    l\_client\_id     IN VARCHAR2 DEFAULT NULL,
    l\_plsql\_entry   IN VARCHAR2 DEFAULT NULL,
    l\_data\_src      IN NUMBER  DEFAULT 0,
    l\_container     IN VARCHAR2 DEFAULT NULL)
RETURN awrpt\_text\_type\_table PIPELINED;
```

#### Parameters

### Table 188-11 ASH_REPORT_TEXT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>l\_dbid</code></td>
<td>Database identifier</td>
</tr>
</tbody>
</table>
Table 188-11  (Cont.) ASH_REPORT_TEXT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_inst_num</td>
<td>Instance number</td>
</tr>
<tr>
<td>l_btime</td>
<td>The 'begin time'</td>
</tr>
<tr>
<td>l_etime</td>
<td>The 'end time'</td>
</tr>
<tr>
<td>l_options</td>
<td>Report level (currently not used)</td>
</tr>
<tr>
<td>l_slot_width</td>
<td>Specifies (in seconds) how wide the slots used in the &quot;Top Activity&quot; section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.</td>
</tr>
<tr>
<td>l_sid</td>
<td>Session ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_sql_id</td>
<td>SQL ID (see Usage Notes)</td>
</tr>
<tr>
<td>l_wait_class</td>
<td>Wait class name (see Usage Notes)</td>
</tr>
<tr>
<td>l_service_hash</td>
<td>Service name hash (see Usage Notes)</td>
</tr>
<tr>
<td>l_module</td>
<td>Module name (see Usage Notes)</td>
</tr>
<tr>
<td>l_action</td>
<td>Action name (see Usage Notes)</td>
</tr>
<tr>
<td>l_client_id</td>
<td>Client ID for end-to-end backtracing (see Usage Notes)</td>
</tr>
<tr>
<td>l_plsql_entry</td>
<td>PL/SQL entry point (see Usage Notes)</td>
</tr>
<tr>
<td>l_data_src</td>
<td>Can be used to specify a data source (see Usage Notes)</td>
</tr>
<tr>
<td></td>
<td>• 1 =&gt; memory (V$ACTIVE_SESSION_HISTORY)</td>
</tr>
<tr>
<td></td>
<td>• 2 =&gt; disk (DBA_HIST_ACTIVE_SESS_HISTORY)</td>
</tr>
<tr>
<td></td>
<td>• 0 =&gt; both. This is the default value. Here, the begin and end time parameters are used to get the samples from the appropriate data source, which can be memory, disk, or both.</td>
</tr>
<tr>
<td>l_container</td>
<td>Name of the container for which report activity is limited. Valid values other than NULL (default) should be taken from container names in V$CONTAINERS. Behavior is as follows:</td>
</tr>
<tr>
<td></td>
<td>• If NULL: When connected to a root container the report is on all containers. When connected to a PDB the report is on only that PDB.</td>
</tr>
<tr>
<td></td>
<td>• If not NULL: When connected to a root container the report is on activity from the specified container. When connected to a PDB the report is the same as NULL value for l_container regarding the connected PDB.</td>
</tr>
<tr>
<td></td>
<td>Note: If while connected to a PDB you request information from another PDB this produces an empty report.</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(80).

Usage Notes

• You can call the function directly but Oracle recommends you use the ashrti.sql script which prompts users for the required information.
• By default, the report uses the begin and end time parameters (l_btime and l_etime, respectively) to find all rows in that time range either from memory, or
disk, or both. However, using l_data_src, one can explicitly specify one of those data sources. For example, to generate an ASH report on all rows between l_btime and l_time found in memory, use

l_data_src => 1

Similarly, to generate a report on samples found only on disk, use

l_data_src => 2

• The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as SQL_ID 'abcdefghij123' pass that SQL_ID value to the l_sql_id argument:

l_sql_id => 'abcdefghij123'

Table 188-12  ASH_REPORT_TEXT: Wildcards Allowed (or Not) in Arguments

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Comment</th>
<th>Wildcard Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_sid</td>
<td>Session ID (for example, V$SESSION.SID)</td>
<td>No</td>
</tr>
<tr>
<td>l_sql_id</td>
<td>SQL ID (for example, V$SQL.SQL_ID)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_wait_class</td>
<td>Wait class name (for example, V$EVENT_NAME.WAIT_CLASS)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_service_hash</td>
<td>Service name hash (for example, V$ACTIVE_SERVICES.NAME_HASH)</td>
<td>No</td>
</tr>
<tr>
<td>l_module</td>
<td>Module name (for example, V$SESSION.MODULE)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_action</td>
<td>Action name (for example, V$SESSION.ACTION)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_client_id</td>
<td>Client ID for end-to-end backtracing (for example, V$SESSION.CLIENT_IDENTIFIER)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_plsql_entry</td>
<td>PL/SQL entry point (for example, &quot;SYS.DBMS_LOB.&quot;)</td>
<td>Yes</td>
</tr>
<tr>
<td>l_data_src</td>
<td>Wildcards are not allowed for l_data_src as it is of numeric datatype</td>
<td>No</td>
</tr>
</tbody>
</table>

• Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, AND conditional logic is used to connect them. For example, to generate an ASH report on MODULE "PAYROLL" and ACTION "PROCESS", use the following predicate:

l_module => 'PAYROLL', l_action => 'PROCESS'

Valid SQL wildcards can be used in all the arguments that are of type VARCHAR2.
### 188.3.7 AWR_DIFF_REPORT_HTML Function

This table function displays the AWR Compare Periods report in HTML.

**Syntax**

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_HTML(
    dbid1 IN NUMBER,
    inst_num1 IN NUMBER,
    bid1 IN NUMBER,
    eid1 IN NUMBER,
    dbid2 IN NUMBER,
    inst_num2 IN NUMBER,
    bid2 IN NUMBER,
    eid2 IN NUMBER)
RETURN awrdrpt_text_type_table PIPELINED;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid1</td>
<td>1st database identifier</td>
</tr>
<tr>
<td>inst_num1</td>
<td>1st instance number</td>
</tr>
<tr>
<td>bid1</td>
<td>1st beginning snapshot ID</td>
</tr>
<tr>
<td>eid1</td>
<td>1st ending snapshot ID</td>
</tr>
<tr>
<td>dbid2</td>
<td>2nd database identifier</td>
</tr>
<tr>
<td>inst_num2</td>
<td>2nd instance number</td>
</tr>
<tr>
<td>bid2</td>
<td>2nd beginning snapshot ID</td>
</tr>
<tr>
<td>eid2</td>
<td>2nd ending snapshot ID</td>
</tr>
</tbody>
</table>

**Return Values**

The output will be one column of VARCHAR2(500).

**Usage Notes**

You can call the function directly but Oracle recommends you use the `awrddrpt.sql` script which prompts users for the required information.

### 188.3.8 AWR_DIFF_REPORT_TEXT Function

This table function displays the AWR Compare Periods report in text.

**Syntax**

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_TEXT(
    dbid1 IN NUMBER,
    inst_num1 IN NUMBER,
    bid1 IN NUMBER,
    eid1 IN NUMBER,
    dbid2 IN NUMBER,
```
Parameters

Table 188-14  AWR_DIFF_REPORT_TEXT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid1</td>
<td>1st database identifier</td>
</tr>
<tr>
<td>inst_num1</td>
<td>1st instance number</td>
</tr>
<tr>
<td>bid1</td>
<td>1st beginning snapshot ID</td>
</tr>
<tr>
<td>eid1</td>
<td>1st ending snapshot ID</td>
</tr>
<tr>
<td>dbid2</td>
<td>2nd database identifier</td>
</tr>
<tr>
<td>inst_num2</td>
<td>2nd instance number</td>
</tr>
<tr>
<td>bid2</td>
<td>2nd beginning snapshot ID</td>
</tr>
<tr>
<td>eid2</td>
<td>2nd ending snapshot ID</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(500).

Usage Notes

You can call the function directly but Oracle recommends you use the awrddrpt.sql script which prompts users for the required information.

188.3.9 AWR_GLOBAL_DIFF_REPORT_HTML Functions

This table function displays Global AWR Compare Periods Report in HTML format.

The first overload accepts comma-separated lists of instance numbers for `inst_num1` and `inst_num2`. No leading zeroes are allowed and there is a limit of 1023 characters.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_HTML (  
dbid1      IN    NUMBER,  
inst_num1  IN    AWRRPT_INSTANCE_LIST_TYPE,  
bid1       IN    NUMBER,  
eid1       IN    NUMBER,  
dbid2      IN    NUMBER,  
inst_num2  IN    AWRRPT_INSTANCE_LIST_TYPE,  
bid2       IN    NUMBER,  
eid2       IN    NUMBER)  
RETURN awrrpt_html_type_table PIPELINED;
```

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_HTML (  
dbid1      IN    NUMBER,  
inst_num1  IN    VARCHAR2,  
bid1       IN    NUMBER,  
eid1       IN    NUMBER,  
dbid2      IN    NUMBER,  
```
Parameters

Table 188-15  AWR_GLOBAL_DIFF_REPORT_HTML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid1</td>
<td>1st database identifier</td>
</tr>
<tr>
<td>inst_num1</td>
<td>1st list of instance numbers. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.</td>
</tr>
<tr>
<td>bid1</td>
<td>1st beginning snapshot ID</td>
</tr>
<tr>
<td>eid1</td>
<td>1st ending snapshot ID</td>
</tr>
<tr>
<td>dbid2</td>
<td>2nd database identifier</td>
</tr>
<tr>
<td>inst_num2</td>
<td>2nd list of instance numbers to be included in report. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.</td>
</tr>
<tr>
<td>bid2</td>
<td>2nd beginning snapshot ID</td>
</tr>
<tr>
<td>eid2</td>
<td>2nd ending snapshot ID</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(1500).

188.3.10 AWR_GLOBAL_DIFF_REPORT_TEXT Functions

This table function displays Global AWR Compare Periods Report in text format.

The first overload accepts comma-separated lists of instance numbers for inst_num1 and inst_num2. No leading zeroes are allowed and there is a limit of 1023 characters.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_TEXT (  
    dbid1        IN    NUMBER,  
    inst_num1    IN    AWRRPT_INSTANCE_LIST_TYPE,  
    bid1         IN    NUMBER,  
    eid1         IN    NUMBER,  
    dbid2        IN    NUMBER,  
    inst_num2    IN    AWRRPT_INSTANCE_LIST_TYPE,  
    bid2         IN    NUMBER,  
    eid2         IN    NUMBER)  
RETURN awrdrpt_text_type_table PIPELINED;
```

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_TEXT (  
    dbid1        IN    NUMBER,  
    inst_num1    IN    VARCHAR2,  
    bid1         IN    NUMBER,  
    eid1         IN    NUMBER,  
    dbid2        IN    NUMBER,  
```
Parameters

Table 188-16  AWR_GLOBAL_DIFF_REPORT_TEXT Functions Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid1</td>
<td>1st database identifier</td>
</tr>
<tr>
<td>inst_num1</td>
<td>1st list of instance numbers. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.</td>
</tr>
<tr>
<td>bid1</td>
<td>1st beginning snapshot ID</td>
</tr>
<tr>
<td>eid1</td>
<td>1st ending snapshot ID</td>
</tr>
<tr>
<td>dbid2</td>
<td>2nd database identifier</td>
</tr>
<tr>
<td>inst_num2</td>
<td>2nd list of instance numbers to be included in report. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.</td>
</tr>
<tr>
<td>bid2</td>
<td>2nd beginning snapshot ID</td>
</tr>
<tr>
<td>eid2</td>
<td>2nd ending snapshot ID</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(320).

188.3.11 AWR_GLOBAL_REPORT_HTML Functions

This table function displays the Global AWR report in HTML.

The first overload accepts a comma-separated list of instance numbers. No leading zeroes are allowed and there is a limit of 1023 characters.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_HTML (  
  l_dbid    IN    NUMBER,  
  l_inst_num    IN    AWRRT_INSTANCE_LIST_TYPE,  
  l_bid    IN    NUMBER,  
  l_eid    IN    NUMBER,  
  l_options    IN    NUMBER DEFAULT 0)  
RETURN awrrpt_html_type_table PIPELINED;
```

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_HTML (  
  l_dbid    IN    NUMBER,  
  l_inst_num    IN    VARCHAR2,  
  l_bid    IN    NUMBER,  
  l_eid    IN    NUMBER,  
  l_options    IN    NUMBER DEFAULT 0)  
RETURN awrrpt_html_type_table PIPELINED;
```
Parameters

Table 188-17  AWR_GLOBAL_REPORT_HTML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>List of instance numbers to be included in report. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.</td>
</tr>
<tr>
<td>l_bid</td>
<td>Beginning snapshot ID</td>
</tr>
<tr>
<td>l_eid</td>
<td>Ending snapshot ID</td>
</tr>
<tr>
<td>l_options</td>
<td>Report level (currently not used)</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(1500).

188.3.12 AWR_GLOBAL_REPORT_TEXT Functions

This table function displays the Global AWR report in text.

The first overload accepts a comma-separated list of instance numbers. No leading zeroes are allowed and there is a limit of 1023 characters.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_TEXT(
  l_dbid    IN    NUMBER,
  l_inst_num IN    AWRRPT_INSTANCE_LIST_TYPE,
  l_bid     IN    NUMBER,
  l_eid     IN    NUMBER,
  l_options IN    NUMBER DEFAULT 0)
RETURN awrdrpt_text_type_table PIPELINED;
```

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_TEXT(
  l_dbid    IN    NUMBER,
  l_inst_num IN    VARCHAR2,
  l_bid     IN    NUMBER,
  l_eid     IN    NUMBER,
  l_options IN    NUMBER DEFAULT 0)
RETURN awrdrpt_text_type_table PIPELINED;
```

Parameters

Table 188-18  AWR_GLOBAL_REPORT_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
</tbody>
</table>
### Table 188-18  (Cont.) AWR_GLOBAL_REPORT_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_inst_num</td>
<td>List of instance numbers to be included in report. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.</td>
</tr>
<tr>
<td>l_bid</td>
<td>Beginning snapshot ID</td>
</tr>
<tr>
<td>l_eid</td>
<td>Ending snapshot ID</td>
</tr>
<tr>
<td>l_options</td>
<td>A flag to specify to control the output of the report. Currently, not used.</td>
</tr>
</tbody>
</table>

**Return Values**

The output will be one column of VARCHAR2(320).

#### 188.3.13 AWR_REPORT_HTML Function

This table function displays the AWR report in HTML.

**Syntax**

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML(
    l_dbid IN NUMBER,
    l_inst_num IN NUMBER,
    l_bid IN NUMBER,
    l_eid IN NUMBER,
    l_options IN NUMBER DEFAULT 0
) RETURN awrrpt_text_type_table PIPELINED;
```

**Parameters**

#### Table 188-19  AWR_REPORT_HTML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>Instance number</td>
</tr>
<tr>
<td>l_bid</td>
<td>Beginning snapshot ID</td>
</tr>
<tr>
<td>l_eid</td>
<td>Ending snapshot ID</td>
</tr>
<tr>
<td>l_options</td>
<td>A flag to specify to control the output of the report. Currently, Oracle supports one value:</td>
</tr>
<tr>
<td></td>
<td>- l_options = 8. Displays the ADDM specific portions of the report. These sections include the Buffer Pool Advice, Shared Pool Advice, and PGA Target Advice.</td>
</tr>
</tbody>
</table>

**Return Values**

The output will be one column of VARCHAR2(1500).
Usage Notes

You can call the function directly but Oracle recommends you use the awrrpt.sql script which prompts users for the required information.

188.3.14 AWR_REPORT_TEXT Function

This table function displays the AWR report in text.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
    l_dbid       IN    NUMBER,
    l_inst_num   IN    NUMBER,
    l_bid        IN    NUMBER,
    l_eid        IN    NUMBER,
    l_options    IN    NUMBER DEFAULT 0)
RETURN awrrpt_text_type_table PIPELINED;
```

Parameters

**Table 188-20  AWR_REPORT_TEXT Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>Instance number</td>
</tr>
<tr>
<td>l_bid</td>
<td>Beginning snapshot ID</td>
</tr>
<tr>
<td>l_eid</td>
<td>Ending snapshot ID</td>
</tr>
<tr>
<td>l_options</td>
<td>A flag to specify to control the output of the report. Currently, Oracle supports one value:</td>
</tr>
<tr>
<td></td>
<td>· l_options - 8. Displays the ADDM specific portions of the report. These sections include the Buffer Pool Advice, Shared Pool Advice, and PGA Target Advice.</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(80).

Usage Notes

You can call the function directly but Oracle recommends you use the awrrpt.sql script which prompts users for the required information.

188.3.15 AWR_SET_REPORT_THRESHOLDS Procedure

This procedure configure specified report thresholds, including the number of rows in the report.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_SET_REPORT_THRESHOLDS(
    top_n_events         IN   NUMBER DEFAULT NULL,
    top_n_files          IN   NUMBER DEFAULT NULL,
```
Parameters

Table 188-21  AWR_SET_REPORT_THRESHOLDS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top_n_events</td>
<td>Number of most significant wait events to be included</td>
</tr>
<tr>
<td>top_n_files</td>
<td>Number of most active files to be included</td>
</tr>
<tr>
<td>top_n_segments</td>
<td>Number of most active segments to be included</td>
</tr>
<tr>
<td>top_n_services</td>
<td>Number of most active services to be included</td>
</tr>
<tr>
<td>top_n_sql</td>
<td>Number of most significant SQL statements to be included</td>
</tr>
<tr>
<td>top_n_sql_max</td>
<td>Number of SQL statements to be included if their activity is greater than</td>
</tr>
<tr>
<td></td>
<td>that specified by top_sql_pct</td>
</tr>
<tr>
<td>top_sql_pct</td>
<td>Significance threshold for SQL statements between top_n_sql and top_n_sql_max</td>
</tr>
<tr>
<td>shmem_threshold</td>
<td>Shared memory low threshold</td>
</tr>
<tr>
<td>versions_threshold</td>
<td>Plan version count low threshold</td>
</tr>
<tr>
<td>top_n_disks</td>
<td>Number of cell disks with most I/O</td>
</tr>
<tr>
<td>outlier_pct</td>
<td>Percentage of maximum capacity before displaying outliers for Exadata</td>
</tr>
<tr>
<td>outlier_cpu_pct</td>
<td>Threshold for mean percentage CPU to display outliers</td>
</tr>
</tbody>
</table>

User Notes

- The effect of each setting depends on the type of report being generated as well as on the underlying AWR data. Not all settings are meaningful for each report type. Invalid settings (such as negative numbers) are ignored.
- Settings are effective only in the context of the session that executes the AWR_SET_REPORT_THRESHOLDS procedure. For example, to get a report that lists top 12 segments as compared to the default, one can invoke as follows:
  
  DBMS_WORKLOAD_REPOSITORY.AWR_SET_REPORT_THRESHOLDS (top_n_segments=>12);

188.3.16 AWR_SQL_REPORT_HTML Function

This table function displays the AWR SQL Report in HTML format.

Syntax

DBMS_WORKLOAD_REPOSITORY.AWR_SQL_REPORT_HTML(
  l_dbid IN NUMBER,
Parameters

Table 188-22  **AWR_SQL_REPORT_HTML Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>Instance number</td>
</tr>
<tr>
<td>l_bid</td>
<td>Beginning snapshot ID</td>
</tr>
<tr>
<td>l_eid</td>
<td>Ending snapshot ID</td>
</tr>
<tr>
<td>l_sqlid</td>
<td>SQL ID of statement to be analyzed</td>
</tr>
<tr>
<td>l_options</td>
<td>A flag to specify to control the output of the report.</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(500).

Usage Notes

You can call the function directly but Oracle recommends you use the awrsqrpt.sql script which prompts users for the required information.

### 188.3.17 AWR_SQL_REPORT_TEXT Function

This table function displays the AWR SQL Report in text format.

**Syntax**

```sql
DBMS_WORKLOAD_REPOSITORY.AWR_SQL_REPORT_TEXT(
    l_dbid       IN    NUMBER,
    l_inst_num   IN    NUMBER,
    l_bid        IN    NUMBER,
    l_eid        IN    NUMBER,
    l_sqlid      IN    VARCHAR2,
    l_options    IN    NUMBER DEFAULT 0)
RETURN awrrpt_text_type_table PIPELINED;
```

Parameters

Table 188-23  **AWR_SQL_REPORT_TEXT Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_dbid</td>
<td>Database identifier</td>
</tr>
<tr>
<td>l_inst_num</td>
<td>Instance number</td>
</tr>
<tr>
<td>l_bid</td>
<td>Beginning snapshot ID</td>
</tr>
</tbody>
</table>
Table 188-23  (Cont.) AWR_SQL_REPORT_TEXT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_eid</td>
<td>Ending snapshot ID</td>
</tr>
<tr>
<td>l_sqlid</td>
<td>SQL ID of statement to be analyzed</td>
</tr>
<tr>
<td>l_options</td>
<td>Flag to specify to control the output of the report. Currently, not used.</td>
</tr>
</tbody>
</table>

Return Values

The output will be one column of VARCHAR2(120).

Usage Notes

You can call the function directly but Oracle recommends you use the awrsqrpt.sql script which prompts users for the required information.

188.3.18 CONTROL_RESTRICTED_SNAPSHOT Procedure

This procedure controls the AWR snapshot creation for a database in the restricted session mode. If this procedure is not used, then by default, the AWR snapshots cannot be created for a database in the restricted session mode. This procedure affects the local database on which it is executed.

Note:

To enable AWR snapshot creation for an Oracle RAC in the restricted session mode, this procedure must be executed on every database instance in the Oracle RAC.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.CONTROL_RESTRICTED_SNAPSHOT(
   allow IN BOOLEAN);
```

Parameters

Table 188-24  CONTROL_RESTRICTED_SNAPSHOT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>This parameter can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• TRUE: AWR snapshots can be created for the database in the restricted session mode.</td>
</tr>
<tr>
<td></td>
<td>• FALSE: AWR snapshots cannot be created for the database in the restricted session mode.</td>
</tr>
</tbody>
</table>
188.3.19 CREATE_BASELINE Functions and Procedures

This function and procedure creates a baseline.

Syntax

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
    start_snap_id    IN  NUMBER,
    end_snap_id      IN  NUMBER,
    baseline_name    IN  VARCHAR2,
    dbid             IN  NUMBER DEFAULT NULL,
    expiration       IN  NUMBER DEFAULT NULL);

DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
    start_snap_id    IN  NUMBER,
    end_snap_id      IN  NUMBER,
    baseline_name    IN  VARCHAR2,
    dbid             IN  NUMBER DEFAULT NULL,
    expiration       IN  NUMBER DEFAULT NULL) RETURN NUMBER;

DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
    start_time       IN  DATE,
    end_time         IN  DATE,
    baseline_name    IN  VARCHAR2,
    dbid             IN  NUMBER DEFAULT NULL,
    expiration       IN  NUMBER DEFAULT NULL) RETURN NUMBER;

DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
    start_time       IN  DATE,
    end_time         IN  DATE,
    baseline_name    IN  VARCHAR2,
    dbid             IN  NUMBER DEFAULT NULL,
    expiration       IN  NUMBER DEFAULT NULL) RETURN NUMBER;
```

Parameters

Table 188-25  CREATE_BASELINE Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_snap_id</td>
<td>Start snapshot sequence number for the baseline.</td>
</tr>
<tr>
<td>end_snap_id</td>
<td>End snapshot sequence number for the baseline.</td>
</tr>
<tr>
<td>start_time</td>
<td>Start time for the baseline.</td>
</tr>
<tr>
<td>end_time</td>
<td>End time for the baseline.</td>
</tr>
<tr>
<td>baseline_name</td>
<td>Name of baseline.</td>
</tr>
<tr>
<td>dbid</td>
<td>Database Identifier for baseline. If NULL, this takes the database identifier for the local database. Defaults to NULL.</td>
</tr>
<tr>
<td>expiration</td>
<td>Expiration in number of days for the baseline. If NULL, then expiration is infinite, meaning do not drop baseline ever. Defaults to NULL.</td>
</tr>
</tbody>
</table>
Exceptions

- An error will be returned if this baseline name already exists in the system.
- The snapshot range that is specified for this interface must be an existing pair of snapshots in the database. An error will be returned if the inputted snapshots do not exist in the system.

Examples

This example creates a baseline (named 'oltp_peakload_bl') between snapshots 105 and 107 for the local database:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE (start_snap_id => 105,
                                              end_snap_id   => 107,
                                              baseline_name => 'oltp_peakload_bl');
```

If you query the `DBA_HIST_BASELINE` view after the `CREATE BASELINE` action, you will see the newly created baseline in the Workload Repository.

188.3.20 CREATE_BASELINE_TEMPLATE Procedures

This procedure specifies a template for how they would like baselines to be created for future time periods.

Syntax

Specifies a template for generating a baseline for a single time period in the future.

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE(
    start_time              IN DATE,
    end_time                IN DATE,
    baseline_name           IN VARCHAR2,
    template_name           IN VARCHAR2,
    expiration              IN NUMBER,
    dbid                    IN NUMBER   DEFAULT NULL);
```

Specifies a template for creating and dropping baseline based on repeating time periods:

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE(
    day_of_week             IN VARCHAR2,
    hour_in_day             IN NUMBER,
    duration                IN NUMBER,
    start_time              IN DATE,
    end_time                IN DATE,
    baseline_name_prefix    IN VARCHAR2,
    template_name           IN VARCHAR2,
    expiration              IN NUMBER,
    dbid                    IN NUMBER   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_time</td>
<td>Start Time for the baseline to be created</td>
</tr>
</tbody>
</table>
Table 188-26  (Cont.) CREATE_BASELINE_TEMPLATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end_time</td>
<td>End Time for the baseline to be created</td>
</tr>
<tr>
<td>baseline_name</td>
<td>Name of baseline to be created</td>
</tr>
<tr>
<td>template_name</td>
<td>Name for the template</td>
</tr>
<tr>
<td>expiration</td>
<td>Expiration in number of days for the baseline. If NULL, then expiration is</td>
</tr>
<tr>
<td></td>
<td>infinite, meaning do not drop baseline ever. Defaults to NULL</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID for which the baseline template needs to be used. If NULL, this</td>
</tr>
<tr>
<td></td>
<td>takes the database identifier of the local database. Defaults to NULL</td>
</tr>
<tr>
<td>day_of_week</td>
<td>Day of week that the baseline should repeat on. Specify one of the following</td>
</tr>
<tr>
<td></td>
<td>values: SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY.</td>
</tr>
<tr>
<td>hour_in_day</td>
<td>Value of 0-23 to specify the Hour in the Day the baseline should start</td>
</tr>
<tr>
<td>duration</td>
<td>Duration (in number of hours) after hour in the day that the baseline</td>
</tr>
<tr>
<td>baseline_name_prefix</td>
<td>Name for baseline prefix. When creating the baseline, the name of the</td>
</tr>
<tr>
<td></td>
<td>baseline will be the prefix appended with the date information.</td>
</tr>
</tbody>
</table>

188.3.21 CREATE_REMOTE_SNAPSHOT Function and Procedure

This function and procedure create a remote snapshot using the Remote Management Framework (RMF). The function returns the snapshot ID.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT(
    node_id IN NUMBER,
    flush_level IN VARCHAR2 DEFAULT 'BESTFIT');
```

```sql
DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT(
    node_id IN NUMBER,
    flush_level IN VARCHAR2 DEFAULT 'BESTFIT')
RETURN NUMBER;
```

```sql
DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT(
    node_name IN VARCHAR2,
    topology_name IN VARCHAR2 DEFAULT NULL,
    flush_level IN VARCHAR2 DEFAULT 'BESTFIT');
```

```sql
DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT(
    node_name IN VARCHAR2,
    topology_name IN VARCHAR2 DEFAULT NULL,
    flush_level IN VARCHAR2 DEFAULT 'BESTFIT')
RETURN NUMBER;
```
Parameters

Table 188-27 CREATE_REMOTE_SNAPSHOT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_id</td>
<td>RMF node identifier of the database for which the snapshot needs to be created.</td>
</tr>
<tr>
<td>node_name</td>
<td>RMF node name of the database for which the snapshot needs to be created.</td>
</tr>
<tr>
<td>topology_name</td>
<td>RMF topology name of the database for which the snapshot needs to be created.</td>
</tr>
<tr>
<td>flush_level</td>
<td>The flush level can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>• BESTFIT: Uses the default value depending on the type of snapshot being taken.</td>
</tr>
<tr>
<td></td>
<td>• LITE: Lightweight snapshot. Only the most important statistics are collected. This is default for a pluggable database (PDB) and application container.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: Regular snapshot. Most of the statistics are collected. This is default for a container database root (CDB root) and non-CDB database.</td>
</tr>
<tr>
<td></td>
<td>• ALL: Heavyweight snapshot. All the possible statistics are collected. This consumes a considerable amount of disk space and takes a long time to create.</td>
</tr>
</tbody>
</table>

Examples

This example creates a remote snapshot of the database having the RMF node identifier of 10:

```sql
EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT(10);
```

If you query the `DBA_HIST_SNAPSHOT` view after executing the above procedure, you will see a new snapshot record added to the Automatic Workload Repository (AWR).

188.3.22 CREATE_SNAPSHOT Function and Procedure

This function and procedure create a snapshot. The function returns the snapshot ID. If both, the database ID and the database name are not specified in this subprogram, then the snapshot is created for the local database on which the subprogram is executed.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(
    flush_level IN VARCHAR2 DEFAULT 'BESTFIT',
    dbid        IN NUMBER   DEFAULT NULL,
    source_name IN VARCHAR2 DEFAULT NULL);
```

```sql
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(
    flush_level IN VARCHAR2 DEFAULT 'BESTFIT',
    dbid        IN NUMBER   DEFAULT NULL,
    source_name IN VARCHAR2 DEFAULT NULL);
```
CREATE_SNAPSHOT Parameters

Table 188-28  CREATE_SNAPSHOT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flush_level</td>
<td>The flush level can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>• BESTFIT: Uses the default value depending on the type of snapshot being</td>
</tr>
<tr>
<td></td>
<td>taken.</td>
</tr>
<tr>
<td></td>
<td>• LITE: Lightweight snapshot. Only the most important statistics are</td>
</tr>
<tr>
<td></td>
<td>collected. This is default for a pluggable database (PDB) and application</td>
</tr>
<tr>
<td></td>
<td>container.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: Regular snapshot. Most of the statistics are collected. This is</td>
</tr>
<tr>
<td></td>
<td>default for a container database root (CDB root) and non-CDB database.</td>
</tr>
<tr>
<td></td>
<td>• ALL: Heavyweight snapshot. All the possible statistics are collected. This</td>
</tr>
<tr>
<td></td>
<td>consumes a considerable amount of disk space and takes a long time to</td>
</tr>
<tr>
<td></td>
<td>create.</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID of the database for which the snapshot needs to be</td>
</tr>
<tr>
<td></td>
<td>created.</td>
</tr>
<tr>
<td>source_name</td>
<td>Name of the database for which the snapshot needs to be created.</td>
</tr>
</tbody>
</table>

Examples

This example creates a snapshot of the local database with the flush level of ALL:

EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT('ALL');

If you query the DBA_HIST_SNAPSHOT view after executing the above procedure, you will see a new snapshot record added to the Automatic Workload Repository (AWR).

188.3.23 DROP_BASELINE Procedure

This procedure drops a previously-defined baseline.

Syntax

DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE(
    baseline_name  IN  VARCHAR2,
    cascade        IN  BOOLEAN  DEFAULT FALSE,
    dbid           IN  NUMBER   DEFAULT NULL);
Parameters

Table 188-29  DROP_BASELINE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline_name</td>
<td>Name of baseline to drop from the system</td>
</tr>
<tr>
<td>cascade</td>
<td>If TRUE, the pair of snapshots associated with the baseline will also be dropped. Otherwise, only the baseline is removed.</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID for which the baseline needs to be dropped (defaults to local DBID).</td>
</tr>
</tbody>
</table>

Examples

This example drops the baseline 'oltp_peakload_bl' without dropping the underlying snapshots:

EXECUTE DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE (baseline_name => 'oltp_peakload_bl');

If you query the DBA_HIST_BASELINE view after the DROP_BASELINE action, you will see the specified baseline definition is removed. You can query the DBA_HIST_SNAPSHOT view to find that the underlying snapshots are left intact.

188.3.24 DROP_BASELINE_TEMPLATE Procedure

This procedure removes a template that is no longer needed.

Syntax

DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE_TEMPLATE (template_name IN VARCHAR2, dbid IN NUMBER DEFAULT NULL);

Parameters

Table 188-30  DROP_BASELINE_TEMPLATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>template_name</td>
<td>Name of the template to remove</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID for which the baseline template needs to be dropped. If NULL, this takes the database identifier of the local database. Defaults to NULL.</td>
</tr>
</tbody>
</table>

188.3.25 DROP_SNAPSHOT_RANGE Procedure

This procedure drops a range of snapshots.

Syntax

DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE (low_snap_id IN NUMBER,
high_snap_id    IN  NUMBER,
dbid           IN  NUMBER   DEFAULT NULL);

Parameters

Table 188-31  DROP_SNAPSHOT_RANGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>low_snap_id</td>
<td>Low snapshot id of snapshots to drop.</td>
</tr>
<tr>
<td>high_snap_id</td>
<td>High snapshot id of snapshots to drop.</td>
</tr>
<tr>
<td>dbid</td>
<td>Database id (defaults to local DBID).</td>
</tr>
</tbody>
</table>

Examples

This example drops the range of snapshots between snapshot id 102 to 105 for the local database:

EXECUTE DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE(102, 105);

If you query the dba_hist_snapshot view after the Drop Snapshot action, you will see that snapshots 102 to 105 are removed from the Workload Repository.

188.3.26 LOCAL_AWR_DBID Function

This function returns the database identifier for the local AWR database.

Syntax

DBMS_WORKLOAD_REPOSITORY.LOCAL_AWR_DBID()
RETURN NUMBER;

188.3.27 MODIFY_BASELINE_WINDOW_SIZE Procedure

This procedure modifies the window size for the Default Moving Window Baseline.

Syntax

DBMS_WORKLOAD_REPOSITORY.MODIFY_BASELINE_WINDOW_SIZE(
    window_size    IN   NUMBER,
    dbid           IN   NUMBER   DEFAULT NULL);

Parameters

Table 188-32  MODIFY_BASELINE_WINDOW_SIZE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window_size</td>
<td>New Window size for the default Moving Window Baseline, in number of days.</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID (defaults to local DBID).</td>
</tr>
</tbody>
</table>
Usage Notes

The window size must be less than or equal to the AWR retention setting. If the window size needs to be greater than the retention setting, the MODIFY_SNAPSHOT_SETTINGS Procedures can be used to adjust the retention setting. A moving window can be set to a maximum of 13 weeks.

188.3.28 MODIFY_SNAPSHOT_SETTINGS Procedures

This procedure controls three aspects of snapshot generation.

- The INTERVAL setting affects how often snapshots are automatically captured.
- The RETENTION setting affects how long snapshots are retained in the Workload Repository.
- The number of SQL captured for each Top criteria. If the user manually specifies a value for Top N SQL, the AWR SQL collection will use the user-specified number for both automatic and manual snapshots.

There are two overloads. The first takes a NUMBER and the second takes a VARCHAR2 for the topnsql argument. The differences are described under the Parameters description.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
    retention          IN  NUMBER    DEFAULT  NULL,
    interval           IN  NUMBER    DEFAULT  NULL,
    topnsql            IN  NUMBER    DEFAULT  NULL,
    dbid               IN  NUMBER    DEFAULT  NULL);

DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
    retention          IN  NUMBER    DEFAULT NULL,
    interval           IN  NUMBER    DEFAULT NULL,
    topnsql            IN  VARCHAR2,
    dbid               IN  NUMBER    DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retention</td>
<td>New retention time (in minutes). The specified value must be in the range of MIN_RETENTION (1 day) to MAX_RETENTION (100 years). If ZERO is specified, snapshots will be retained forever. A large system-defined value will be used as the retention setting. If NULL is specified, the old value for retention is preserved. NOTE: The retention setting must be greater than or equal to the window size of the 'SYSTEM_MOVING_WINDOW' baseline. If the retention needs to be less than the window size, the MODIFY_BASELINE_WINDOW_SIZE Procedure can be used to adjust the window size.</td>
</tr>
</tbody>
</table>
### Table 188-33  (Cont.) MODIFY_SNAPSHOT_SETTINGS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| interval  | New interval setting between each snapshot, in units of minutes. The specified value must be in the range MIN_INTERVAL (10 minutes) to MAX_INTERVAL (1 year).  
  If ZERO is specified, automatic and manual snapshots will be disabled. A large system-defined value will be used as the retention setting.  
  If NULL is specified, the current value is preserved. |
| topnsql   | · If NUMBER: Top N SQL size. The number of Top SQL to flush for each SQL criteria (Elapsed Time, CPU Time, Parse Calls, Shareable Memory, Version Count). The value for this setting will not be affected by the statistics/flush level and will override the system default behavior for the AWR SQL collection. The setting will have a minimum value of 30 and a maximum value of 50,000. Specifying NULL will keep the current setting.  
  · If VARCHAR2: Users are allowed to specify the following values: (DEFAULT, MAXIMUM, N), where N is the number of Top SQL to flush for each SQL criteria. Specifying DEFAULT will revert the system back to the default behavior of Top 30 for statistics level TYPICAL and Top 100 for statistics level ALL. Specifying MAXIMUM will cause the system to capture the complete set of SQL in the cursor cache. Specifying the number N is equivalent to setting the Top N SQL with the NUMBER type. Specifying NULL for this argument will keep the current setting. |
| dbid      | Database identifier in AWR for which to modify the snapshot settings. If NULL is specified, the local dbid will be used. Defaults to NULL. |

### Examples

This example changes the interval setting to one hour and the retention setting to two weeks for the local database:

```sql
EXECUTE DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(  
  interval => 60,  
  retention => 20160);
```

If you query the DBA_HIST_WR_CONTROL table after this procedure is executed, you will see the changes to these settings.

### 188.3.29 PURGE_SQL_DETAILS Procedure

This procedure purges SQL details, specifically rows from WRH$_SQLTEXT, WRH$_SQL_PLAN, and WRH$_SQL_BIND_METADATA that do not have corresponding rows (DBID, SQL_ID) in WRH$_SQLSTAT.

The subprogram calls for the DBID for which to run the purge. If the DBID is not specified, the database DBID is used. You can constrain runtime by specifying the maximum number of rows to purge per table. If no maximum is specified, the subprograms tries to purge all applicable rows.
Syntax

DBMS_WORKLOAD_REPOSITORY.PURGE_SQL_DETAILS(
    numrows IN NUMBER DEFAULT NULL,
    dbid    IN NUMBER DEFAULT NULL);

Parameters

Table 188-34  PURGE_SQL_DETAILS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numrows</td>
<td>Number of rows</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID (default to local DBID)</td>
</tr>
</tbody>
</table>

188.3.30 REGISTER_REMOTE_DATABASE Procedures

This procedure registers a remote database in the Automatic Workload Repository (AWR) using the Remote Management Framework (RMF).

Syntax

DBMS_WORKLOAD_REPOSITORY.REGISTER_REMOTE_DATABASE(
    node_id  IN NUMBER);

DBMS_WORKLOAD_REPOSITORY.REGISTER_REMOTE_DATABASE(
    node_name      IN VARCHAR2,
    topology_name  IN VARCHAR2 DEFAULT NULL);

Parameters

Table 188-35  REGISTER_REMOTE_DATABASE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_id</td>
<td>RMF node identifier of the database that needs to be registered in the AWR.</td>
</tr>
<tr>
<td>node_name</td>
<td>RMF node name of the database that needs to be registered in the AWR.</td>
</tr>
<tr>
<td>topology_name</td>
<td>RMF topology name of the database that needs to be registered in the AWR.</td>
</tr>
</tbody>
</table>

Examples

This example registers the remote database having the RMF node identifier of 10 in the AWR:

EXECUTE DBMS_WORKLOAD_REPOSITORY.REGISTER_REMOTE_DATABASE(10);
188.3.31 REMOVE_COLORED_SQL Procedure

This procedure removes a colored SQL ID. After a SQL is uncolored, it will no longer be captured in a snapshot automatically, unless it makes the TOP list.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.REMOVE_COLORED_SQL(
    sql_id         IN VARCHAR2,
    dbid           IN NUMBER DEFAULT NULL);
```

Parameters

Table 188-36  REMOVE_COLORED_SQL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>13-character external SQL ID</td>
</tr>
<tr>
<td>dbid</td>
<td>Optional dbid, defaults to Local DBID</td>
</tr>
</tbody>
</table>

188.3.32 RENAME_BASELINE Procedure

This procedure renames a baseline.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.RENAME_BASELINE(
    old_baseline_name     IN   VARCHAR2,
    new_baseline_name     IN VARCHAR2,
    dbid                  IN NUMBER   DEFAULT NULL);
```

Parameters

Table 188-37  RENAME_BASELINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_baseline_name</td>
<td>Old baseline name.</td>
</tr>
<tr>
<td>new_baseline_name</td>
<td>New baseline name.</td>
</tr>
<tr>
<td>dbid</td>
<td>Database ID for which the baseline needs to be renamed (defaults to local DBID).</td>
</tr>
</tbody>
</table>

188.3.33 SELECT_BASELINE_DETAILS Function

This table function shows the values of the metrics corresponding to a baseline for a range of snapshots.

This table function returns an object of AWR_BASELINE_METRIC_TYPE Object Type.
Syntax

DBMS_WORKLOAD_REPOSITORY.SELECT_BASELINE_DETAILS(
    l_baseline_id   IN  NUMBER,
    l_begin_snap    IN  NUMBER DEFAULT NULL,
    l_end_snap      IN  NUMBER DEFAULT NULL,
    l_dbid          IN  NUMBER DEFAULT NULL)
RETURN awrbl_details_type_table PIPELINED;

Parameters

Table 188-38  SELECT_BASELINE_DETAILS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_baseline_id</td>
<td>ID of the baseline for which the statistics need to be retrieved. Specifying the value 0 returns the statistics for the moving window baseline.</td>
</tr>
<tr>
<td>l_begin_snap</td>
<td>Start snapshot sequence number for the baseline.</td>
</tr>
<tr>
<td>l_end_snap</td>
<td>End snapshot sequence number for the baseline.</td>
</tr>
<tr>
<td>l_dbid</td>
<td>Database identifier for the baseline. If its value is set to NULL, then the database identifier for the local database is used. Its default value is NULL.</td>
</tr>
</tbody>
</table>

188.3.34 SELECT_BASELINE_METRIC Function

This table function shows the values of the metrics corresponding to a baseline for all the snapshots.

This table function returns an object of AWR_BASELINE_METRIC_TYPE Object Type.

Syntax

DBMS_WORKLOAD_REPOSITORY.SELECT_BASELINE_METRIC(
    l_baseline_name     IN VARCHAR2,
    l_dbid              IN NUMBER DEFAULT NULL,
    l_instance_num      IN NUMBER DEFAULT NULL)
RETURN awr_metric_type_table PIPELINED;

Parameters

Table 188-39  SELECT_BASELINE_METRIC Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_baseline_name</td>
<td>Name of the baseline for which the metrics need to be viewed.</td>
</tr>
<tr>
<td>l_dbid</td>
<td>Database identifier for the baseline. If set to NULL, the database identifier for the local database is used. Default is NULL.</td>
</tr>
<tr>
<td>l_instance_num</td>
<td>The instance number for which the metrics need to be viewed. If set to NULL, metrics for the local database instance are shown. Default is NULL.</td>
</tr>
</tbody>
</table>
188.3.35 UNREGISTER_REMOTE_DATABASE Procedures

This procedure removes all the statistics, metadata, partitions, and so on of a remote database from the Automatic Workload Repository (AWR). After executing this procedure, the remote database cannot be used for any AWR operations, such as creating remote snapshots.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.UNREGISTER_REMOTE_DATABASE(
    node_id        IN NUMBER,
    remote_check   IN BOOLEAN  DEFAULT TRUE);
```

```sql
DBMS_WORKLOAD_REPOSITORY.UNREGISTER_REMOTE_DATABASE(
    node_name      IN VARCHAR2,
    topology_name  IN VARCHAR2 DEFAULT NULL,
    remote_check   IN BOOLEAN  DEFAULT TRUE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_id</td>
<td>Identifier of the remote database whose data needs to be removed from the AWR.</td>
</tr>
<tr>
<td>node_name</td>
<td>Name of the remote database whose data needs to be removed from the AWR.</td>
</tr>
<tr>
<td>topology_name</td>
<td>RMF topology name of the remote database.</td>
</tr>
</tbody>
</table>
| remote_check    | If set to **TRUE**, the remote database’s metadata is validated before removing its data from the AWR. This option requires the remote database to be available.  
                    If set to **FALSE**, the remote database’s data is removed from the AWR without validating its metadata. This option must be selected to unregister a remote database that is not available (it is offline or there is a network outage). |

Examples

This example removes the AWR data related to the remote database having the database identifier of 10:

```sql
EXECUTE DBMS_WORKLOAD_REPOSITORY.UNREGISTER_REMOTE_DATABASE(10);
```

188.3.36 UPDATE_DATAFILE_INFO Procedure

This procedure updates the data file and tablespace information stored in the Automatic Workload Repository (AWR) with the current information in the database. This procedure is useful when a data file or a tablespace has been moved or renamed. As this change is not always captured in the next snapshot, AWR report may not show the most current information.

Syntax

```sql
DBMS_WORKLOAD_REPOSITORY.UPDATE_DATAFILE_INFO();
```
188.3.37 UPDATE_OBJECT_INFO Procedure

This procedure updates rows of WRH$ SEG STAT OBJ table that represent objects in the local database. It attempts to determine the current names for all object belonging to the local database, except those with 'MISSING' and/or 'TRANSIENT' values in the name columns.

The amount of work performed at each invocation of this routine may be controlled by setting the input parameter.

Syntax

DBMS_WORKLOAD_REPOSITORY.UPDATE_OBJECT_INFO(
    maxrows IN NUMBER DEFAULT 0);

Parameters

Table 188-41  UPDATE_OBJECT_INFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxrows</td>
<td>Maximum number of rows to be updated. Default= 0, meaning there is no limit.</td>
</tr>
</tbody>
</table>
The **DBMS_XA** package contains the XA/Open interface for applications to call XA interface in PL/SQL. Using this package, application developers can switch or share transactions across SQL*Plus sessions or processes using PL/SQL.

The chapter contains the following topics:

- Overview
- Security Model
- Constants
- Operational Notes
- Data Structures
- Summary of DBMS_XA Subprograms

**See Also:**

*Oracle Database Advanced Application Developer's Guide* for more information about "Developing Applications with Oracle XA"

### 189.1 DBMS_XA Overview

These subprograms allow a PL/SQL application to define a global transaction branch ID (XID) and associate or disassociate the current session with the transaction branch.

Subsequently, these transaction branches may be prepared and committed by following the two-phase commit protocol. A single-phase commit protocol is also supported if only one resource manager is involved.

Interfaces are also provided for a PL/SQL application to set the timeout values for any new global transaction branches that may start with the current session.

### 189.2 DBMS_XA Security Model

This package is created under **SYS**. Operations provided by this package are performed under the current calling user, not under the package owner **SYS**. Any **DBMS_XA** subprogram called from an anonymous PL/SQL block is executed using the privileges of the current user. Any **DBMS_XA** subprogram called from a stored procedure is executed using the privileges of the owner of the stored procedure.

**SELECT** or **READ** privilege on **SYS.DBA_PENDING_TRANSACTIONS** is required for users who need to execute **XA_RECOVER** subprogram.
FORCE ANY TRANSACTION privilege is required for users who need to manipulate XA transactions created by other users.

189.3 DBMS_XA Constants

The DBMS_XA package defines several constants that can be used for specifying parameter values.

The package uses the constants shown in Table 189-1 for use in the flag field of the XA_START Function and the XA_END Function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMNOFLAGS</td>
<td>PLS_INTEGER</td>
<td>00000000</td>
<td>Indicates no flag value is selected.</td>
</tr>
<tr>
<td>TMSUCCESS</td>
<td>PLS_INTEGER</td>
<td>UTL_RAW.CAST_TO_BINARY_INTEGER ('04000000')</td>
<td>Dissociates caller from transaction branch</td>
</tr>
<tr>
<td>TMJOIN</td>
<td>PLS_INTEGER</td>
<td>UTL_RAW.CAST_TO_BINARY_INTEGER ('02000000')</td>
<td>Caller is joining existing transaction branch.</td>
</tr>
<tr>
<td>TMSUSPEND</td>
<td>PLS_INTEGER</td>
<td>UTL_RAW.CAST_TO_BINARY_INTEGER ('08000000')</td>
<td>Caller is resuming association with suspended transaction branch.</td>
</tr>
</tbody>
</table>

The DBMS_XA package uses the constants shown in Table 189-2 for Possible Return Values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XA_RBBASE</td>
<td>PLS_INTEGER</td>
<td>100</td>
<td>Inclusive lower bound of the rollback codes</td>
</tr>
<tr>
<td>XA_RBROLLBACK</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE</td>
<td>Rollback was caused by an unspecified reason</td>
</tr>
<tr>
<td>XA_RBCOMMFAIL</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE+1</td>
<td>Rollback was caused by a communication failure</td>
</tr>
<tr>
<td>XA_RBDEADLOCK</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE+2</td>
<td>Deadlock was detected</td>
</tr>
<tr>
<td>XA_RBINTEGRITY</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE+3</td>
<td>Condition that violates the integrity of the resources was detected</td>
</tr>
<tr>
<td>XA_RBOTHER</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE+4</td>
<td>Resource manager rolled back the transaction for an unlisted reason</td>
</tr>
</tbody>
</table>
### Table 189-2  (Cont.) DBMS_XA Constants for Possible Return Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XA_RBPROTO</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE+5</td>
<td>Protocol error occurred in the resource manager</td>
</tr>
<tr>
<td>XA_RBTIMEOUT</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE+6</td>
<td>transaction branch took long</td>
</tr>
<tr>
<td>XA_RBTRANSIENT</td>
<td>PLS_INTEGER</td>
<td>XA_RBBASE+7</td>
<td>May retry the transaction branch</td>
</tr>
<tr>
<td>XA_RBEND</td>
<td>PLS_INTEGER</td>
<td>XA_RBTRANSIENT</td>
<td>Inclusive upper bound of the rollback codes</td>
</tr>
<tr>
<td>XA_NOMIGRATE</td>
<td>PLS_INTEGER</td>
<td>9</td>
<td>Transaction branch may have been heuristically completed</td>
</tr>
<tr>
<td>XA_HEURHAZ</td>
<td>PLS_INTEGER</td>
<td>8</td>
<td>Transaction branch may have been heuristically completed</td>
</tr>
<tr>
<td>XA_HEURCOM</td>
<td>PLS_INTEGER</td>
<td>7</td>
<td>Transaction branch has been heuristically committed</td>
</tr>
<tr>
<td>XA_HEURRRB</td>
<td>PLS_INTEGER</td>
<td>6</td>
<td>Transaction branch has been heuristically rolled back</td>
</tr>
<tr>
<td>XA_HEURMIX</td>
<td>PLS_INTEGER</td>
<td>5</td>
<td>Some of the transaction branches have been heuristically committed, others rolled back</td>
</tr>
<tr>
<td>XA_RETRY</td>
<td>PLS_INTEGER</td>
<td>4</td>
<td>Routine returned with no effect and may be re-issued</td>
</tr>
<tr>
<td>XA_RDONLY</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td>Transaction was read-only and has been committed</td>
</tr>
<tr>
<td>XA_OK</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>Normal execution</td>
</tr>
<tr>
<td>XAER_ASYNC</td>
<td>PLS_INTEGER</td>
<td>-2</td>
<td>Asynchronous operation already outstanding</td>
</tr>
<tr>
<td>XAER_RMERR</td>
<td>PLS_INTEGER</td>
<td>-3</td>
<td>Resource manager error occurred in the transaction branch</td>
</tr>
<tr>
<td>XAER_NOTA</td>
<td>PLS_INTEGER</td>
<td>-4</td>
<td>XID is not valid</td>
</tr>
<tr>
<td>XAER_INVAL</td>
<td>PLS_INTEGER</td>
<td>-5</td>
<td>Invalid arguments were given</td>
</tr>
<tr>
<td>XAER_PROTO</td>
<td>PLS_INTEGER</td>
<td>-6</td>
<td>Routine invoked in an improper context</td>
</tr>
<tr>
<td>XAER_RMFIAL</td>
<td>PLS_INTEGER</td>
<td>-7</td>
<td>Resource manager unavailable</td>
</tr>
<tr>
<td>XAER_DUPID</td>
<td>PLS_INTEGER</td>
<td>-8</td>
<td>XID already exists</td>
</tr>
<tr>
<td>XAER_OUTSIDE</td>
<td>PLS_INTEGER</td>
<td>-9</td>
<td>Resource manager doing work outside global transaction</td>
</tr>
</tbody>
</table>

### 189.4 DBMS_XA Operational Notes

In compliance with the XA specification of the X/Open CAE Standard for Distributed Transaction Processing, `XA_PREPARE/COMMIT/ROLLBACK/FORGET` may not be called when the transaction is still associated with the current session. Only after `XA_END` has been called so
that there is not any transaction associated with the current session, the application may call \texttt{XA\_PREPARE/COMMIT/ROLLBACK/FORGET}.

\texttt{XAER\_PROTO} error is returned from \texttt{XA\_PREPARE/COMMIT/ROLLBACK/FORGET} if a transaction is being associated with the current session.

Prior to calling any of the package subprograms, a connection/session must have already been established to the Oracle database server backend, or a resource manager. Resource manager identifiers are not supported. If multiple resource managers are involved, multiple connections/sessions must be pre-established to each resource manager before calling any of the package subprograms. If multiple connections/sessions are established during the course of global transaction processing, the caller must ensure that all of those connections/sessions associated with a specific global transaction branch identifier (XID) are established to the same resource manager.

189.5 DBMS\_XA Data Structures

The \texttt{DBMS\_XA} package uses this \texttt{OBJECT} type and associated \texttt{TABLE} type.

\textbf{OBJECT Types}

- \texttt{DBMS\_XA\_XID Object Type}

\textbf{TABLE Types}

- \texttt{DBMS\_XA\_XID\_ARRAY Table Type}

189.5.1 DBMS\_XA DBMS\_XA\_XID Object Type

The PL/SQL XA interface allows the PL/SQL application to define a global transaction branch id (XID) and associate/disassociate the current session with the transaction branch. XID is defined as a PL/SQL object type.

\begin{quote}
\textbf{Note:}

For more information, see "Distributed Transaction Processing: The XA Specification" in the public XA Standard.
\end{quote}

\textbf{Syntax}

\begin{verbatim}
TYPE DBMS_XA_XID IS OBJECT(
  formatid      NUMBER,
  gtrid         RAW(64),
  bqual         RAW(64),
  constructor function DBMS_XA_XID(
    gtrid     IN   NUMBER)
  RETURN SELF AS RESULT,
  constructor function DBMS_XA_XID ( 
    gtrid     IN   RAW,
    bqual     IN   RAW)
  RETURN SELF AS RESULT,
  constructor function DBMS_XA_XID( 
    formatid IN   NUMBER,

\end{verbatim}
Attributes

**Table 189-3  DBMS_XA_XID Object Type**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formatid</td>
<td>Format identifier, a number identifying different transaction managers (TM)</td>
</tr>
<tr>
<td>gtrid</td>
<td>Global transaction identifier uniquely identifying a global transaction, of which the maximum size is 64 bytes</td>
</tr>
<tr>
<td>bqual</td>
<td>Branch qualifier, of which the maximum size is 64 bytes</td>
</tr>
</tbody>
</table>

189.5.2 DBMS_XA DBMS_XA_XID_ARRAY Table Type

This type is used to define an array of `xid` that represent a list of global transaction branches.

**Syntax**

```
TYPE DBMS_XA_XID_ARRAY as TABLE of DBMS_XA_XID
```

189.6 Summary of DBMS_XA Subprograms

This table lists the DBMS_XA subprograms and briefly describes them.

**Table 189-4  DBMS_XA Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIST_TXN_SYNC Procedure</td>
<td>Used in recovery of synchronization when utilizing Oracle Real Application Clusters (Oracle RAC)</td>
</tr>
<tr>
<td>XA_COMMIT Function</td>
<td>Commits the global transaction specified by <code>xid</code></td>
</tr>
<tr>
<td>XA_END Function</td>
<td>Disassociates the current session from the transaction branch specified by <code>xid</code></td>
</tr>
<tr>
<td>XA_FORGET Function</td>
<td>Informs the resource manager to forget about a heuristically committed or rolled back transaction branch.</td>
</tr>
<tr>
<td>XA_GETLASTOER Function</td>
<td>Obtains the last Oracle error code, in case of failure of previous XA calls.</td>
</tr>
<tr>
<td>XA_PREPARE Function</td>
<td>Prepares the transaction branch specified in <code>xid</code> for committing the transaction subsequently if possible</td>
</tr>
<tr>
<td>XA_RECOVER Function</td>
<td>Obtains a list of prepared or heuristically completed transaction branches from a resource manager</td>
</tr>
<tr>
<td>XA_ROLLBACK Function</td>
<td>Informs the resource manager to roll back work done on behalf of a transaction branch</td>
</tr>
<tr>
<td>XA_SETTIMEOUT Function</td>
<td>Sets the transaction timeout in seconds for the current session</td>
</tr>
<tr>
<td>XA_START Function</td>
<td>Associates the current session with the transaction branch specified by <code>xid</code></td>
</tr>
</tbody>
</table>
189.6.1 DIST_TXN_SYNC Procedure

This procedure can be used to synchronize in-doubt transactions when one of the Oracle Real Application Clusters (Oracle RAC) instances fails.

Syntax

DBMS_XA.DIST_TXN_SYNC;

189.6.2 XA_COMMIT Function

This function commits the global transaction specified by xid.

Syntax

DBMS_XA_XID COMMIT (  
    xid IN DBMS_XA_XID,  
    onePhase IN BOOLEAN)  
RETURN PLS_INTEGER;

Parameters

Table 189-5  XA_COMMIT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>See DBMS_XA_XID Object Type</td>
</tr>
<tr>
<td>onePhase</td>
<td>If TRUE, apply single phase commit</td>
</tr>
</tbody>
</table>

Return Values

See Table 189-2. Possible return values indicating error are: XAER_RMERR, XAER_RMFAIL, XAER_NOTA, XAER_INVAL, or XAER_PROTO. Other possible return values include: XA_OK, XA_RB*, XA_HEURHAZ, XA_HEURCOM, XA_HEURRB, and XA_HEURMIX.

Usage Notes

• An application must not call COMMIT, but instead must call XA_COMMIT to commit the global transaction specified by xid. If a user needs to commit a transaction branch that is created by other users, FORCE ANY TRANSACTION must be granted to the user.

• If onePhase is TRUE, the resource manager should use a one-phase commit protocol to commit the work done on behalf of xid. Otherwise, only if all branches of the global transaction have been prepared successfully and the preceding XA_PREPARE call has returned XA_OK, should XA_COMMIT be called.

• The application must make a separate XA_COMMIT call for each of the transaction branches of the global transaction for which XA_PREPARE has returned XA_OK.

• If the resource manager did not commit the transaction and the parameter onePhase is set to TRUE, the resource manager may return one of the XA_RB* code. Upon return, the resource manager has rolled back the branch's work and has released all held resources.
189.6.3 XA_END Function

This function disassociates the current session from the transaction branch specified by \texttt{xid}. A transaction manager calls \texttt{XA\_END} when a thread of control finishes, or needs to suspend work on, a transaction branch. This occurs when the application completes a portion of its work, either partially or in its entirety (for example, before blocking on some event in order to let other threads of control work on the branch). When \texttt{XA\_END} successfully returns, the calling thread of control is no longer actively associated with the branch but the branch still exists.

Syntax

\begin{verbatim}
DBMS_XA.XA_END (  
    xid IN DBMS_XA_XID,  
    flag IN PLS_INTEGER)  
RETURN PLS_INTEGER;
\end{verbatim}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>See DBMS_XA_XID Object Type</td>
</tr>
<tr>
<td>flag</td>
<td>See Table 189-1.</td>
</tr>
</tbody>
</table>

Return Values

See Table 189-2. Possible return values in error are \texttt{XA\_RMERR, XA\_RMFAILED, XA\_NOTA, XA\_INVAL, XA\_PROTO, or XA\_RB*}.

Usage Notes

- \texttt{TMSUCCESS} or \texttt{TMSUSPEND} may be specified in flag, and the transaction branch is disassociated with the current session in detached state if the return value is \texttt{XA\_OK}. \texttt{TMFAIL} is not supported. \texttt{XA\_END} may be called with either \texttt{TMSUCCESS} or \texttt{TMSUSPEND} to disassociate the transaction branch identified by \texttt{xid} from the current session.
- \texttt{XA\_OK} is returned if \texttt{XA\_END} succeeds. An application must check the return value and handle error cases. Only when \texttt{XA\_OK} is returned, the application should proceed for other normal operations.
- Executing a \texttt{ROLLBACK} statement without calling \texttt{XA\_END} first will rollback the changes made by the current transaction. However, the transaction context is still associated with the current session until \texttt{XA\_END} is called.
- Executing a \texttt{COMMIT} statement without calling \texttt{XA\_END} first will result in ORA-02089: \texttt{COMMIT} is not allowed in a subordinate session.
- Executing a \texttt{COMMIT} or a \texttt{ROLLBACK} statement after \texttt{XA\_END} has no effect on the transaction identified by \texttt{xid}, since this transaction is no longer associated with the current session.
189.6.4 XA_FORGET Function

This function informs the resource manager to forget about a heuristically committed or rolled back transaction branch.

Syntax

```sql
DBMS_XA.XA_FORGET (
    xid IN DBMS_XA_XID
) RETURN PLS_INTEGER;
```

Parameters

Table 189-7  XA_FORGET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>See DBMS_XA_XID Object Type</td>
</tr>
</tbody>
</table>

Return Values

See Table 189-2. Possible return values are XA_OK, XAER_RMERR, XAER_RMFAIL, XAER_NOTA, XAER_INVAL, or XAER_PROTO.

189.6.5 XA_GETLASTOER Function

This function obtains the last Oracle error code, in case of failure of previous XA calls.

Syntax

```sql
DBMS_XA.XA_GETLASTOER
RETURN PLS_INTEGER;
```

Return Values

The return value carries the last Oracle error code.

189.6.6 XA_PREPARE Function

This function prepares the transaction branch specified in xid for committing the transaction subsequently if possible.

Syntax

```sql
DBMS_XA.XA_PREPARE (
    xid IN DBMS_XA_XID
) RETURN PLS_INTEGER;
```
Parameters

Table 189-8  XA_Prepare Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>See DBMS_XA_XID Object Type</td>
</tr>
</tbody>
</table>

Return Values

See Table 189-2. Possible return codes include: XA_OK, XA_RDONLY, XA_RB*, XAER_RMERR, XAER_RMFAIL, XAER_NOTA, XAER_INVAL, or XAER_PROTO.

Usage Notes

• If a user needs to prepare a transaction branch that is created by other users, FORCE ANY TRANSACTION must be granted to the user.

• An application must keep track of all the branches of one global transaction, and prepare each transaction branch. Only if all branches of the global transaction have been prepared successfully and XA_PREPARE has returned XA_OK, the application may proceed to call XA_COMMIT.

189.6.7 XA_RECOVER Function

This function obtains a list of prepared or heuristically completed transaction branches from a resource manager.

Syntax

```
DBMS_XA.XA_RECOVER
RETURN DBMS_XA_XID_ARRAY;
```

Return Values

See DBMS_XA_XID_ARRAY Table Type

Usage Notes

• The flags TMSTARTSCAN, TMMENDSCAN, TMNOFLAGS are not supported.

• The privilege SELECT ON DBA_PENDING_TRANSACTIONS must be granted to the user who needs to call XA_RECOVER.

189.6.8 XA_ROLLBACK Function

This function informs the resource manager to roll back work done on behalf of a transaction branch.

Syntax

```
DBMS_XA.XA_ROLLBACK (
    xid       IN  DBMS_XA_XID)
RETURN PLS_INTEGER;
```
### Parameters

#### Table 189-9  XA_ROLLBACK Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>See DBMS_XA_XID Object Type</td>
</tr>
</tbody>
</table>

#### Return Values

See [Table 189-2](#). Possible return values are: `XA_OK`, `XA_RB*`, `XA_HEURHAZ`, `XA_HEURCOM`, `XA_HEURRB`, or `XA_HEURMIX`.

#### Usage Notes

If a user needs to rollback a transaction branch that created by other users, the privilege `FORCE ANY TRANSACTION` must be granted to the user.

### 189.6.9 XA_SETTIMEOUT Function

This function sets the transaction timeout in seconds for the current session.

#### Syntax

```
DBMS_XA.XA_SETTIMEOUT (    
   seconds IN PLS_INTEGER)  
RETURN PLS_INTEGER;
```

#### Parameters

#### Table 189-10  XA_SETTIMEOUT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds</td>
<td>The timeout value indicates the maximum time in seconds that a transaction branch may be disassociated from the session before the system automatically aborts the transaction. The default value is 60 seconds.</td>
</tr>
</tbody>
</table>

#### Return Values

See [Table 189-2](#). Possible return values are `XA_OK`, `XAER_RMERR`, `XAER_RMFAIL`, or `XAERINVAL`.

#### Usage Notes

Only if return value is `XA_OK`, is the timeout value successfully set.
189.6.10 XA_START Function

This function associates the current session with a transaction branch specified by the xid.

Syntax

```
DBMS_XA.XA_START (
    xid   IN  DBMS_XA_XID,    flag  IN  PLS_INTEGER)  RETURN PLS_INTEGER;
```

Parameters

Table 189-11  XA_START Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>See DBMS_XA_XID Object Type</td>
</tr>
<tr>
<td>flag</td>
<td>See Table 189-1</td>
</tr>
</tbody>
</table>

Return Values

See Table 189-2

Usage Notes

- If TMJOIN or TMRESUME is specified in flag, the start is for joining an existing transaction branch identified by the xid. TMJOIN flag should be used when the transaction is detached with TMSUCCESS flag. TMRESUME should be used when the transaction branch is detached with TMSUSPEND flag. XA_START may be called with either flag to join an existing transaction branch.
- If TMNOFLAGS is specified in flag, and neither TMJOIN nor TMRESUME is specified, a new transaction branch is to be started. If the transaction branch specified in xid already exists, XA_START returns an XAER_DUPID error code.
- Possible return values in error include: XAER_RMERR, XAER_RMFAIL, XAER_DUPID, XAER_OUTSIDE, XAER_NOTA, XAERINVAL, and XAER_PROTO.
- XA_OK is returned if XA_START succeeds. An application must check the return value and handle error cases. Only when XA_OK is returned, the PL/SQL application should proceed for other normal operations. Transaction stacking is not supported. If there is an active transaction associated with the current session, may not be called to start or join another transaction. XAER_PROTO will be returned if XA_START is called with an active global transaction branch associated with the session. XAER_OUTSIDE will be returned if XA_START is called with a local transaction associated with the current session.
190

DBMS_XDB

The DBMS_XDB package supports the following features:

• Resource Management subprograms which complement Resource Views
• The Access Control List (ACL)-based Security Mechanism
• Configuration Session Management
• Creation of the XDB username

This chapter contains the following topics:

• DBMS_XDB Overview
• DBMS_XDB Security Model
• DBMS_XDB Constants
• Summary of DBMS_XDB Subprograms

See Also:

• Oracle XML DB Developer's Guide
• Oracle Database New Features Guide

190.1 DBMS_XDB Overview

DBMS_XDB provides a range of management functions.

The DBMS_XDB package supports the following features:


• The Access Control List (ACL)-based Security Mechanism can be used with in-hierarchy ACLs stored by the database or in-memory ACLs that may be stored outside the database. Some of these methods can be used for both Oracle resources and arbitrary database objects. Use CHECKPRIVILEGES Function, GETACLDOCUMENT Function, and CHANGEPRIVILEGES Function for Oracle Resources. ACLCHECKPRIVILEGES Function provides access to Oracle’s ACL-based Security mechanism without storing objects in the Hierarchy.
The XDB username is created during XDB installation. This user owns a set of
default tables and packages. `GETXDB_TABLESPACE` Function and the
`DBMS_XDB_ADMIN.MOVEXDB_TABLESPACE` Procedure enable movement of schemas
to a specified tablespace, and support the default `SYSAUX` tablespace introduction.

### 190.2 DBMS_XDB Security Model

Owned by XDB, the `DBMS_XDB` package must be created by `SYS` or `XDB`. The `EXECUTE`
privilege is granted to `PUBLIC`. Subprograms in this package are executed using the
privileges of the current user. Subprograms that operate on the XDB Configuration will
succeed only if the current user is `SYS` or `XDB`, or the current user has the `XDBADMIN` or
`DBA` role.

### 190.3 DBMS_XDB Constants

All constants described in the following table are deprecated in Oracle Release 12c.

They are relocated to either the `DBMS_XDB_CONFIG` package or the
`DBMS_XDB_REPOS` package. The specifics of transference in each case are detailed
in the `Relocated` column.

Oracle recommends that you do not use constants in their `DBMS_XDB` context in new
applications. Support for deprecated features is for backward compatibility only and
may be terminated in future releases.

#### Table 190-1 DBMS_XDB Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
<th>Relocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE_RESOURCE</td>
<td>NUMBER</td>
<td>1</td>
<td>Deletes a resource; fails if the resource has children.</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>DELETE_RECURSIVE</td>
<td>NUMBER</td>
<td>2</td>
<td>Deletes a resource and its children, if any.</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>DELETE_FORCE</td>
<td>NUMBER</td>
<td>3</td>
<td>Deletes the resource, even if the object it contains is invalid.</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>DELETE_RECURSIVE_FORCE</td>
<td>NUMBER</td>
<td>4</td>
<td>Deletes a resource and its children, if any, even if the object it contains is invalid.</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>DELETE_RES_METADATA_CASCADE</td>
<td>NUMBER</td>
<td>1</td>
<td>Deletes the corresponding row in the metadata table</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>DELETE_RES_METADATA_NOCASCADE</td>
<td>NUMBER</td>
<td>2</td>
<td>Does not delete the row in the metadata table</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>DEFAULT_LOCK_TIMEOUT_CONSTANT</td>
<td>PLS_INTEGER</td>
<td>(60*60)</td>
<td>Default time (in seconds) after which lock will expire</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>LINK_TYPE_HARD</td>
<td>NUMBER</td>
<td>1</td>
<td>Type of link to be created (default)</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>LINK_TYPE_WEAK</td>
<td>NUMBER</td>
<td>2</td>
<td>Type of link to be created</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
<tr>
<td>LINK_TYPE_SYMBOLIC</td>
<td>NUMBER</td>
<td>3</td>
<td>Type of link to be created</td>
<td><code>DBMS_XDB_REPOS</code></td>
</tr>
</tbody>
</table>
Table 190-1  DBMS_XDB Constants (Cont.)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
<th>Relocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON_DENY_NEXT_CUSTOM</td>
<td>NUMBER</td>
<td>1</td>
<td>If access denied, the next custom authorization is tried</td>
<td>DBMS_XDB_C_ONFIG</td>
</tr>
<tr>
<td>ON_DENY_BASIC</td>
<td>NUMBER</td>
<td>2</td>
<td>If access denied, basic authentication is used</td>
<td>DBMS_XDB_C_ONFIG</td>
</tr>
</tbody>
</table>

190.4 Summary of DBMS_XDB Subprograms

This table lists the DBMS_XDB subprograms and briefly describes them.

Table 190-2  DBMS_XDB Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLCHECKPRIVILEGES Function</td>
<td>Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the ‘owner’ parameter.</td>
</tr>
<tr>
<td>APPENDRESOURCEMETADATA Procedure</td>
<td>Takes in user-defined metadata either as a REF to XMLTYPE or an XMLTYPE and adds it to the desired resource.</td>
</tr>
<tr>
<td>CHANGEOWNER Procedure</td>
<td>Changes the owner of the resource/s to the specified owner.</td>
</tr>
<tr>
<td>CHANGEPRIVILEGES Function</td>
<td>Adds a specified ACE to a specified resource’s ACL.</td>
</tr>
<tr>
<td>CHECKPRIVILEGES Function</td>
<td>Checks access privileges granted to the current user on the specified resource.</td>
</tr>
<tr>
<td>CREATEFOLDER Function</td>
<td>Creates a new folder resource in the hierarchy.</td>
</tr>
<tr>
<td>CREATEOIDPATH Function</td>
<td>Creates a virtual path to the resource based on object ID.</td>
</tr>
<tr>
<td>CREATETERESOURCE Functions</td>
<td>Creates a new resource.</td>
</tr>
<tr>
<td>DELETERESOURCE Procedure</td>
<td>Deletes a resource from the hierarchy.</td>
</tr>
<tr>
<td>DELETERESOURCEMETADATA Procedures</td>
<td>Deletes metadata from a resource (can be used for schema-based or nonschema-based metadata).</td>
</tr>
<tr>
<td>ENABLEDIGESTAUTHENTICATION Procedure</td>
<td>Enables digest authentication.</td>
</tr>
<tr>
<td>EXISTSRESOURCE Function</td>
<td>Determines if a resource is the hierarchy, based on its absolute path.</td>
</tr>
<tr>
<td>GETACLDOCUMENT Function</td>
<td>Retrieves ACL document that protects resource given its path name.</td>
</tr>
<tr>
<td>GETCONTENTBLOB Function</td>
<td>Retrieves the contents of a resource returned as a BLOB.</td>
</tr>
<tr>
<td>GETCONTENTCLOB Function</td>
<td>Retrieves the contents of a resource returned as a CLOB.</td>
</tr>
<tr>
<td>GETCONTENTVARCHAR2 Function</td>
<td>Retrieves the contents of a resource returned as a string.</td>
</tr>
<tr>
<td>GETCONTENTXMLREF Function</td>
<td>Retrieves the contents of a resource returned as a REF to an XMLTYPE.</td>
</tr>
<tr>
<td>GETCONTENTXMLTYPE Function</td>
<td>Retrieves the contents of a resource returned as an XMLTYPE.</td>
</tr>
</tbody>
</table>
### Table 190-2  (Cont.) DBMS_XDB Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETHTTPREQUESTHEADER Function</td>
<td>Gets the values of the passed header</td>
</tr>
<tr>
<td>GETLOCKTOKEN Procedure</td>
<td>Returns that resource's lock token for the current user given a path to a resource</td>
</tr>
<tr>
<td>GETPRIVILEGES Function</td>
<td>Gets all privileges granted to the current user on a specified resource</td>
</tr>
<tr>
<td>GETRESOID Function</td>
<td>Returns the object ID of the resource from its absolute path</td>
</tr>
<tr>
<td>GETXDB_TABLESPACE Function</td>
<td>Returns the current tablespace of the XDB (user)</td>
</tr>
<tr>
<td>HASBLOBCONTENT Function</td>
<td>Returns <code>TRUE</code> if the resource has BLOB content</td>
</tr>
<tr>
<td>HASCHARCONTENT Function</td>
<td>Returns <code>TRUE</code> if the resource has character content</td>
</tr>
<tr>
<td>HASXMLCONTENT Function</td>
<td>Returns <code>TRUE</code> if the resource has XML content</td>
</tr>
<tr>
<td>HASXMLREFERENCE Function</td>
<td>Returns <code>TRUE</code> if the resource has <code>REF</code> to XML content</td>
</tr>
<tr>
<td>ISFOLDER Function</td>
<td>Returns <code>TRUE</code> if the resource is a folder or container</td>
</tr>
<tr>
<td>LINK Procedures</td>
<td>Creates a link to an existing resource</td>
</tr>
<tr>
<td>LOCKRESOURCE Function</td>
<td>Gets a WebDAV-style lock on that resource given a path to that resource</td>
</tr>
<tr>
<td>PROCESSLINKS Procedure</td>
<td>Processes document links in the specified resource</td>
</tr>
<tr>
<td>PURGERESOURCEMETADATA Procedure</td>
<td>Deletes all user metadata from a resource</td>
</tr>
<tr>
<td>RENAMERESOURCE Procedure</td>
<td>Renames the XDB resource</td>
</tr>
<tr>
<td>SETACL Procedure</td>
<td>Sets the ACL on a specified resource</td>
</tr>
<tr>
<td>SPLITPATH Procedure</td>
<td>Splits the path into a parent path and childpath</td>
</tr>
<tr>
<td>TOUCHRESOURCE Procedure</td>
<td>Changes the modification time of the resource to the current time</td>
</tr>
<tr>
<td>UNLOCKRESOURCE Function</td>
<td>Unlocks the resource given a lock token and resource path</td>
</tr>
<tr>
<td>UPDATERESOURCEMETADATA Procedures</td>
<td>Updates metadata for a resource</td>
</tr>
</tbody>
</table>

### 190.4.1 ACLCHECKPRIVILEGES Function

This function checks access privileges granted to the current user by specified ACL document by the `OWNER` of the resource. Returns positive integer if all privileges are granted.

**Note:**

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the `DBMS_XDB_REPOS` package - the `ACLCHECKPRIVILEGES` Function.
Syntax

DBMS_XDB.ALCHECKPRIVILEGES(
    acl_path  IN  VARCHAR2,
    owner     IN  VARCHAR2,
    privs     IN  xmltype)
RETURN PLS_INTEGER;

Parameters

Table 190-3 ALCHECKPRIVILEGES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_path</td>
<td>Absolute path in the Hierarchy for ACL document</td>
</tr>
<tr>
<td>owner</td>
<td>Resource owner name; the pseudo user “DAV:owner” is replaced by this user during ACL privilege resolution</td>
</tr>
<tr>
<td>privs</td>
<td>An XMLType instance of the privilege element specifying the requested set of access privileges. See description for CHECKPRIVILEGES Function.</td>
</tr>
</tbody>
</table>

190.4.2 APPENDRESOURCEMETADATADATA Procedure

This procedure takes in user-defined metadata either as a REF to XMLTYPE or an XMLTYPE and adds it to the desired resource.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the APPENDRESOURCEMETADATADATA Procedure.

Syntax

DBMS_XDB.APPENDRESOURCEMETADATADATA (  
    abspath   IN  VARCHAR2,
    metadata  IN  XMLTYPE);      

DBMS_XDB.APPENDRESOURCEMETADATADATA (  
    abspath   IN  VARCHAR2,
    metadata  IN  REF SYS.XMLTYPE);      

Parameters

Table 190-4 APPENDRESOURCEMETADATADATA Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>metadata</td>
<td>Metadata can be schema based or nonschema-based. Schema-based metadata is stored in its own table.</td>
</tr>
</tbody>
</table>
Usage Notes

- In the case in which a REF is passed in, the procedure stores the REF in the resource, and the metadata is stored in a separate table. In this case you are responsible for populating the RESID column for the metadata table. Note that the REF passed in must be unique. In other words, there must not be a REF with the same value in the resource metadata, as this would violate uniqueness of properties. An error is thrown if users attempt to add a REF that already exists.

- In the case where the XMLTYPE is passed in, the data is parsed to determine if it is schema-based or not and stored accordingly.

190.4.3 CHANGEOWNER Procedure

This procedure changes the owner of the resource/s to the specified owner. This procedure is deprecated in Release 12c.

![Note:]

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the CHANGEOWNER Procedure.

Syntax

```sql
DBMS_XDB.CHANGEOWNER(
    abspath    IN   VARCHAR2,
    owner      IN   VARCHAR2,
    recurse    IN   BOOLEAN := FALSE);
```

Parameters

Table 190-5  CHANGEOWNER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>owner</td>
<td>New owner for the resource</td>
</tr>
<tr>
<td>recurse</td>
<td>If TRUE, recursively change owner of all resources in the folder tree</td>
</tr>
</tbody>
</table>
190.4.4 CHANGEPRIVILEGES Function

This function adds a specified ACE to a specified resource's ACL. This procedure is deprecated in Release 12c.

![Note:]

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the CHANGEPRIVILEGES Function.

Syntax

```
DBMS_XDB.CHANGEPRIVILEGES(
    res_path   IN    VARCHAR2,
    ace        IN    xmltype)
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Path name of the resource for which privileges need to be changed</td>
</tr>
<tr>
<td>ace</td>
<td>An XMLType instance of the &lt;ace&gt; element which specifies the &lt;principal&gt;, the operation &lt;grant&gt; and the list of privileges</td>
</tr>
</tbody>
</table>

Return Values

A positive integer if the ACL was successfully modified.

Usage Notes

If no ACE with the same principal and the same operation (grant/deny) already exists in the ACL, the new ACE is added at the end of the ACL.

190.4.5 CHECKPRIVILEGES Function

This function checks access privileges granted to the current user on the specified resource.

![Note:]

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the CHECKPRIVILEGES Function.
Syntax

DBMS_XDB.CHECKPRIVILEGES(
    res_path IN VARCHAR2,
    privs IN xmltype
) RETURN PLS_INTEGER;

Parameters

Table 190-7  CHECKPRIVILEGES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Absolute path in the Hierarchy for resource</td>
</tr>
<tr>
<td>privs</td>
<td>An XMLType instance of the privilege element specifying the requested set of access privileges</td>
</tr>
</tbody>
</table>

Return Values

A positive integer if all requested privileges granted.

190.4.6 CREATEFOLDER Function

This deprecated function creates a new folder resource in the hierarchy.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the CREATEFOLDER Function.

Syntax

DBMS_XDB.CREATEFOLDER(
    path IN VARCHAR2
) RETURN BOOLEAN;

Parameters

Table 190-8  CREATEFOLDER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name for the new folder</td>
</tr>
</tbody>
</table>

Return Values

TRUE if operation successful; FALSE, otherwise.
Usage Notes

The given path name's parent folder must already exist in the hierarchy: if '/folder1/folder2' is passed as the path parameter, then '/folder1' must already exist.

190.4.7 CREATEOIDPATH Function

This deprecated function creates a virtual path to the resource based on object ID.

Note:

This function is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the CREATEOIDPATH Function.

Syntax

DBMS_XDB.CREATEOIDPATH(
    oid IN RAW
) RETURN VARCHAR2;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>Object ID of the resource</td>
</tr>
</tbody>
</table>

190.4.8 CREATERESOURCE Functions

The deprecated function creates a new resource. The description of the overload options precede each version of the syntax.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the CREATERESOURCE Functions.

Syntax

Given a REF to an existing XMLType row, creates a resource whose contents point to that row. That row should not already exist inside another resource:

DBMS_XDB.CREATERESOURCE(
    abspath IN VARCHAR2,
    datarow IN REF SYS.XMLTYPE,
)
CREATERESOURCEx IN BOOLEAN := FALSE)
RETURN BOOLEAN;

Creates a resource with a specified BLOB as its contents, and specifies character set of
the source BLOB:

DBMS_XDB.CREATERESOURCEx
-abspath IN VARCHAR2,
data IN BLOB,
csid IN NUMBER :=0,
-createfolders IN BOOLEAN := FALSE)
RETURN BOOLEAN;

Creates a resource with a specified BFILE as its contents, and specifies character set of
the source BFILE:

DBMS_XDB.CREATERESOURCEx
-abspath IN VARCHAR2,
data IN BFILE,
csid IN NUMBER :=0,
-createfolders IN BOOLEAN := FALSE)
RETURN BOOLEAN;

Creates a resource with a specified CLOB as its contents:

DBMS_XDB.CREATERESOURCEx
-abspath IN VARCHAR2,
data IN CLOB,
-createfolders IN BOOLEAN := FALSE)
RETURN BOOLEAN;

Given a string, inserts a new resource into the hierarchy with the string as the contents:

DBMS_XDB.CREATERESOURCEx
-abspath IN VARCHAR2,
data IN VARCHAR2,
schemaurl IN VARCHAR2 := NULL,
elem IN VARCHAR2 := NULL)
RETURN BOOLEAN;

Given an XMLTYPE and a schema URL, inserts a new resource into the hierarchy with
the XMLTYPE as the contents:

DBMS_XDB.CREATERESOURCEx
-abspath IN VARCHAR2,
data IN SYS.XMLTYPE,
schemaurl IN VARCHAR2 := NULL,
elem IN VARCHAR2 := NULL)
RETURN BOOLEAN;

Parameters

Table 190-10  CREATERESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource to create. The path name’s parent folder must already exist in the hierarchy. In other words, if /foo/bar.txt is passed in, then folder /foo must already exist.</td>
</tr>
</tbody>
</table>
Table 190-10  (Cont.) CREATETRERESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>String buffer containing new resource's contents. The data is parsed to check if it contains a schema-based XML document, and the contents are stored as schema-based in the schema's default table. Otherwise, it is saved as binary data.</td>
</tr>
<tr>
<td>datarow</td>
<td>REF to an XMLType row to be used as the contents</td>
</tr>
<tr>
<td>csid</td>
<td>Character set id of the document. Must be a valid Oracle ID; otherwise returns an error. If CSID is not specified, or if a zero CSID is specified, then the character set id of the document is determined as follows:</td>
</tr>
<tr>
<td></td>
<td>• From the abspath extension, determine the resource's MIME type.</td>
</tr>
<tr>
<td></td>
<td>• If the MIME type is */xml, then the encoding is detected based on Appendix F of the W3C XML 1.0 Reference at <a href="http://www.w3.org/TR/2000/REC-xml-20001006">http://www.w3.org/TR/2000/REC-xml-20001006</a>;</td>
</tr>
<tr>
<td></td>
<td>• Otherwise, it is defaulted to the database character set.</td>
</tr>
<tr>
<td>createfolders</td>
<td>If TRUE, create the parent folders if they do not exist</td>
</tr>
<tr>
<td>schemaurl</td>
<td>For XML data, schema URL data conforms to (default NULL)</td>
</tr>
<tr>
<td>elem</td>
<td>Element name (default NULL)</td>
</tr>
</tbody>
</table>

Return Values

TRUE if operation successful; FALSE, otherwise.

190.4.9 DELETERESOURCE Procedure

This deprecated procedure deletes a resource from the hierarchy.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the DELETERESOURCE Procedure.

Syntax

```sql
DBMS_XDB.DELETERESOURCE(
    path          IN      VARCHAR2,
    delete_option IN      PLS_INTEGER);
```

Parameters

Table 190-11  DELETERESOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name of the resource to delete</td>
</tr>
</tbody>
</table>
### Table 190-11  (Cont.) DELETERESOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete_option</td>
<td>The option that controls how a resource is deleted; defined in Table 190-1:</td>
</tr>
<tr>
<td></td>
<td>• DELETE_RESOURCE</td>
</tr>
<tr>
<td></td>
<td>• DELETE_RECURSIVE</td>
</tr>
<tr>
<td></td>
<td>• DELETE_FORCE</td>
</tr>
<tr>
<td></td>
<td>• DELETE_RECURSIVE_FORCE</td>
</tr>
</tbody>
</table>

### 190.4.10 DELETERESOURCEMETADATA Procedures

This *deprecated procedure* takes in a resource by absolute path and removes either the schema-based metadata identified by the REF, or the metadata identified by the namespace and name combination, which can be either schema-based or non-schema based. It also takes an additional (optional) parameter that specifies how to delete it. This parameter is only relevant for schema-based resource metadata that needs to be deleted. For non-schema based metadata, this parameter is ignored.

**Note:**

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the DELETERESOURCEMETADATA Procedures.

**Syntax**

Can be used only for schema-based metadata:

```sql
DBMS_XDB.DELETERESOURCEMETADATA (  
    abspath        IN  VARCHAR2,  
    metadata       IN  REF SYS.XMLTYPE,  
    delete_option  IN  pls_integer := dbms_xdb.DELETE_RESOURCE_METADATA_CASCADE);
```

Can be used for schema-based or nonschema-based metadata:

```sql
DBMS_XDB.DELETERESOURCEMETADATA (  
    abspath        IN  VARCHAR2,  
    metadatans     IN  VARCHAR2,  
    metadataname   IN  VARCHAR2,  
    delete_option  IN  pls_integer := dbms_xdb.DELETE_RESOURCE_METADATA_CASCADE);
```

**Parameters**

### Table 190-12  DELETERESOURCEMETADATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>metadata</td>
<td>REF to the piece of metadata (schema based) to be deleted</td>
</tr>
</tbody>
</table>
Table 190-12  (Cont.) DELETHERESOURCEMETADATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mettadatans</td>
<td>Namespace of the metadata fragment to be removed</td>
</tr>
<tr>
<td>mettadataname</td>
<td>Local name of the metadata fragment to be removed</td>
</tr>
<tr>
<td>delete_option</td>
<td>Only applicable for schema-based metadata, this can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>• DELETE_RES_METADATA_CASCADE - deletes the corresponding row in the metadata table</td>
</tr>
<tr>
<td></td>
<td>• DELETE_RES_METADATA_NOCASCADE - does not delete the row in the metadata table</td>
</tr>
</tbody>
</table>

190.4.11 ENABLEDIGESTAUTHENTICATION Procedure

This deprecated procedure enabling digest authentication. It will list digest as the first authentication mechanism to be used by the XML DB HTTP server.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_CONFIG package - the ENABLEDIGESTAUTHENTICATION Procedure.

Syntax

DBMS_XDB.ENABLEDIGESTAUTHENTICATION;

190.4.12 EXISTSRESOURCE Function

This deprecated function indicates if a resource is in the hierarchy. Matches resource by a string that represents its absolute path.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the EXISTSRESOURCE Function.

Syntax

DBMS_XDB.EXISTSRESOURCE(
    abspath    IN    VARCHAR2)
RETURN BOOLEAN;
Parameters

Table 190-13  EXISTSRESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Path name of the resource whose ACL document is required</td>
<td></td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource is found.

190.4.13 GETACLDOCUMENT Function

This deprecated function retrieves ACL document that protects resource given its path name.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETACLDOCUMENT Function.

Syntax

DBMS_XDB.GETACLDOCUMENT(
    abspath IN VARCHAR2
) RETURN sys.xmltype;

Parameters

Table 190-14  GETACLDOCUMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Path name of the resource whose ACL document is required</td>
<td></td>
</tr>
</tbody>
</table>

Return Values

The XMLType for ACL document.
190.4.14 GETCONTENTBLOB Function

This deprecated function retrieves the contents of a resource returned as a BLOB.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETCONTENTBLOB Function.

Syntax

```sql
DBMS_XDB.GETCONTENTBLOB(
    abspath    IN     VARCHAR2,
    csid       OUT    PLS_INTEGER,
    locksrc    IN     BOOLEAN := FALSE)
RETURN BLOB;
```

Parameters

Table 190-15 GETCONTENTBLOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>csid</td>
<td>If TRUE, lock and return the source LOB. If FALSE, return a temp LOB copy.</td>
</tr>
<tr>
<td>locksrc</td>
<td>Contents of the resource as a BLOB</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a BLOB.

190.4.15 GETCONTENTCLOB Function

This deprecated function gets the contents of a resource returned as a CLOB.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETCONTENTCLOB Function.

Syntax

```sql
DBMS_XDB.GETCONTENTCLOB(
    abspath    IN     VARCHAR2,
RETURN CLOB;
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a CLOB.

190.4.16 GETCONTENTVARCHAR2 Function

This deprecated function gets the contents of a resource returned as a string.

**Note:**

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETCONTENTVARCHAR2 Function.

Syntax

```sql
DBMS_XDB.GETCONTENTVARCHAR2(
   abspath IN VARCHAR2,
   RETURN BLOB;
)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a string.
190.4.17 GETCONTENTXMLREF Function

This deprecated function retrieves the contents of a resource returned as a REF to an XMLTYPE.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETCONTENTXMLREF Function.

Syntax

DBMS_XDB.GETCONTENTXMLREF(
    abspath    IN     VARCHAR2,
    RETURN SYS.XMLTYPE;
)

Parameters

Table 190-18  GETCONTENTXMLREF Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a REF to an XMLTYPE.

190.4.18 GETCONTENTXMLTYPE Function

This deprecated function retrieves the contents of a resource returned as an XMLTYPE.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETCONTENTXMLTYPE Function.

Syntax

DBMS_XDB.GETCONTENTXMLTYPE(
    abspath    IN     VARCHAR2,
    RETURN SYS.XMLTYPE;
)
Parameters

Table 190-19  GETCONTENTXMLTYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as an XMLTYPE.

190.4.19 GETHTTPREQUESTHEADER Function

This deprecated function, if called during an HTTP request serviced by XDB, returns the values of the passed header. It is used by routines that implement custom authentication.

_note:_

This procedure is deprecated in Release 12c.

Syntax

```sql
DBMS_XDB.GETHTTPREQUESTHEADER(
    header_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table 190-20  GETHTTPREQUESTHEADER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header_name</td>
<td>Passed header</td>
</tr>
</tbody>
</table>

Return Values

Returns NULL in case the header is not present in the request, or for AUTHENTICATION, for security reasons.
190.4.20 GETLOCKTOKEN Procedure

Given a path to a resource, this deprecated procedure returns that resource's lock token for the current user.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETLOCKTOKEN Procedure.

Syntax

```
DBMS_XDB.GETLOCKTOKEN(
    path         IN      VARCHAR2,
    locktoken    OUT     VARCHAR2);
```

Parameters

Table 190-21  GETLOCKTOKEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name to the resource</td>
</tr>
<tr>
<td>locktoken</td>
<td>Logged-in user's lock token</td>
</tr>
<tr>
<td></td>
<td>for the resource</td>
</tr>
</tbody>
</table>

Usage Notes

The user must have READPROPERTIES privilege on the resource.

190.4.21 GETPRIVILEGES Function

This deprecated function gets all privileges granted to the current user on a specified resource.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETPRIVILEGES Function.

Syntax

```
DBMS_XDB.GETPRIVILEGES(
    res_path    IN     VARCHAR2
) RETURN sys.xmltype;
```
Parameters

Table 190-22 GETPRIVILEGES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Absolute path in the hierarchy of the resource</td>
</tr>
</tbody>
</table>

Return Values

An XMLType instance of `<privilege>` element, which contains the list of all leaf privileges granted on this resource to the current user.

190.4.22 GETRESOID Function

This deprecated procedure returns the object ID of the resource from its absolute path.

**Note:**

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETRESOID Function.

Syntax

```sql
DBMS_XDB.GETRESOID(
    abspath IN VARCHAR2
) RETURN RAW;
```

Parameters

Table 190-23 GETRESOID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

NULL if the resource is not present.
190.4.23 GETXDB_TABLESPACE Function

This deprecated function returns the current tablespace of the XDB (user).

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the GETXDB_TABLESPACE Function.

Syntax

```sql
DBMS_XDB.GETXDB_TABLESPACE
RETURN VARCHAR2;
```

190.4.24 HASBLOBCONTENT Function

This deprecated function returns TRUE if the resource has BLOB content.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the HASBLOBCONTENT Function.

Syntax

```sql
DBMS_XDB.HASBLOBCONTENT
    abspath    IN     VARCHAR2)
RETURN BOOLEAN;
```

Parameters

Table 190-24  HASBLOBCONTENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource has BLOB content.
190.4.25 HASCHARCONTENT Function

This deprecated function returns \texttt{TRUE} if the resource has character content.

\textbf{Note:}

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the \texttt{DBMS_XDB_REPOS} package - the \texttt{HASCHARCONTENT} Function.

\textbf{Syntax}

\texttt{DBMS_XDB.HASCHARCONTENT}\[\texttt{abspath \hspace{1em} IN \hspace{1em} VARCHAR2})\]RETURN BOOLEAN;

\textbf{Parameters}

\textbf{Table 190-25} \texttt{HASCHARCONTENT} Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{abspath_path}</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

\textbf{Return Values}

\texttt{TRUE} if the \texttt{resource} has character content.

190.4.26 HASXMLCONTENT Function

This deprecated function returns \texttt{TRUE} if the resource has XML content.

\textbf{Note:}

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the \texttt{DBMS_XDB_REPOS} package - the \texttt{HASXMLCONTENT} Function.

\textbf{Syntax}

\texttt{DBMS_XDB.HASXMLCONTENT}\[\texttt{abspath \hspace{1em} IN \hspace{1em} VARCHAR2})\]RETURN BOOLEAN;
Parameters

Table 190-26    HASXMLCONTENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource has XML content.

190.4.27 HASXMLREFERENCE Function

This deprecated function returns TRUE if the resource has a REF to XML content.

>Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the HASXMLREFERENCE Function.

Syntax

DBMS_XDB.HASXMLREFERENCE
    (abspath IN VARCHAR2)
RETURN BOOLEAN;

Parameters

Table 190-27    HASXMLREFERENCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE resource has a REF to XML content.
190.4.28 ISFOLDER Function

This deprecated function returns TRUE if the resource is a folder or container.

```
Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the ISFOLDER Function.
```

Syntax

```sql
DBMS_XDB.ISFOLDER
    abspath IN VARCHAR2)
RETURN BOOLEAN;
```

Parameters

Table 190-28  DBMS_XDB.ISFOLDER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource is a folder or container.

190.4.29 LINK Procedures

This deprecated procedure creates a link from a specified folder to a specified resource.

```
Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the LINK Procedures.
```

Syntax

```sql
DBMS_XDB.LINK(
    srcpath      IN   VARCHAR2,
    linkfolder   IN   VARCHAR2,
    linkname     IN   VARCHAR2);
```

```sql
DBMS_XDB.LINK(
    srcpath      IN   VARCHAR2,
    linkfolder   IN   VARCHAR2,
```
Parameters

Table 190-29  LINK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcpath</td>
<td>Path name of the resource to which a link is created</td>
</tr>
<tr>
<td>linkfolder</td>
<td>Folder in which the new link is placed</td>
</tr>
<tr>
<td>linename</td>
<td>Name of the new link</td>
</tr>
<tr>
<td>linktype</td>
<td>Type of link to be created:</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDB.LINK_TYPE_HARD (default)</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDB.LINK_TYPE_WEAK</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDB.LINK_TYPE_SYMBOLIC</td>
</tr>
</tbody>
</table>

190.4.30 LOCKRESOURCE Function

Given a path to a resource, this deprecated function gets a WebDAV-style lock on that resource.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the LOCKRESOURCE Function.

Syntax

```sql
DBMS_XDB.LOCKRESOURCE(
    path      IN  VARCHAR2,
    depthzero IN  BOOLEAN,  
    shared    IN  boolean)
RETURN BOOLEAN;
```

Parameters

Table 190-30  LOCKRESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name of the resource to lock.</td>
</tr>
<tr>
<td>depthzero</td>
<td>Currently not supported</td>
</tr>
<tr>
<td>shared</td>
<td>Passing TRUE obtains a shared write lock</td>
</tr>
</tbody>
</table>

Return Values

TRUE if successful.
Usage Notes
The user must have UPDATE privileges on the resource.

190.4.31 PROCESSLINKS Procedure

This deprecated procedure processes document links in the specified resource.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the PROCESSLINKS Procedure.

Syntax

```sql
DBMS_XDB.PURGERESOURCEMETADATA(
    abspath  IN  VARCHAR2,
    recurse  IN  BOOLEAN := FALSE);
```

Parameters

Table 190-31  PROCESSLINKS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource. If the path is a folder, use the recurse flag.</td>
</tr>
<tr>
<td>recurse</td>
<td>Used only if abspath specifies a folder. If TRUE, process links of all resources in the folder hierarchy rooted at the specified resource. If FALSE, process links of all documents in this folder only.</td>
</tr>
</tbody>
</table>

190.4.32 PURGERESOURCEMETADATADATA Procedure

This deprecated procedure deletes all user metadata from a resource. Schema-based metadata is removed in cascade mode, rows being deleted from the corresponding metadata tables.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the PURGERESOURCEMETADATADATA Procedure.

Syntax

```sql
DBMS_XDB.PURGERESOURCEMETADATADATA(
    abspath  IN  VARCHAR2);
```
### Parameters

**Table 190-32  PURGERESOURCEMETADATA Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

### 190.4.33 RENAMERESOURCE Procedure

This deprecated procedure renames the XDB resource.

**Note:**
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the RENAMERESOURCE Procedure.

**Syntax**

```sql
DBMS_XDB.RENAMERESOURCE(
    srcpath    IN  VARCHAR2,
    destfolder IN  VARCHAR2,
    newname    IN  VARCHAR2);
```

**Parameters**

**Table 190-33  RENAMERESOURCE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcpath</td>
<td>Absolute path in the Hierarchy for the source resource destination folder</td>
</tr>
<tr>
<td>destfolder</td>
<td>Absolute path in the Hierarchy for the destination folder</td>
</tr>
<tr>
<td>newname</td>
<td>Name of the child in the destination folder</td>
</tr>
</tbody>
</table>

### 190.4.34 SETACL Procedure

This deprecated procedure set the ACL on a specified resource to be the ACL specified by path.

**Note:**
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the SETACL Procedure.
Syntax

DBMS_XDB.SETACL(
    res_path IN VARCHAR2,
    acl_path IN VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Absolute path in the Hierarchy for resource</td>
</tr>
<tr>
<td>acl_path</td>
<td>Absolute path in the Hierarchy for ACL</td>
</tr>
</tbody>
</table>

Usage Notes

The user must have `<write-acl>` privileges on the resource.

190.4.35 SPLITPATH Procedure

This deprecated procedure splits the path into a parentpath and childpath.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the SPLITPATH Procedure.

Syntax

DBMS_XDB.SPLITPATH(
    abspath     IN  VARCHAR2,
    parentpath  OUT VARCHAR2,
    childpath   OUT VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path to be split</td>
</tr>
<tr>
<td>parentpath</td>
<td>Parentpath</td>
</tr>
<tr>
<td>childpath</td>
<td>Childpath</td>
</tr>
</tbody>
</table>
190.4.36 TOUCHRESOURCE Procedure

This deprecated procedure changes the modification time of the resource to the current time.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the TOUCHRESOURCE Procedure.

Syntax

```
DBMS_XDB.TOUCHRESOURCE
    abspath    IN     VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

190.4.37 UNLOCKRESOURCE Function

This deprecated function unlocks the resource given a lock token and a path to the resource.

Note:
This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the UNLOCKRESOURCE Function.

Syntax

```
DBMS_XDB.UNLOCKRESOURCE(
    path     IN  VARCHAR2,
    deltoken IN  VARCHAR2)
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name to the resource</td>
</tr>
</tbody>
</table>
Table 190-37 (Cont.) UNLOCKRESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deltoken</td>
<td>Lock token to be removed</td>
</tr>
</tbody>
</table>

Return Values

TRUE if operation successful.

Usage Notes

The user must have UPDATE privileges on the resource.

190.4.38 UPDATERESOURCEMETADATAN Procedures

This deprecated procedure updates metadata for a resource.

Note:

This procedure is deprecated in Release 12c. This functionality is replaced by a subprogram of the same name in the DBMS_XDB_REPOS package - the UPDATERESOURCEMETADATAN Procedures.

The procedure takes in a resource identified by absolute path and the metadata in it to replace identified by its REF. It replaces that piece of metadata with user-defined metadata which is either in the form of a REF to XMLTYPE or an XMLTYPE.

Syntax

Can be used to update schema-based metadata only. The new metadata must be schema-based:

```sql
DBMS_XDB.UPDATERESOURCEMETADATAN(
    abspath  IN VARCHAR2,
    oldmetadata    IN   REF SYS.XMLTYPE,
    newmetadata    IN   REF SYS.XMLTYPE)
```

Can be used to update schema-based metadata only. The new metadata must be schema-based or nonschema-based:

```sql
DBMS_XDB.UPDATERESOURCEMETADATAN(
    abspath        IN   VARCHAR2,
    oldmetadata    IN   REF SYS.XMLTYPE,
    newmetadata    IN   XMLTYPE);
```

Can be used for both schema-based and nonschema-based metadata:

```sql
DBMS_XDB.UPDATERESOURCEMETADATAN(
    abspath        IN   VARCHAR2,
    oldns          IN   VARCHAR2,
    oldname IN VARCHAR,
    ...
```
Can be used for both schema-based or nonschema-based metadata. New metadata must be schema-based:

```sql
DBMS_XDB.UPDATERESOURCEMETADATA(
    abspath       IN   VARCHAR2,
    oldns         IN   VARCHAR2,
    oldname       IN   VARCHAR,
    newmetadata   IN   REF SYS.XMLTYPE);
```

### Parameters

**Table 190-38**  
**UPDATERESOURCEMETADATA Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>oldmetadata</td>
<td>REF to the old of metadata</td>
</tr>
<tr>
<td>newmetadata</td>
<td>REF to the new, replacement metadata (can be either schema-based or nonschema-based depending on the overload)</td>
</tr>
<tr>
<td>oldns</td>
<td>Namespace identifying old metadata</td>
</tr>
<tr>
<td>oldname</td>
<td>Local name identifying old metadata</td>
</tr>
</tbody>
</table>

### Usage Notes

In the case of **REF**, it stores the **REF** in the resource and the metadata is stored in a separate table. Uniqueness of **REFs** is enforced. In the case where the **XMLTYPE** is passed in, data is parsed to determine if it is schema-based or not and is stored accordingly.
The DBMS_XDB_ADMIN package provides an interface to manage the Oracle XML DB repository.

This chapter contains the following topics:

- DBMS_XDB_ADMIN Security Model
- Summary of DBMS_XDB_ADMIN Subprograms

See Also:
Oracle XML DB Developer’s Guide for information about Oracle XML DB Repository

191.1 DBMS_XDB_ADMIN Security Model

Owned by XDB, the DBMS_XDB_ADMIN package must be created by SYS or XDB. The EXECUTE privilege is granted to SYS or XDB or DBA. Subprograms in this package are executed using the privileges of the current user.

191.2 Summary of DBMS_XDB_ADMIN Subprograms

This table lists the package subprograms in alphabetical order.

Table 191-1  DBMS_XDB_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATENONCEKEY Procedure</td>
<td>Generates a nonce value for use in digest authentication</td>
</tr>
<tr>
<td>INSTALLDEFAULTWALLET Procedure</td>
<td>Installs the default XDB wallet in the default XDB wallet directory</td>
</tr>
<tr>
<td>MOVEXDB_TABLESPACE Procedure</td>
<td>Moves the XDB (user) to the specified tablespace</td>
</tr>
<tr>
<td>REBUILDHIERARCHICALINDEX Procedure</td>
<td>Rebuilds the hierarchical index after import or export operations</td>
</tr>
</tbody>
</table>

191.2.1 CREATENONCEKEY Procedure

This procedure generates a nonce value for use in digest authentication.

Syntax

```
DBMS_XDB_ADMIN.CREATENONCEKEY;
```
191.2.2 INSTALLDEFAULTWALLET Procedure

This procedure installs the default XDB wallet in the default XDB wallet directory.

The directory name where the XDB wallet is stored is prefixed either by ORACLE_BASE when it is defined, or ORACLE_HOME. It is then followed by /admin/db_name/xdb_wallet where db_name is the unique database name.

Syntax

DBMS_XDB_ADMIN.INSTALLDEFAULTWALLET;

Usage Notes

Only SYS can install or replace the default wallet.

191.2.3 MOVEXDB_TABLESPACE Procedure

This procedure moves the XDB (user) to the specified tablespace.

Syntax

DBMS_XDB_ADMIN.MOVEXDB_TABLESPACE(
    new_tablespace   IN  VARCHAR2);

Parameters

Table 191-2  MOVEXDB_TABLESPACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_tablespace</td>
<td>Name of the tablespace to where the XDB is moved</td>
</tr>
</tbody>
</table>

Usage Notes

- This operation waits for all concurrent XDB sessions to exit.
- If MOVEXDB_TABLESPACE fails, the user should restart the database before issuing any further command. Failure to do so will result into unexpected behavior from the database.
- The XDB repository by default resides in the SYSAUX tablespace. Using this procedure it can be moved to another tablespace. As a best practice we recommend to create a dedicated tablespace for the XDB repository only and not share it with other objects (such as tables). The tablespace containing the XDB repository should never be set to READ ONLY because this might affect various XML operations being executed.
191.2.4 REBUILDHIERARCHICALINDEX Procedure

This procedure rebuilds the hierarchical index after import or export operations. This is necessary because data cannot be exported from index tables.

Syntax

```
DBMS_XDB_ADMIN.REBUILDHIERARCHICALINDEX;
```
The DBMS_XDB_CONFIG package provides an interface for configuring Oracle XML DB and its repository.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Summary of DBMS_XDB_CONFIG Subprograms

See Also:
Oracle XML DB Developer's Guide

192.1 DBMS_XDB_CONFIG Overview

PL/SQL package DBMS_XDB_CONFIG is the Oracle XML DB resource application program interface (API) for PL/SQL for DBAs to configure their system. This API provides functions and procedures to access and manage Oracle XML DB Repository resources using PL/SQL. It includes methods for managing resource security and Oracle XML DB configuration.

Oracle XML DB Repository is modeled on XML, and provides a database file system for any data. The repository maps path names (or URLs) onto database objects of XMLType and provides management facilities for these objects.

PL/SQL package DBMS_XDB_CONFIG is an API that you can use to manage all of the following:

- Oracle XML DB resources
- Oracle XML DB security based on access control lists (ACLs). An ACL is a list of access control entries (ACEs) that determines which principals (users and roles) have access to which resources.
- Oracle XML DB configuration

192.2 DBMS_XDB_CONFIG Security Model

Owned by XDB, the DBMS_XDB_CONFIG package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.
192.3 DBMS_XDB_CONFIG Constants

The **DBMS_XDB_CONFIG** package defines several enumerated constants that should be used for specifying parameter values.

These constants are shown in the following table.

**Table 192-1  DBMS_XDB_CONFIG Constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDB_ENDPOINT_HTTP</td>
<td>NUMBER</td>
<td>1</td>
<td>Defining listener for first HTTP port</td>
</tr>
<tr>
<td>XDB_ENDPOINT_HTTP2</td>
<td>NUMBER</td>
<td>2</td>
<td>Defining listener for second HTTP port</td>
</tr>
<tr>
<td>XDB_ENDPOINT_RHTTP</td>
<td>NUMBER</td>
<td>3</td>
<td>Defining listener for a remote HTTP port</td>
</tr>
<tr>
<td>XDB_ENDPOINT_RHTTPS</td>
<td>NUMBER</td>
<td>4</td>
<td>Defining listener for a remote HTTPS port</td>
</tr>
<tr>
<td>XDB_PROTOCOL_TCP</td>
<td>NUMBER</td>
<td>1</td>
<td>Defining listener for HTTP protocol</td>
</tr>
<tr>
<td>XDB_PROTOCOL_TCPS</td>
<td>NUMBER</td>
<td>2</td>
<td>Defining listener for HTTPS protocol</td>
</tr>
</tbody>
</table>

192.4 Summary of DBMS_XDB_CONFIG Subprograms

The **DBMS_XDB_CONFIG** package uses subprograms for configuring Oracle XML DB and its repository.

These subprograms are listed in the following table.

**Table 192-2  DBMS_XDB_CONFIG Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDHTTPEXPIREMAPPING</td>
<td>Adds to XDB$CONFIG a mapping of the URL pattern to an expiration date. This will control the Expire headers for URLs matching the pattern.</td>
</tr>
<tr>
<td>ADDMIMEMAPPING Procedure</td>
<td>Adds a mime mapping to XDB configuration</td>
</tr>
<tr>
<td>ADDSCHEMALOCMAPPING</td>
<td>Adds a schema location mapping to the XDB configuration</td>
</tr>
<tr>
<td>ADDSERVLET Procedure</td>
<td>Adds a servlet to XDB configuration</td>
</tr>
<tr>
<td>ADDSERVLETMAPPING</td>
<td>Adds a servlet mapping to XDB configuration</td>
</tr>
<tr>
<td>ADDSERVLETSECROLE</td>
<td>Adds a security role REF to a specified servlet in XDB configuration</td>
</tr>
<tr>
<td>ADDXMLEXTENSION Procedure</td>
<td>Adds adds the following XML extension to the XDB configuration under &lt;xml-extensions&gt;:&lt;extension&gt;extension&lt;/extension&gt;</td>
</tr>
<tr>
<td>CFG_GET Function</td>
<td>Retrieves the session’s configuration information as an XMLType instance</td>
</tr>
</tbody>
</table>
### Table 192-2  (Cont.) DBMS_XDB_CONFIG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFG_REFRESH Procedure</td>
<td>Refreshes the session's configuration information to the latest configuration</td>
</tr>
<tr>
<td>CFG_UPDATE Procedure</td>
<td>Updates the configuration information and commits the change.</td>
</tr>
<tr>
<td>DELETETHTTPEXPIREMAPPING Procedure</td>
<td>Deletes from XDB$CONFIG all mappings of the URL pattern to an expiration date</td>
</tr>
<tr>
<td>DELETEMIMEMAPPING Procedure</td>
<td>Deletes the mime mapping for a specified extension from the XDB configuration</td>
</tr>
<tr>
<td>DELETESCHEMALLOCMAPPING Procedure</td>
<td>Deletes the schema location mapping for a specified schema URL from the XDB configuration</td>
</tr>
<tr>
<td>DELETESERVLET Procedure</td>
<td>Deletes a servlet from the XDB configuration</td>
</tr>
<tr>
<td>DELETESERVLETMAPPING Procedure</td>
<td>Deletes the servlet mapping for a specified servlet name from the XDB configuration</td>
</tr>
<tr>
<td>DELETESERVLETSECROLE Procedure</td>
<td>Deletes the specified role from a servlet in the XDB configuration</td>
</tr>
<tr>
<td>DELETEXMLEXTENSION Procedure</td>
<td>Deletes the specified XML extension from the XDB configuration</td>
</tr>
<tr>
<td>ENABLEDIGESTAUTHENTICATION Procedure</td>
<td>Enables digest authentication</td>
</tr>
<tr>
<td>GETFTPPORT Function</td>
<td>Gets the value of the current FTP port</td>
</tr>
<tr>
<td>GETHTTPPORT Function</td>
<td>Gets the value of the current HTTP port</td>
</tr>
<tr>
<td>GETHTTPCONFIGREALM Function</td>
<td>Gets the realm name</td>
</tr>
<tr>
<td>GETHTTPS.PORT Function</td>
<td>Gets the value of the current HTTPS port</td>
</tr>
<tr>
<td>GETLISTENERENDPOINT Procedure</td>
<td>Gets the parameters of a listener end point corresponding to the XML DB HTTP server</td>
</tr>
<tr>
<td>GETREMOTEHTTPPORT Function</td>
<td>Gets the value of a remote HTTP port</td>
</tr>
<tr>
<td>GETREMOTEHTTPS_PORT Function</td>
<td>Gets the value of a remote HTTPS port</td>
</tr>
<tr>
<td>ISGLOBALPORTENABLED Function</td>
<td>Returns the value of the DBMS_XDB_CONFIG.SETGLOBALPORTENABLED procedure setting</td>
</tr>
<tr>
<td>SETGLOBALPORTENABLED Procedure</td>
<td>Enables the use of a global port for Oracle Enterprise Manager Database Express (EM Express) clients</td>
</tr>
<tr>
<td>SETFTPPORT Procedure</td>
<td>Sets the FTP port to a new value</td>
</tr>
<tr>
<td>SETHTTPPORT Procedure</td>
<td>Sets the HTTP port to a new value</td>
</tr>
<tr>
<td>SETHTTPCONFIGREALM Procedure</td>
<td>Sets the realm to a new value</td>
</tr>
<tr>
<td>SETHTTPS.PORT Procedure</td>
<td>Sets the HTTPS port to a new value</td>
</tr>
<tr>
<td>SETLISTENERENDPOINT Procedure</td>
<td>Sets the parameters of a listener end point corresponding to the XML DB HTTP server</td>
</tr>
</tbody>
</table>
Table 192-2  (Cont.) DBMS_XDB_CONFIG Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETLISTENERLOCALACCESS</td>
<td>Procedure</td>
</tr>
<tr>
<td></td>
<td>Restricts all listener end points of the XML DB HTTP server to listen either only on the localhost interface (when <code>l_access</code> is set to <code>TRUE</code>) or to listen on both localhost and non-localhost interfaces (when <code>l_access</code> is set to <code>FALSE</code>).</td>
</tr>
<tr>
<td>SETREMOTEHTTPPORT</td>
<td>Procedure</td>
</tr>
<tr>
<td></td>
<td>Sets the remote HTTP port to a new value</td>
</tr>
<tr>
<td>SETREMOTEHTTPS.PORT</td>
<td>Procedure</td>
</tr>
<tr>
<td></td>
<td>Sets the remote HTTPS port to a new value</td>
</tr>
<tr>
<td>USEDPORT</td>
<td>Procedure</td>
</tr>
<tr>
<td></td>
<td>Obtains the port numbers used by other pluggable databases in the same consolidation database</td>
</tr>
</tbody>
</table>

192.4.1 ADDHTTPEXPIREMAPPING Procedure

This procedure adds to `XDB$CONFIG` a mapping of the URL pattern to an expiration date. This will control the Expire headers for URLs matching the pattern.

Syntax

```
DBMS_XDB_REPOS.ADDHTTPEXPIREMAPPING (
    pattern    IN    VARCHAR2,
    expire     IN    VARCHAR2);
```

Parameters

Table 192-3  ADDHTTPEXPIREMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>URL pattern (only * accepted as wildcards)</td>
</tr>
</tbody>
</table>
| expire    | Expiration directive, follows the ExpireDefault in Apache's `mod_expires`:
```
  base [plus] (num type)*
  -- base: now | modification
  -- type: year|years|month|months|week|weeks|day|days|minute|minutes|second|seconds
```

Examples

```
DBMS_XDB_REPOS.ADDHTTPEXPIREMAPPING ('/public/test1/**', 'now plus 4 weeks');
DBMS_XDB_REPOS.ADDHTTPEXPIREMAPPING ('/public/test2/**', 'modification plus 1 day 30 seconds');
```

192.4.2 ADDMIMEMAPPING Procedure

This procedure adds the following mime mapping to XDB configuration:

```
<mime-mapping>
  <extension>extension</extension>
  <mime-type>mimetype</mime-type>
</mime-mapping>
```
Syntax

DBMS_XDB_CONFIG.ADDMIMEMAPPING(
   extension    IN   VARCHAR2,
   mimetype     IN   VARCHAR2);

Parameters

Table 192-4  ADDMIMEMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension</td>
<td>Extension for which a mime type is being added</td>
</tr>
<tr>
<td>mimetype</td>
<td>Mime type</td>
</tr>
</tbody>
</table>

192.4.3 ADDSCHEMALOCMAPPING Procedure

This procedure adds the following schema location mapping to the XDB configuration:

<schemamapping>
   <namespace>namespace</namespace>
   <element>element</element>
   <schemaURL>schemaURL</schemaURL>
</schemamapping>

Syntax

DBMS_XDB_CONFIG.ADDSCHEMALOCMAPPING(
   namespace    IN   VARCHAR2,
   element      IN   VARCHAR2,
   schemaURL    IN   VARCHAR2);

Parameters

Table 192-5  ADDSCHEMALOCMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>Namespace</td>
</tr>
<tr>
<td>element</td>
<td>Element</td>
</tr>
<tr>
<td>schemaURL</td>
<td>Schema URL</td>
</tr>
</tbody>
</table>

192.4.4 ADDSERVLET Procedure

This procedure adds a servlet to XDB configuration.

IT adds the following servlet:

<servlet>
   <servlet-name>name</servlet-name>     <servlet-language>language</servlet-language>
   <display-name>dispname</display-name>
   <description>descript</description>
   <servlet-class>class</servlet-class>
DBMS_XDB_CONFIG.ADDSERVLET(
    name        IN   VARCHAR2,
    language    IN   VARCHAR2,
    dispname    IN   VARCHAR2,
    icon        IN   VARCHAR2 := NULL,
    descript    IN   VARCHAR2 := NULL,
    class       IN   VARCHAR2 := NULL,
    jspfile     IN   VARCHAR2 := NULL,
    plsql       IN   VARCHAR2 := NULL,
    schema      IN   VARCHAR2 := NULL);

Parameters

Table 192-6  ADDSERVLET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Servlet name</td>
</tr>
<tr>
<td>language</td>
<td>Must be one of &quot;C&quot;, &quot;Java&quot;, &quot;PL/SQL&quot;</td>
</tr>
<tr>
<td>dispname</td>
<td>Display name</td>
</tr>
<tr>
<td>icon</td>
<td>Icon</td>
</tr>
<tr>
<td>descript</td>
<td>Description</td>
</tr>
<tr>
<td>class</td>
<td>The class/jspfile/plsql function corresponding to this servlet. The first non-NULL argument amongst these three is chosen, and the others are treated as NULL.</td>
</tr>
<tr>
<td>jspfile</td>
<td>This parameter is not supported. Always provide NULL value for this parameter. System throws an error if the value is not NULL.</td>
</tr>
<tr>
<td>plsql</td>
<td>The class/jspfile/plsql function corresponding to this servlet. The first non-NULL argument amongst these three is chosen, and the others are treated as NULL.</td>
</tr>
<tr>
<td>schema</td>
<td>Schema is used to specify servlet-schema xml element name in xdbconfig.xml. It indicates the Oracle schema in which the Java class is loaded. If you do not specify any value, then the schema is searched using the default resolver specification.</td>
</tr>
</tbody>
</table>

192.4.5 ADDSERVLETMAPPING Procedure

This procedure adds a servlet mapping to XDB configuration.

It adds the following servlet mapping:

```xml
<servlet-mapping>
    <servlet-pattern>pattern</servlet-pattern>
    <servlet-name>name</servlet-name>
</servlet-mapping>
```
Syntax

DBMS_XDB_CONFIG.ADDSERVLETMAPPING(
    pattern   IN   VARCHAR2,     name      IN   VARCHAR2);

Parameters

Table 192-7  ADDSERVLETMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>Servlet pattern</td>
</tr>
<tr>
<td>name</td>
<td>Servlet name</td>
</tr>
</tbody>
</table>

192.4.6 ADDSERVLETSECROLE Procedure

This procedure adds the security role REF to a specified servlet in XDB configuration.

It adds the following security role as shown in the following:

```xml
<security-role-ref>
    <role-name>rolename</role-name>
    <role-link>rolelink</role-link>
    <description>description</description>
</security-role-ref>
```

Syntax

DBMS_XDB_CONFIG.ADDSERVLETSECROLE(
    servname    IN   VARCHAR2,     rolename    IN   VARCHAR2,     rolelink    IN VARCHAR2,     descript    IN   VARCHAR2 := NULL);

Parameters

Table 192-8  ADDSERVLETSECROLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>servname</td>
<td>Servlet name</td>
</tr>
<tr>
<td>rolename</td>
<td>Role name</td>
</tr>
<tr>
<td>rolelink</td>
<td>Role link</td>
</tr>
<tr>
<td>descript</td>
<td>Description</td>
</tr>
</tbody>
</table>

192.4.7 ADDXMLEXTENSION Procedure

This procedure adds an XML extension to the XDB configuration under `<xml-extensions>`.

It adds the following XML extension to the XDB configuration, under `<xml-extensions>`:

```xml
<extension>extension</extension>
```
Syntax

DBMS_XDB_CONFIG.ADDXMLEXTENSION(
    extension IN VARCHAR2);

Parameters

Table 192-9  ADDXMLEXTENSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension</td>
<td>XML extension to be added</td>
</tr>
</tbody>
</table>

192.4.8 CFG_GET Function

This function retrieves the session's configuration information as an XMLType instance.

Syntax

DBMS_XDB_CONFIG.CFG_GET
RETURN SYS.XMLType;

192.4.9 CFG_REFRESH Procedure

This procedure refreshes the session's configuration information to the latest configuration.

Syntax

DBMS_XDB_CONFIG.CFG_REFRESH;

192.4.10 CFG_UPDATE Procedure

This procedure updates the configuration information and commits the change.

Syntax

DBMS_XDB_CONFIG.CFG_UPDATE(
    xdbconfig IN SYS.XMLTYPE);

Parameters

Table 192-10  CFG_UPDATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdbconfig</td>
<td>The new configuration data</td>
</tr>
</tbody>
</table>
192.4.11 DELETEHTTPEXPIREMAPPING Procedure

This procedure deletes from XDB$CONFIG all mappings of the URL pattern to an expiration date.

Syntax

```sql
DBMS_XDB_REPOS.DELETEHTTPEXPIREMAPPING(
    pattern IN VARCHAR2);
```

Parameters

Table 192-11  DELETEHTTPEXPIREMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>URL pattern (only * accepted as wildcards)</td>
</tr>
</tbody>
</table>

192.4.12 DELETEMIMEMAPPING Procedure

This procedure deletes the mime mapping for a specified extension from the XDB configuration.

Syntax

```sql
DBMS_XDB_CONFIG.DELETEMIMEMAPPING(
    extension IN VARCHAR2);
```

Parameters

Table 192-12  DELETEMIMEMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension</td>
<td>Extension for which a mime type is to be deleted</td>
</tr>
</tbody>
</table>

192.4.13 DELETESCHEMALOCMAPPING Procedure

This procedure deletes the schema location mapping for a specified schema URL from the XDB configuration.

Syntax

```sql
DBMS_XDB_CONFIG.DELETESCHEMALOCMAPPING(
    schemaURL IN VARCHAR2);
```

Parameters

Table 192-13  DELETESCHEMALOCMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaURL</td>
<td>Schema URL</td>
</tr>
</tbody>
</table>
192.4.14 DELETESERVLET Procedure

This procedure deletes a servlet from the XDB configuration.

Syntax

```sql
DBMS_XDB_CONFIG.DELETESERVLET(
    name IN VARCHAR2);
```

Parameters

Table 192-14  DELETESERVLET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Servlet name</td>
</tr>
</tbody>
</table>

192.4.15 DELETESERVLETMAPPING Procedure

This procedure deletes the servlet mapping for a specified servlet name from the XDB configuration.

Syntax

```sql
DBMS_XDB_CONFIG.DELETESERVLETMAPPING(
    name IN VARCHAR2);
```

Parameters

Table 192-15  DELETESERVLETMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Servlet name</td>
</tr>
</tbody>
</table>

192.4.16 DELETESERVLETSECROLE Procedure

This procedure deletes the specified role from a servlet in the XDB configuration.

Syntax

```sql
DBMS_XDB_CONFIG.DELETESERVLETSECROLE(
    servname IN VARCHAR2, rolename IN VARCHAR2);
```

Parameters

Table 192-16  DELETESERVLETSECROLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>servname</td>
<td>Servlet name</td>
</tr>
<tr>
<td>rolename</td>
<td>Name of the role to be deleted</td>
</tr>
</tbody>
</table>
192.4.17 DELETEXMLEXTENSION Procedure

This procedure deletes the specified XML extension from the XDB configuration.

**Syntax**

```sql
DBMS_XDB_CONFIG.DELETEXMLEXTENSION(
    extension IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extension</td>
<td>XML extension to be deleted</td>
</tr>
</tbody>
</table>

192.4.18 ENABLEDIGESTAUTHENTICATION Procedure

This procedure enables digest authentication. It will list digest as the first authentication mechanism to be used by the XML DB HTTP server.

**Syntax**

```sql
DBMS_XDB_CONFIG.ENABLEDIGESTAUTHENTICATION;
```

192.4.19 GETFTPPORT Function

This procedure gets the value of the current FTP port.

**Syntax**

```sql
DBMS_XDB_CONFIG.GETFTPPORT
RETURN NUMBER;
```

192.4.20 GETHTTPCONFIGREALM Function

This function gets the realm name. Definition of a realm is referenced in IETF's RFC2617.

**Syntax**

```sql
DBMS_XDB_CONFIG.GETHTTPCONFIGREALM
RETURN VARCHAR2;
```

192.4.21 GETHTTPPORT Function

This function gets the value of the current HTTP port.

**Syntax**

```sql
DBMS_XDB_CONFIG.GETHTTPPORT
RETURN NUMBER;
```
192.4.22 GETHTTPSPORT Function

This procedure gets the value of the current HTTPS port.

Syntax

```sql
DBMS_XDB_CONFIG.GETHTTPSPORT
RETURN NUMBER;
```

Usage Notes

Returns NULL if no port has been configured

192.4.23 GETLISTENERENDPOINT Procedure

This procedure retrieves the parameters of a listener end point corresponding to the XML DB HTTP server. The parameters of both HTTP and HTTP2 end points can be retrieved by invoking this procedure.

Syntax

```sql
DBMS_XDB_CONFIG.GETLISTENERENDPOINT ( endpoint IN NUMBER,
                                        host      OUT VARCHAR2,    port      OUT NUMBER,
                                        protocol  OUT NUMBER);
```

Parameters

**Table 192-18  GETLISTENERENDPOINT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpoint</td>
<td>End point to be retrieved. Its value can be XDB_ENDPOINT_HTTP or XDB_ENDPOINT_HTTP2.</td>
</tr>
<tr>
<td>host</td>
<td>Interface on which the listener end point listens</td>
</tr>
<tr>
<td>port</td>
<td>Port on which the listener end point listens</td>
</tr>
<tr>
<td>protocol</td>
<td>Transport protocol accepted by the listener end point</td>
</tr>
</tbody>
</table>

192.4.24 GETREMOTEHTTPPORT Function

This function gets the value of a remote HTTP port.

Syntax

```sql
DBMS_XDB_CONFIG.GETREMOTEHTTPPORT
RETURN NUMBER;
```
192.4.25 GETREMOTEHTTPSPORT Function

This function gets the value of a remote HTTPS port.

Syntax

```sql
DBMS_XDB_CONFIG.GETREMOTEHTTPSPORT
RETURN NUMBER;
```

192.4.26 ISGLOBALPORTENABLED Function

This procedure returns the value of the `GlobalPortEnabled` setting.

Syntax

```sql
DBMS_XDB_CONFIG.ISGLOBALPORTENABLED RETURN BOOLEAN;
```

Usage Notes

- This procedure returns `TRUE` if `GlobalPortEnabled` has been set to `TRUE`; otherwise it returns `FALSE`.
- In a multitenant environment, you can execute this function in both the CDB root and PDBs.

192.4.27 SETFTPPORT Procedure

This procedure sets the FTP port to a new value.

Syntax

```sql
DBMS_XDB_CONFIG.SETFTPPORT(
    new_port  IN  NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_port</td>
<td>Value to which the FTP port is set</td>
</tr>
</tbody>
</table>

192.4.28 SETGLOBALPORTENABLED Procedure

This procedure enables Oracle Enterprise Manager Database Express (EM Express) clients to use a single port (called a global port), for the session rather than using a port dedicated to the PDB.

Syntax

```sql
DBMS_XDB_CONFIG.SETGLOBALPORTENABLED (
    isenabled  IN  BOOLEAN);
```
Parameters

Table 192-20  SETGLOBALPORTENABLED Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isenabled</td>
<td>TRUE enables the EM Express client to use a global port for the session rather than using the PDB that allocated the port. FALSE enables the port ID to determine the PDB to use. The default is TRUE for PDBs but FALSE for the CDB root.</td>
</tr>
</tbody>
</table>

Usage Notes

• Use the DBMS_XDB_CONFIG.SETGLOBALPORTENABLED procedure if you use EM Express to manage database ports in a multitenant environment.

• To use a global port, you must enable it in both the CDB and the PDB. PDBs have it enabled by default. For CDBs, you must run the DBMS_XDB_CONFIG.SETGLOBALPORTENABLED procedure in the CDB root.

• After you set DBMS_XDB_CONFIG.SETGLOBALPORTENABLED to TRUE, then you must specify a PDB to use for the global port. You can do so by specifying the PDB name in the Container Name field of the EM Express login page. For example, if you enter hr_pdb, then the EM Express URL becomes http://server_name:5500/hr_pdb/em. This example uses the port 5500 in the CDB root, which in turn routes messages to the EM Express in the PDB hr_pdb. The port can be an HTTP or HTTPS port.

• The PDB that you specify in the EM Express login page must be open in read/write mode before you can run DBMS_XDB_CONFIG.SETGLOBALPORTENABLED. You can check the read/write mode status of a PDB by connecting to the PDB and then querying the OPEN_MODE column of the V$PDBS dynamic view.

• The DBMS_XDB_CONFIG.ISGLOBALPORTENABLED function returns the value of the DBMS_XDB_CONFIG.SETGLOBALPORTENABLED setting.

• You can use the XDB_PROTOCOLS lockdown profile to control the security for the global port.

See Also:

• Oracle Database 2 Day DBA for information about using DBMS_XDB_CONFIG.SETGLOBALPORTENABLED to start EM Express for a PDB

• Oracle Database Security Guide

for information about lockdown profiles
192.4.29 SETHTTPPORT Procedure

This procedure sets the HTTP port to a new value.

Syntax

```
DBMS_XDB_CONFIG.SETHTTPPORT(
    new_port IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_port</td>
<td>Value to which the HTTP port is set</td>
</tr>
</tbody>
</table>

192.4.30 SETHTTPCONFIGREALM Procedure

This procedure modifies the realm value.

Syntax

```
DBMS_XDB_CONFIG.SETHTTPCONFIGREALM(
    realm IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realm</td>
<td>Realm as defined in IETF’s RFC2617</td>
</tr>
</tbody>
</table>

192.4.31 SETHTTPSPORT Procedure

This procedure sets the HTTPS port to a new value.

Syntax

```
DBMS_XDB_CONFIG.SETHTTPSPORT(
    new_port IN NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_port</td>
<td>Value to which the HTTPS port is set</td>
</tr>
</tbody>
</table>
192.4.32 SETLISTENERENDPOINT Procedure

This procedure sets the parameters of a listener end point corresponding to the XML DB HTTP server.

Both HTTP and HTTP2 end points can be set by invoking this procedure.

Syntax

```sql
DBMS_XDB_CONFIG.SETLISTENERENDPOINT (
    endpoint  IN  NUMBER,
    host      IN  VARCHAR2,    port      IN  NUMBER,
    protocol  IN  NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpoint</td>
<td>End point to be set. Its value can be XDB_ENDPOINT_HTTP or XDB_ENDPOINT_HTTP2.</td>
</tr>
<tr>
<td>host</td>
<td>Interface on which the listener end point is to listen. Its value can be 'LOCALHOST,' NULL, or a hostname. If its value is 'LOCALHOST' the listener end point is permitted to only listen on the localhost interface. If its value is NULL or hostname, the listener end point is permitted to listen on both localhost and non-localhost interfaces.</td>
</tr>
<tr>
<td>port</td>
<td>Port on which the listener end point is to listen</td>
</tr>
<tr>
<td>protocol</td>
<td>Transport protocol that the listener end point is to accept. Its value can be XDB_PROTOCOL_TCP or XDB_PROTOCOL_TCPS</td>
</tr>
</tbody>
</table>

192.4.33 SETLISTENERLOCALACCESS Procedure

This procedure restricts all listener end points of the XML DB HTTP server to listen either only on the localhost interface (when `l_access` is set to TRUE) or to listen on both localhost and non-localhost interfaces (when `l_access` is set to FALSE).

Syntax

```sql
DBMS_XDB_CONFIG.SETLISTENERLOCALACCESS (
    l_access   BOOLEAN);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_access</td>
<td>TRUE or FALSE.</td>
</tr>
</tbody>
</table>
192.4.34 SETREMOTEHTTPPORT Procedure

This procedure sets a remote HTTP port to a new value.

**Syntax**

```sql
DBMS_XDB_CONFIG.SETREMOTEHTTPPORT(
    new_port  IN  NUMBER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_port</td>
<td>Value to which the remote HTTP port is set</td>
</tr>
</tbody>
</table>

192.4.35 SETREMOTEHTTPSPORT Procedure

This procedure sets a remote HTTPS port to a new value.

**Syntax**

```sql
DBMS_XDB_CONFIG.SETREMOTEHTTPSPORT (  
    new_port  IN  NUMBER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_port</td>
<td>Value to which the remote HTTPS port is set</td>
</tr>
</tbody>
</table>

192.4.36 USEDPORT Function

This function obtains the port numbers used by other pluggable databases in the same consolidation database.

**Syntax**

```sql
DBMS_XDB_CONFIG.USEDPORT RETURN XMLTYPE;
```
The DBMS_XDB_CONSTANTS package provides an interface to commonly used constants. Users should use constants instead of dynamic strings to avoid typographical errors.

This chapter contains the following topics:

- Security Model
- Summary of DBMS_XDB_CONSTANTS Subprograms

193.1 DBMS_XDB_CONSTANTS Security Model

Owned by XDB, the DBMS_XDB_CONSTANTS package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

193.2 Summary of DBMS_XDB_CONSTANTS Subprograms

This table lists the DBMS_XDB_CONSTANTS subprograms and briefly describes them.

**Table 193-1  DBMS_XDB_CONSTANTS Package Subprograms**

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCODING_DEFAULT Function</td>
<td>Returns 'AL32UTF8'</td>
</tr>
<tr>
<td>ENCODING_ISO88591 Function</td>
<td>Returns 'WE8ISO8859P1'</td>
</tr>
<tr>
<td>ENCODING_UTF8 Function</td>
<td>Returns 'AL32UTF8'</td>
</tr>
<tr>
<td>ENCODING_WIN1252 Function</td>
<td>Returns 'WE8MSWIN1252'</td>
</tr>
<tr>
<td>NAMESPACE_ACL Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/ac1.xsd">http://xmlns.oracle.com/xdb/ac1.xsd</a>'</td>
</tr>
<tr>
<td>NAMESPACE_METADATA Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdbuserMetaData">http://xmlns.oracle.com/xdbuserMetaData</a>'</td>
</tr>
<tr>
<td>NAMESPACE_ORACLE Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb">http://xmlns.oracle.com/xdb</a>'</td>
</tr>
<tr>
<td>NAMESPACE_ORACLE_XDB Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb">http://xmlns.oracle.com/xdb</a>'</td>
</tr>
<tr>
<td>NAMESPACE_RESOURCE Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/XDBResource.xsd">http://xmlns.oracle.com/xdb/XDBResource.xsd</a>'</td>
</tr>
<tr>
<td>NAMESPACE_RESOURCE_EVENT Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/resourceEvent">http://xmlns.oracle.com/resourceEvent</a>'</td>
</tr>
<tr>
<td>NAMESPACE_RESOURCE_CONFIG Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/XDBResConfig.xsd">http://xmlns.oracle.com/XDBResConfig.xsd</a>'</td>
</tr>
<tr>
<td>NAMESPACE_XDBSCHEMA Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/XDBSchema.xsd">http://xmlns.oracle.com/xdb/XDBSchema.xsd</a>'</td>
</tr>
<tr>
<td>NAMESPACE_XMLEDIFF Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/xdiff.xsd">http://xmlns.oracle.com/xdb/xdiff.xsd</a>'</td>
</tr>
</tbody>
</table>
### Table 193-1  (Cont.) DBMS_XDB_CONSTANTS Package Subprograms

<table>
<thead>
<tr>
<th>Subprograms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMESPACE_XMLINSTANCE Function</td>
<td>Returns '<a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a>'</td>
</tr>
<tr>
<td>NAMESPACE_XMLSCHEMA Function</td>
<td>Returns '<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>'</td>
</tr>
<tr>
<td>NSPREFIX_ACL_ACL Function</td>
<td>Returns 'xmlns:acl= '<a href="http://xmlns.oracle.com/acs.xsd">http://xmlns.oracle.com/acs.xsd</a>'</td>
</tr>
<tr>
<td>NSPREFIX_RESOURCE_R Function</td>
<td>Returns 'xmlns:r=&quot;<a href="http://xmlns.oracle.com/XDBResource.xsd">http://xmlns.oracle.com/XDBResource.xsd</a>'</td>
</tr>
<tr>
<td>NSPREFIX_XDB_XDB Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb">http://xmlns.oracle.com/xdb</a>'</td>
</tr>
<tr>
<td>NSPREFIX_XMLINSTANCE_XSI Function</td>
<td>Returns 'xmlns:xsi=&quot;<a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a>&quot;'</td>
</tr>
<tr>
<td>NSPREFIX_XMLDIFF_XD Function</td>
<td>Returns 'xmlns:xd= &quot;<a href="http://xmlns.oracle.com/xdb/xdiff.xsd">http://xmlns.oracle.com/xdb/xdiff.xsd</a>&quot;'</td>
</tr>
<tr>
<td>NSPREFIX_XMLSCHEMA_XSD Function</td>
<td>Returns 'xmlns:xsd=&quot;<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>&quot;'</td>
</tr>
<tr>
<td>SCHEMAURL_ACL Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/acl.xsd">http://xmlns.oracle.com/xdb/acl.xsd</a>'</td>
</tr>
<tr>
<td>SCHEM AeLEM_RES_ACL Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/acl.xsd#acl">http://xmlns.oracle.com/xdb/acl.xsd#acl</a>'</td>
</tr>
<tr>
<td>SCHEM AeLEM_RESCONTENT_BINARY Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/XDBSchema.xsd#binary">http://xmlns.oracle.com/xdb/XDBSchema.xsd#binary</a>'</td>
</tr>
<tr>
<td>SCHEM AeLEM_RESCONTENT_TEXT Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/XDBSchema.xsd#text">http://xmlns.oracle.com/xdb/XDBSchema.xsd#text</a>'</td>
</tr>
<tr>
<td>SCHEM AeURL_RESOURCE Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/XDBResource.xsd">http://xmlns.oracle.com/xdb/XDBResource.xsd</a>'</td>
</tr>
<tr>
<td>SCHEM AeURL_XDBSCHEMA Function</td>
<td>Returns '<a href="http://xmlns.oracle.com/xdb/XDBSchema.xsd">http://xmlns.oracle.com/xdb/XDBSchema.xsd</a>'</td>
</tr>
<tr>
<td>XSD_ATTRIBUTE Function</td>
<td>Returns 'attribute'</td>
</tr>
<tr>
<td>XSD_COMPLEX_TYPE Function</td>
<td>Returns 'complexType'</td>
</tr>
<tr>
<td>XSD_ELEMENT Function</td>
<td>Returns 'element'</td>
</tr>
<tr>
<td>XSD_GROUP Function</td>
<td>Returns 'group'</td>
</tr>
</tbody>
</table>

#### 193.2.1 ENCODING_DEFAULT Function

This function returns 'AL32UTF8'.

**Syntax**

```
DBMS_XDB_CONSTANTS.ENCODING_DEFAULT
RETURN VARCHAR2 DETERMINISTIC;
```
193.2.2 ENCODING_ISO88591 Function

This function returns 'WE8ISO8859P1'.

Syntax

DBMS_XDB_CONSTANTS.ENCODING_ISO88591
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'WE8ISO8859P1'

193.2.3 ENCODING_UTF8 Function

This function returns 'AL32UTF8'.

Syntax

DBMS_XDB_CONSTANTS.ENCODING_UTF8
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'AL32UTF8'

193.2.4 ENCODING_WIN1252 Function

This function returns 'WE8MSWIN1252'.

Syntax

DBMS_XDB_CONSTANTS.ENCODING_WIN1252
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'WE8MSWIN1252'

193.2.5 NAMESPACE_ACL Function

This function returns 'http://xmlns.oracle.com/xdb/acl.xsd'.

Syntax

DBMS_XDB_CONSTANTS.NAMESPACE_ACL
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'http://xmlns.oracle.com/xdb/acl.xsd'
193.2.6 NAMESPACE_METADATA Function

This function returns 'http://xmlns.oracle.com/xdbuserMetaData'.

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_METADATA
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdbuserMetaData'

193.2.7 NAMESPACE_ORACLE Function

This function returns 'http://xmlns.oracle.com'.

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_ORACLE
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com'

193.2.8 NAMESPACE_ORACLE_XDB Function

This function returns 'http://xmlns.oracle.com/xdb'.

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_ORACLE_XDB
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdb'

193.2.9 NAMESPACE_RESOURCE Function

This function returns '/XDBResource.xsd'

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_RESOURCE
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdb/XDBResource.xsd'
193.2.10 NAMESPACE_RESOURCE_EVENT Function

This function returns 'http://xmlns.oracle.com/resourceEvent'.

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_RESOURCE_EVENT
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/resourceEvent'

193.2.11 NAMESPACE_RESOURCE_CONFIG Function

This function returns 'http://xmlns.oracle.com/XDBResConfig.xsd'.

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_RESOURCE_CONFIG
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/XDBResConfig.xsd'

193.2.12 NAMESPACE_XDBSCHEMA Function

This function returns 'http://xmlns.oracle.com/xdb/XDBSchema.xsd'.

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_XDBSCHEMA
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdb/XDBSchema.xsd'

193.2.13 NAMESPACE_XMLDIFF Function

This function returns 'http://xmlns.oracle.com/xdb/xdiff.xsd'.

Syntax

```
DBMS_XDB_CONSTANTS.NAMESPACE_XMLDIFF
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdb/xdiff.xsd'
193.2.14 NAMESPACE_XMLINSTANCE Function

This function returns 'http://www.w3.org/2001/XMLSchema-instance'.

Syntax

DBMS_XDB_CONSTANTS.NAMESPACE_XMLINSTANCE
  RETURN VARCHAR2 DETERMINISTIC;

Return Value

'http://www.w3.org/2001/XMLSchema-instance'

193.2.15 NAMESPACE_XMLSCHEMA Function

This function returns 'http://www.w3.org/2001/XMLSchema'.

Syntax

DBMS_XDB_CONSTANTS.NAMESPACE_XMLSCHEMA
  RETURN VARCHAR2 DETERMINISTIC;

Return Value

'http://www.w3.org/2001/XMLSchema'

193.2.16 NSPREFIX_ACL_ACL Function

This function returns 'xmlns:acl= 'http://xmlns.oracle.com/acs.xsd'.

Syntax

DBMS_XDB_CONSTANTS.NSPREFIX_ACL_ACL
  RETURN VARCHAR2 DETERMINISTIC;

Return Value

'xmlns:acl= 'http://xmlns.oracle.com/acs.xsd'

193.2.17 NSPREFIX_RESCONFIG_RC Function

This function returns ''xmlns:rc="http://xmlns.oracle.com/xdb/XDBResConfig.xsd"'.

Syntax

DBMS_XDB_CONSTANTS.NSPREFIX_RESCONFIG_RC
  RETURN VARCHAR2 DETERMINISTIC;

Return Value

Returns ''xmlns:rc="http://xmlns.oracle.com/xdb/XDBResConfig.xsd"'
193.2.18 NSPREFIXRESOURCE_R Function

This function returns 'xmlns:r="http://xmlns.oracle.com/XDBResource.xsd"'.

Syntax

DBMS_XDB_CONSTANTS.NSPREFIXRESOURCE_R
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'xmlns:r="http://xmlns.oracle.com/XDBResource.xsd"'

193.2.19 NSPREFIX_XDB_XDB Function

This function returns 'http://xmlns.oracle.com/xdb'.

Syntax

DBMS_XDB_CONSTANTS.NSPREFIX_XDB_XDB
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'http://xmlns.oracle.com/xdb'

193.2.20 NSPREFIX_XMLINSTANCE_XSI Function

This function returns 'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"'.

Syntax

DBMS_XDB_CONSTANTS.NSPREFIX_XMLINSTANCE_XSI
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"'

193.2.21 NSPREFIX_XMLEDIFF_XD Function

This function returns 'xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"'.

Syntax

DBMS_XDB_CONSTANTS.NSPREFIX_XMLEDIFF_XD
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"'
193.2.22 NSPREFIX_XMLSCHEMA_XSD Function

This function returns 'xmlns:xsd="http://www.w3.org/2001/XMLSchema"'.

Syntax

```
DBMS_XDB_CONSTANTS.NSPREFIX_XMLSCHEMA_XSD
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'xmlns:xsd="http://www.w3.org/2001/XMLSchema"'

193.2.23 SCHEMAURL_ACL Function

This function returns 'http://xmlns.oracle.com/xdb/acl.xsd'.

Syntax

```
DBMS_XDB_CONSTANTS.SCHEMAURL_ACL
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdb/acl.xsd'

193.2.24 SCHEMAELEM_RES_ACL Function

This function returns 'http://xmlns.oracle.com/xdb/acl.xsd#acl'.

Syntax

```
DBMS_XDB_CONSTANTS.SCHEMAELEM_RES_ACL
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdb/acl.xsd#acl'

193.2.25 SCHEMAELEM_RESCONTENT_BINARY Function

This function returns 'http://xmlns.oracle.com/xdb/XDBSchema.xsd#binary'.

Syntax

```
DBMS_XDB_CONSTANTS.SCHEMAELEM_RESCONTENT_BINARY
RETURN VARCHAR2 DETERMINISTIC;
```

Return Value

'http://xmlns.oracle.com/xdb/XDBSchema.xsd#binary'
193.2.26 SCHEMAELEM_RESCONTENT_TEXT Function

**This function returns** 'http://xmlns.oracle.com/xdb/XDBSchema.xsd#text'.

**Syntax**

```sql
DBMS_XDB_CONSTANTS.SCHEMAELEM_RESCONTENT_TEXT
RETURN VARCHAR2 DETERMINISTIC;
```

**Return Value**

'http://xmlns.oracle.com/xdb/XDBSchema.xsd#text'

193.2.27 SCHEMAURL_RESOURCE Function

**This function returns** 'http://xmlns.oracle.com/xdb/XDBResource.xsd'.

**Syntax**

```sql
DBMS_XDB_CONSTANTS.SCHEMAURL_RESOURCE
RETURN VARCHAR2 DETERMINISTIC;
```

**Return Value**

'http://xmlns.oracle.com/xdb/XDBResource.xsd'

193.2.28 SCHEMAURL_XDBSCHEMA Function

**This function returns** 'http://xmlns.oracle.com/xdb/XDBSchema.xsd'.

**Syntax**

```sql
DBMS_XDB_CONSTANTS.SCHEMAURL_XDBSCHEMA
RETURN VARCHAR2 DETERMINISTIC;
```

**Return Value**

'http://xmlns.oracle.com/xdb/XDBSchema.xsd'

193.2.29 XDBSCHEMA_PREFIXES Function

**This function returns** 'xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:xdb=http://xmlns.oracle.com/xdb'.

**Syntax**

```sql
DBMS_XDB_CONSTANTS.XDBSCHEMA_PREFIXES
RETURN VARCHAR2 DETERMINISTIC;
```

**Return Value**

193.2.30 XSD_ATTRIBUTE Function

This function returns 'attribute'.

Syntax

DBMS_XDB_CONSTANTS.XSD_ATTRIBUTE
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'attribute'

193.2.31 XSD_COMPLEX_TYPE Function

This function returns 'complexType'.

Syntax

DBMS_XDB_CONSTANTS.XSD_COMPLEX_TYPE
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'complexType'

193.2.32 XSD_ELEMENT Function

This function returns 'element'.

Syntax

DBMS_XDB_CONSTANTS.XSD_ELEMENT
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'element'

193.2.33 XSD_GROUP Function

This function returns 'group'.

Syntax

DBMS_XDB_CONSTANTS.XSD_GROUP
RETURN VARCHAR2 DETERMINISTIC;

Return Value

'group'
The DBMS_XDB_REPOS package provides an interface to operate on the Oracle XML database Repository.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Summary of DBMS_XDB_REPOS Subprograms

See Also:
Oracle XML DB Developer's Guide for more information regarding:
- Using and managing repository resources
- ACL-based security management (controlling access to repository resources)
- Managing XLink and XInclude links
- Loading documents into the repository
- Creating, deleting, and managing resource metadata

194.1 DBMS_XDB_REPOS Overview

The DBMS_XDB_REPOS package lets you operate on the Oracle XML DB Repository to create, modify and delete resources, including managing security based on access control lists (ACLs). The interface provides both query and DML functions.

Using a combination of PL/SQL packages - DBMS_XDB_REPOS, DBMS_XDBZ, and DBMS_XDB_VERSION - you can create, delete, and rename documents and folders, move a file or folder within the folder hierarchy, set and change the access permissions on a file or folder, and initiate and manage versioning.

194.2 DBMS_XDB_REPOS Security Model

Owned by XDB, the DBMS_XDB_REPOS package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user. Subprograms that operate on the XDB Configuration will succeed only if the current user is SYS or XDB, or the current user has the XDBADMIN or DBA role.
194.3 DBMS_XDB_REPOS Constants

The DBMS_XDB_REPOS package defines several constants that can be used for specifying parameter values.

These constants are shown in the following table.

Table 194-1 DBMS_XDB_REPOS Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETERESOURCE</td>
<td>NUMBER</td>
<td>1</td>
<td>Deletes a resource; fails if the resource has children.</td>
</tr>
<tr>
<td>DELETERECURSIVE</td>
<td>NUMBER</td>
<td>2</td>
<td>Deletes a resource and its children, if any.</td>
</tr>
<tr>
<td>DELETEFORCE</td>
<td>NUMBER</td>
<td>3</td>
<td>Deletes the resource, even if the object it contains is invalid.</td>
</tr>
<tr>
<td>DELETERECURSIVEFORCE</td>
<td>NUMBER</td>
<td>4</td>
<td>Deletes a resource and its children, if any, even if the object it contains is invalid.</td>
</tr>
<tr>
<td>DELETERESMETADATACASCADE</td>
<td>NUMBER</td>
<td>1</td>
<td>Deletes the row in the metadata.</td>
</tr>
<tr>
<td>DELETERESMETADATANOCASCADE</td>
<td>NUMBER</td>
<td>2</td>
<td>Does not delete the row.</td>
</tr>
<tr>
<td>DEFAULT_LOCK_TIMEOUT</td>
<td>PLS_INTEGER</td>
<td>(60*60)</td>
<td>Timeout value (in seconds) of the webdav lock.</td>
</tr>
<tr>
<td>LINK_TYPE_HARD</td>
<td>NUMBER</td>
<td>1</td>
<td>Hard link of a folder to a resource.</td>
</tr>
<tr>
<td>LINK_TYPE_WEAK</td>
<td>NUMBER</td>
<td>2</td>
<td>Weak link of a folder to a resource.</td>
</tr>
<tr>
<td>LINK_TYPE_SYMBOLIC</td>
<td>NUMBER</td>
<td>3</td>
<td>Symbolic link of a folder to a resource.</td>
</tr>
</tbody>
</table>

194.4 Summary of DBMS_XDB_REPOS Subprograms

This table lists the DBMS_XDB_REPOS subprograms and briefly describes them.

Table 194-2 DBMS_XDB_REPOS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLCHECKPRIVILEGES Function</td>
<td>Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the 'owner' parameter.</td>
</tr>
<tr>
<td>APPENDRESOURCEMETADATAPROCEDURE</td>
<td>Takes in user-defined metadata either as a REF to XMLTYPE or an XMLTYPE and adds it to the desired resource.</td>
</tr>
<tr>
<td>CHANGEOWNER Procedure</td>
<td>Changes the owner of the resource/s to the specified owner.</td>
</tr>
<tr>
<td>CHANGEPRIVILEGES Function</td>
<td>Adds a specified ACE to a specified resource's ACL.</td>
</tr>
<tr>
<td>CHECKPRIVILEGES Function</td>
<td>Checks access privileges granted to the current user on the specified resource.</td>
</tr>
<tr>
<td>CREATEFOLDER Function</td>
<td>Creates a new folder resource in the hierarchy.</td>
</tr>
</tbody>
</table>
### Table 194-2  (Cont.) DBMS_XDB_REPOS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEOIDPATH Function</td>
<td>Creates a virtual path to the resource based on object ID</td>
</tr>
<tr>
<td>CREATETERESOURCE Functions</td>
<td>Creates a new resource</td>
</tr>
<tr>
<td>DELETERESOURCE Procedure</td>
<td>Deletes a resource from the hierarchy</td>
</tr>
<tr>
<td>DELETERESOURCEMETADATA Procedures</td>
<td>Deletes metadata from a resource (can be used for schema-based or</td>
</tr>
<tr>
<td></td>
<td>nonschema-based metadata)</td>
</tr>
<tr>
<td>EXISTSRESOURCE Function</td>
<td>Determines if a resource is the hierarchy, based on its absolute path</td>
</tr>
<tr>
<td>GETACLDOCUMENT Function</td>
<td>Retrieves ACL document that protects resource given its path name</td>
</tr>
<tr>
<td>GETCONTENTBLOB Function</td>
<td>Retrieves the contents of a resource returned as a BLOB</td>
</tr>
<tr>
<td>GETCONTENTCLOB Function</td>
<td>Retrieves the contents of a resource returned as a CLOB</td>
</tr>
<tr>
<td>GETCONTENTVARCHAR2 Function</td>
<td>Retrieves the contents of a resource returned as a string</td>
</tr>
<tr>
<td>GETCONTENTXMLREF Function</td>
<td>Retrieves the contents of a resource returned as a REF to an XMLTYPE</td>
</tr>
<tr>
<td>GETCONTENTXMLTYPE Function</td>
<td>Retrieves the contents of a resource returned as an XMLTYPE</td>
</tr>
<tr>
<td>GETLOCKTOKEN Procedure</td>
<td>Returns that resource’s lock token for the current user given a path to</td>
</tr>
<tr>
<td></td>
<td>a resource</td>
</tr>
<tr>
<td>GETPRIVILEGES Function</td>
<td>Gets all privileges granted to the current user on a specified resource</td>
</tr>
<tr>
<td>GETRESOID Function</td>
<td>Returns the object ID of the resource from its absolute path</td>
</tr>
<tr>
<td>GETXDB_TABLESPACE Function</td>
<td>Returns the current tablespace of the XDB (user)</td>
</tr>
<tr>
<td>HASBLOBCONTENT Function</td>
<td>Returns TRUE if the resource has BLOB content</td>
</tr>
<tr>
<td>HASCHARCONTENT Function</td>
<td>Returns TRUE if the resource has character content</td>
</tr>
<tr>
<td>HASXMLCONTENT Function</td>
<td>Returns TRUE if the resource has XML content</td>
</tr>
<tr>
<td>HASXMLREFERENCE Function</td>
<td>Returns TRUE if the resource has REF to XML content</td>
</tr>
<tr>
<td>ISFOLDER Function</td>
<td>Returns TRUE if the resource is a folder or container</td>
</tr>
<tr>
<td>LINK Procedures</td>
<td>Creates a link to an existing resource</td>
</tr>
<tr>
<td>LOCKRESOURCE Function</td>
<td>Gets a WebDAV-style lock on that resource given a path to that resource</td>
</tr>
<tr>
<td>PROCESSLINKS Procedure</td>
<td>Processes document links in the specified resource</td>
</tr>
<tr>
<td>PURGERESOURCEMETADATA Procedure</td>
<td>Deletes all user metadata from a resource</td>
</tr>
<tr>
<td>RENAMERESOURCE Procedure</td>
<td>Renames the XDB resource</td>
</tr>
<tr>
<td>SETACL Procedure</td>
<td>Sets the ACL on a specified resource</td>
</tr>
<tr>
<td>SPLITPATH Procedure</td>
<td>Splits the path into a parentpath and childpath</td>
</tr>
</tbody>
</table>
### Table 194-2  (Cont.) DBMS_XDB_REPOS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOUCHRESOURCE Procedure</td>
<td>Changes the modification time of the resource to the current time</td>
</tr>
<tr>
<td>UNLOCKRESOURCE Function</td>
<td>Unlocks the resource given a lock token and resource path</td>
</tr>
<tr>
<td>UPDATERESOURCEMETADATA Procedures</td>
<td>Updates metadata for a resource</td>
</tr>
</tbody>
</table>

#### 194.4.1 ACLCHECKPRIVILEGES Function

This function checks access privileges granted to the current user by specified ACL document by the OWNER of the resource. Returns positive integer if all privileges are granted.

**Syntax**

```sql
DBMS_XDB_REPOS.ACLCHECKPRIVILEGES(
    acl_path  IN  VARCHAR2,
    owner     IN  VARCHAR2,
    privs     IN  xmltype)
RETURN PLS_INTEGER;
```

**Parameters**

#### Table 194-3  ACLCHECKPRIVILEGES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_path</td>
<td>Absolute path in the Hierarchy for ACL document</td>
</tr>
<tr>
<td>owner</td>
<td>Resource owner name; the pseudo user &quot;DAV:owner&quot; is replaced by this user during ACL privilege resolution</td>
</tr>
<tr>
<td>privs</td>
<td>An XMLType instance of the privilege element specifying the requested set of access privileges. See description for CHECKPRIVILEGES Function.</td>
</tr>
</tbody>
</table>

#### 194.4.2 APPENDRESOURCEMETADATA Procedure

This procedure takes in user-defined metadata either as a REF to XMLTYPE or an XMLTYPE and adds it to the desired resource.

**Syntax**

```sql
DBMS_XDB_REPOS.APPENDRESOURCEMETADATA (
    abspath   IN  VARCHAR2,
    metadata  IN  XMLTYPE);

DBMS_XDB_REPOS.APPENDRESOURCEMETADATA (
    abspath   IN  VARCHAR2,
    metadata  IN  REF SYS.XMLTYPE);
```
Parameters

Table 194-4  APPENDRESOURCEMETADATA Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>metadata</td>
<td>Metadata can be schema based or nonschema-based. Schema-based metadata is stored in its own table.</td>
</tr>
</tbody>
</table>

Usage Notes

- In the case in which a REF is passed in, the procedure stores the REF in the resource, and the metadata is stored in a separate table. In this case you are responsible for populating the RESID column for the metadata table. Note that the REF passed in must be unique. In other words, there must not be a REF with the same value in the resource metadata, as this would violate uniqueness of properties. An error is thrown if users attempt to add a REF that already exists.

- In the case where the XMLTYPE is passed in, the data is parsed to determine if it is schema-based or not and stored accordingly.

194.4.3 CHANGEOWNER Procedure

This procedure changes the owner of the resource/s to the specified owner.

Syntax

```sql
DBMS_XDB_REPOS.CHANGEOWNER(
    abspath    IN   VARCHAR2,
    owner      IN   VARCHAR2,
    recurse    IN   BOOLEAN := FALSE);
```

Parameters

Table 194-5  CHANGEOWNER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>owner</td>
<td>New owner for the resource</td>
</tr>
<tr>
<td>recurse</td>
<td>If TRUE, recursively change owner of all resources in the folder tree</td>
</tr>
</tbody>
</table>

194.4.4 CHANGEPRIVILEGES Function

This function adds a specified ACE to a specified resource's ACL.

Syntax

```sql
DBMS_XDB_REPOS.CHANGEPRIVILEGES(
    res_path   IN    VARCHAR2,
    ace        IN    xmltype)
RETURN PLS_INTEGER;
```
**Parameters**

### Table 194-6   CHANGEPRIVILEGES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Path name of the resource for which privileges need to be changed</td>
</tr>
<tr>
<td>ace</td>
<td>An XMLType instance of the <code>&lt;ace&gt;</code> element which specifies the <code>&lt;principal&gt;</code>, the operation <code>&lt;grant&gt;</code> and the list of privileges</td>
</tr>
</tbody>
</table>

**Return Values**

A positive integer if the ACL was successfully modified.

**Usage Notes**

If no ACE with the same principal and the same operation (`grant/deny`) already exists in the ACL, the new ACE is added at the end of the ACL.

### 194.4.5 CHECKPRIVILEGES Function

This function checks access privileges granted to the current user on the specified resource.

**Syntax**

```sql
DBMS_XDB_REPOS.CHECKPRIVILEGES(
    res_path   IN  VARCHAR2,
    privs      IN  xmltype)
RETURN PLS_INTEGER;
```

**Parameters**

### Table 194-7   CHECKPRIVILEGES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Absolute path in the Hierarchy for resource</td>
</tr>
<tr>
<td>privs</td>
<td>An XMLType instance of the privilege element specifying the requested set of access privileges</td>
</tr>
</tbody>
</table>

**Return Values**

A positive integer if all requested privileges granted.

### 194.4.6 CREATEFOLDER Function

This function creates a new folder resource in the hierarchy.

**Syntax**

```sql
DBMS_XDB_REPOS.CREATEFOLDER(
    path   IN  VARCHAR2)
RETURN BOOLEAN;
```
Parameters

Table 194-8  CREATEFOLDER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name for the new folder</td>
</tr>
</tbody>
</table>

Return Values

TRUE if operation successful; FALSE, otherwise.

Usage Notes

The given path name’s parent folder must already exist in the hierarchy: if ‘/folder1/folder2’ is passed as the path parameter, then ‘/folder1’ must already exist.

194.4.7 CREATEOIDPATH Function

This function creates a virtual path to the resource based on object ID.

Syntax

```sql
DBMS_XDB_REPOS.CREATEOIDPATH(
    oid    IN   RAW)
RETURN VARCHAR2;
```

Parameters

Table 194-9  CREATEOIDPATH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>Object ID of the resource</td>
</tr>
</tbody>
</table>

194.4.8 CREATERESOURCE Functions

The functions create a new resource. The description of the overload options precede each version of the syntax

Syntax

Creates a new resource with a specified string as its contents:

```sql
DBMS_XDB_REPOS.CREATERESOURCE(
    abspath        IN  VARCHAR2,
    data           IN  VARCHAR2)
RETURN BOOLEAN;
```

Creates a new resource with a specified XMLType data as its contents:

```sql
DBMS_XDB_REPOS.CREATERESOURCE(
    abspath        IN  VARCHAR2,
    data           IN  SYS.XMLTYPE)
RETURN BOOLEAN;
```
Given a REF to an existing XMLType row, creates a resource whose contents point to that row. That row should not already exist inside another resource:

```sql
DBMS_XDB_REPOS.CREATERESOURCE(
    abspath        IN  VARCHAR2,
    datarow        IN  REF SYS.XMLTYPE)
RETURN BOOLEAN;
```

Creates a resource with a specified BLOB as its contents, and specifies character set of the source BLOB:

```sql
DBMS_XDB_REPOS.CREATERESOURCE(
    abspath        IN  VARCHAR2,
    data           IN  BLOB,
    csid           IN  NUMBER :=0)
RETURN BOOLEAN;
```

Creates a resource with a specified BFILE as its contents, and specifies character set of the source BFILE:

```sql
DBMS_XDB_REPOS.CREATERESOURCE ( 
    abspath        IN  VARCHAR2,
    data           IN  BFILE, 
    csid           IN  NUMBER :=0) 
RETURN BOOLEAN;
```

Creates a resource with a specified CLOB as its contents:

```sql
DBMS_XDB_REPOS.CREATERESOURCE ( 
    abspath        IN  VARCHAR2,
    data           IN  CLOB)
RETURN BOOLEAN;
```

Given a string, inserts a new resource into the hierarchy with the string as the contents:

```sql
DBMS_XDB_REPOS.CREATERESOURCE ( 
    abspath        IN  VARCHAR2,
    data           IN  VARCHAR2,
    schemaurl      IN  VARCHAR2 := NULL,
    elem           IN  VARCHAR2 := NULL)
RETURN BOOLEAN;
```

Given an XMLType and a schema URL, inserts a new resource into the hierarchy with the XMLType as the contents:

```sql
DBMS_XDB_REPOS.CREATERESOURCE ( 
    abspath        IN  VARCHAR2,
    data           IN  SYS.XMLTYPE, 
    schemaurl      IN  VARCHAR2 := NULL,
    elem           IN  VARCHAR2 := NULL)
RETURN BOOLEAN;
```
Parameters

Table 194-10  CREATERESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource to create. The path name's parent folder must already exist in the hierarchy. In other words, if /foo/bar.txt is passed in, then folder /foo must already exist.</td>
</tr>
<tr>
<td>data</td>
<td>String buffer containing new resource's contents. The data is parsed to check if it contains a schema-based XML document, and the contents are stored as schema-based in the schema's default table. Otherwise, it is saved as binary data.</td>
</tr>
<tr>
<td>datarow</td>
<td>REF to an XMLType row to be used as the contents</td>
</tr>
<tr>
<td>csid</td>
<td>Character set id of the document. Must be a valid Oracle ID; otherwise returns an error. If CSID is not specified, or if a zero CSID is specified, then the character set id of the document is determined as follows:</td>
</tr>
<tr>
<td>schemaurl</td>
<td>For XML data, schema URL data conforms to (default NULL)</td>
</tr>
<tr>
<td>elem</td>
<td>Element name (default NULL)</td>
</tr>
</tbody>
</table>

Return Values

TRUE if operation successful; FALSE, otherwise.

194.4.9 DELETERESOURCE Procedure

This procedure deletes a resource from the hierarchy.

Syntax

DBMS_XDB_REPOS.DELETERESOURCE(
    path    IN     VARCHAR2,
    delete_option IN   PLS_INTEGER);

Parameters

Table 194-11  DELETERESOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name of the resource to delete</td>
</tr>
<tr>
<td>delete_option</td>
<td>The option that controls how a resource is deleted:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
194.4.10 DELETRESOURCENETADATA Procedures

This procedure takes in a resource by absolute path and removes either the schema-based metadata identified by the REF, or the metadata identified by the namespace and name combination, which can be either schema-based or non-schema based. It also takes an additional (optional) parameter that specifies how to delete it. This parameter is only relevant for schema-based resource metadata that needs to be deleted. For non-schema based metadata, this parameter is ignored.

Syntax

Can be used only for schema-based metadata:

DBMS_XDB_REPOS.DELETRESOURCENETADATA (  
  abspath IN VARCHAR2,  
  metadata IN REF SYS.XMLTYPE,  
  delete_option IN pls_integer :=  
  DBMS_XDB_REPOS.DELETERESOURCE_METADATA_CASCADE);

Can be used for schema-based or nonschema-based metadata:

DBMS_XDB_REPOS.DELETRESOURCENETADATA (  
  abspath IN VARCHAR2,  
  metadatans IN VARCHAR2,  
  mettadataname IN VARCHAR2,  
  delete_option IN pls_integer :=  
  DBMS_XDB_REPOS.DELETERESOURCE_METADATA_CASCADE);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>metadata</td>
<td>REF to the piece of metadata (schema based) to be deleted</td>
</tr>
<tr>
<td>mettadatans</td>
<td>Namespace of the metadata fragment to be removed</td>
</tr>
<tr>
<td>mettadataname</td>
<td>Local name of the metadata fragment to be removed</td>
</tr>
<tr>
<td>delete_option</td>
<td>Only applicable for schema-based metadata, this can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>· DELETE_RES_METADATA_CASCADE - deletes the corresponding row in the metadata table</td>
</tr>
<tr>
<td></td>
<td>· DELETE_RES_METADATA_NOCASCADE - does not delete the row in the metadata table</td>
</tr>
</tbody>
</table>


194.4.11 EXISTSRESOURCE Function

This function indicates if a resource is in the hierarchy. It matches the resource by a string that represents its absolute path.

Syntax

```sql
DBMS_XDB_REPOS.EXISTSRESOURCE(
    abspath    IN    VARCHAR2)
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Path name of the resource whose ACL document is required</td>
</tr>
</tbody>
</table>

Return Values

`TRUE` if the resource is found.

194.4.12 GETACLDOCUMENT Function

This function retrieves ACL document that protects resource given its path name.

Syntax

```sql
DBMS_XDB_REPOS.GETACLDOCUMENT(
    abspath  IN  VARCHAR2)
RETURN sys.xmltype;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Path name of the resource whose ACL document is required</td>
</tr>
</tbody>
</table>

Return Values

The `XMLType` for ACL document.

194.4.13 GETCONTENTBLOB Function

This function retrieves the contents of a resource returned as a `BLOB`.

Syntax

```sql
DBMS_XDB_REPOS.GETCONTENTBLOB(
    abspath    IN     VARCHAR2,
    csid       OUT    PLS_INTEGER,
```

```sql
data;
```
locksrc  IN  BOOLEAN := FALSE)
RETURN BLOB;

Parameters

Table 194-15  GETCONTENTBLOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>csid</td>
<td>If TRUE, lock and return the source LOB. If FALSE, return a temp LOB copy.</td>
</tr>
<tr>
<td>locksrc</td>
<td>Contents of the resource as a BLOB</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a BLOB.

194.4.14 GETCONTENTCLOB Function

This function gets the contents of a resource returned as a CLOB.

Syntax

DBMS_XDB_REPOS.GETCONTENTCLOB(
    abspath  IN  VARCHAR2,
RETURN CLOB;

Parameters

Table 194-16  GETCONTENTCLOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a CLOB.

194.4.15 GETCONTENTVARCHAR2 Function

This function gets the contents of a resource returned as a string.

Syntax

DBMS_XDB_REPOS.GETCONTENTVARCHAR2(
    abspath  IN  VARCHAR2,
RETURN BLOB;
Parameters

Table 194-17  GETCONTENTVARCHAR2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a string.

194.4.16 GETCONTENTXMLREF Function

This function retrieves the contents of a resource returned as a REF to an XMLTYPE.

Syntax

```
DBMS_XDB_REPOS.GETCONTENTXMLREF(
    abspath IN VARCHAR2,
    RETURN SYS.XMLTYPE;
```

Parameters

Table 194-18  GETCONTENTXMLREF Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

The contents of the resource as a REF to an XMLTYPE.

194.4.17 GETCONTENTXMLTYPE Function

This function retrieves the contents of a resource returned as an XMLTYPE.

Syntax

```
DBMS_XDB_REPOS.GETCONTENTXMLTYPE(
    abspath IN VARCHAR2,
    RETURN SYS.XMLTYPE;
```

Parameters

Table 194-19  GETCONTENTXMLTYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>
Return Values
The contents of the resource as an XMLTYPE.

194.4.18 GETLOCKTOKEN Procedure
Given a path to a resource, this procedure returns that resource's lock token for the current user.

Syntax

```
DBMS_XDB_REPOS.GETLOCKTOKEN(
    path         IN      VARCHAR2,
    locktoken    OUT     VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name to the resource</td>
</tr>
<tr>
<td>locktoken</td>
<td>Logged-in user's lock token for the resource</td>
</tr>
</tbody>
</table>

Usage Notes
The user must have `READPROPERTIES` privilege on the resource.

194.4.19 GETPRIVILEGES Function
This function gets all privileges granted to the current user on a specified resource.

Syntax

```
DBMS_XDB_REPOS.GETPRIVILEGES(
    res_path    IN     VARCHAR2)
RETURN sys.xmltype;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Absolute path in the hierarchy of the resource</td>
</tr>
</tbody>
</table>

Return Values
An XMLType instance of `<privilege>` element, which contains the list of all leaf privileges granted on this resource to the current user.
194.4.20 GETRESOID Function

The GETRESOID function returns the object ID of the resource from its absolute path.

Syntax

```
DBMS_XDB_REPOS.GETRESOID(
    abspath IN VARCHAR2)
RETURN RAW;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

NULL if the resource is not present.

194.4.21 GETXDB_TABLESPACE Function

This function returns the current tablespace of the XDB (user).

Syntax

```
DBMS_XDB_REPOS.GETXDB_TABLESPACE
RETURN VARCHAR2;
```

194.4.22 HASBLOBCONTENT Function

This function returns TRUE if the resource has BLOB content.

Syntax

```
DBMS_XDB_REPOS.HASBLOBCONTENT(
    abspath IN VARCHAR2)
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource has BLOB content.
194.4.23 HASCHARCONTENT Function

This function returns TRUE if the resource has character content.

Syntax

```
DBMS_XDB_REPOS.HASCHARCONTENT
    abspath    IN     VARCHAR2)
RETURN BOOLEAN;
```

Parameters

Table 194-24  HASCHARCONTENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource has character content.

194.4.24 HASXMLCONTENT Function

This function returns TRUE if the resource has XML content.

Syntax

```
DBMS_XDB_REPOS.HASXMLCONTENT
    abspath    IN     VARCHAR2)
RETURN BOOLEAN;
```

Parameters

Table 194-25  HASXMLCONTENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource has XML content.

194.4.25 HASXMLREFERENCE Function

This function returns TRUE if the resource has a REF to XML content.

Syntax

```
DBMS_XDB_REPOS.HASXMLREFERENCE
    abspath    IN     VARCHAR2)
RETURN BOOLEAN;
```
Parameters

Table 194-26  HASXMLREFERENCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE resource has a REF to XML content.

194.4.26 ISFOLDER Function

This function returns TRUE if the resource is a folder or container.

Syntax

DBMS_XDB_REPOS.ISFOLDER
   abspath IN VARCHAR2)
RETURN BOOLEAN;

Parameters

Table 194-27  ISFOLDER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

Return Values

TRUE if the resource is a folder or container.

194.4.27 LINK Procedures

This procedure creates from a specified folder to a specified resource.

Syntax

DBMS_XDB_REPOS.LINK(
   srcpath    IN   VARCHAR2,
   linkfolder IN   VARCHAR2,
   linkname   IN   VARCHAR2);

DBMS_XDB_REPOS.LINK(
   srcpath    IN   VARCHAR2,
   linkfolder IN   VARCHAR2,
   linkname   IN   VARCHAR2,
   linktype   IN   PLS_INTEGER := DBMS_XDB_REPOS.LINK_TYPE_HARD);
Parameters

**Table 194-28  LINK Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcpath</td>
<td>Path name of the resource to which a link is created</td>
</tr>
<tr>
<td>linkfolder</td>
<td>Folder in which the new link is placed</td>
</tr>
<tr>
<td>linkname</td>
<td>Name of the new link</td>
</tr>
<tr>
<td>linktype</td>
<td>Type of link to be created:</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDB.LINK_TYPE_HARD (default)</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDB.LINK_TYPE_WEAK</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDB.LINK_TYPE_SYMBOLIC</td>
</tr>
</tbody>
</table>

194.4.28 LOCKRESOURCE Function

Given a path to a resource, this function gets a WebDAV-style lock on that resource.

**Syntax**

```sql
DBMS_XDB_REPOS.LOCKRESOURCE(
    path      IN  VARCHAR2,
    depthzero IN  BOOLEAN,
    shared    IN  boolean)
RETURN BOOLEAN;
```

**Parameters**

**Table 194-29  LOCKRESOURCE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name of the resource to lock.</td>
</tr>
<tr>
<td>depthzero</td>
<td>Currently not supported</td>
</tr>
<tr>
<td>shared</td>
<td>Passing TRUE obtains a shared write lock</td>
</tr>
</tbody>
</table>

**Return Values**

TRUE if successful.

**Usage Notes**

The user must have UPDATE privileges on the resource.
194.4.29 PROCESSLINKS Procedure

This procedure processes document links in the specified resource.

Syntax

```
DBMS_XDB_REPOS.PURGERESOURCEMETADATA(
  abspath  IN  VARCHAR2,
  recurse  IN  BOOLEAN := FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource. If the path is a folder, use the recurse flag.</td>
</tr>
<tr>
<td>recurse</td>
<td>Used only if abspath specifies a folder. If TRUE, process links of all resources in the folder hierarchy rooted at the specified resource. If FALSE, process links of all documents in this folder only.</td>
</tr>
</tbody>
</table>

194.4.30 PURGERESOURCEMETADATA Procedure

This procedure deletes all user metadata from a resource. Schema-based metadata is removed in cascade mode, rows being deleted from the corresponding metadata tables.

Syntax

```
DBMS_XDB_REPOS.PURGERESOURCEMETADATA(
  abspath  IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

194.4.31 RENAMERESOURCE Procedure

This procedure renames the XDB resource.

Syntax

```
DBMS_XDB_REPOS.RENAMERESOURCE(
  srcpath    IN  VARCHAR2,
  destfolder IN  VARCHAR2,
  newname    IN  VARCHAR2);
```
Parameters

Table 194-32  RENAMERESOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcpath</td>
<td>Absolute path in the Hierarchy for the source resource destination folder</td>
</tr>
<tr>
<td>destfolder</td>
<td>Absolute path in the Hierarchy for the destination folder</td>
</tr>
<tr>
<td>newname</td>
<td>Name of the child in the destination folder</td>
</tr>
</tbody>
</table>

194.4.32 SETACL Procedure

This procedure sets the ACL on a specified resource to be the ACL specified by path.

Syntax

```sql
DBMS_XDB_REPOS.SETACL(
    res_path   IN  VARCHAR2,
    acl_path   IN  VARCHAR2);
```

Parameters

Table 194-33  SETACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res_path</td>
<td>Absolute path in the Hierarchy for resource</td>
</tr>
<tr>
<td>acl_path</td>
<td>Absolute path in the Hierarchy for ACL</td>
</tr>
</tbody>
</table>

Usage Notes

The user must have `<write-acl>` privileges on the resource.

194.4.33 SPLITPATH Procedure

This procedure splits the path into a parentpath and childpath.

Syntax

```sql
DBMS_XDB_REPOS.SPLITPATH(
    abspath     IN  VARCHAR2,
    parentpath  OUT VARCHAR2,
    childpath   OUT VARCHAR2);
```

Parameters

Table 194-34  SPLITPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path to be split</td>
</tr>
<tr>
<td>parentpath</td>
<td>Parentpath</td>
</tr>
</tbody>
</table>
194.4.34 TOUCHRESOURCE Procedure

This procedure changes the modification time of the resource to the current time.

Syntax

```sql
DBMS_XDB_REPOS.TOUCHRESOURCE
    abspath IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath_path</td>
<td>Absolute path of the resource</td>
</tr>
</tbody>
</table>

194.4.35 UNLOCKRESOURCE Function

This function unlocks the resource given a lock token and a path to the resource.

Syntax

```sql
DBMS_XDB_REPOS.UNLOCKRESOURCE(
    path IN VARCHAR2,
    deltoken IN VARCHAR2)
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path name to the resource</td>
</tr>
<tr>
<td>deltoken</td>
<td>Lock token to be removed</td>
</tr>
</tbody>
</table>

Return Values

TRUE if operation successful.

Usage Notes

The user must have UPDATE privileges on the resource.
194.4.36 UPDATERESOURCEMETADATA Procedures

This procedure updates metadata for a resource.

The procedure takes in a resource identified by absolute path and the metadata in it to replace identified by its REF. It replaces that piece of metadata with user-defined metadata which is either in the form of a REF to XMLTYPE or an XMLTYPE.

Syntax

Can be used to update schema-based metadata only. The new metadata must be schema-based:

```sql
DBMS_XDB_REPOS.UPDATERESOURCEMETADATA(
    abspath  IN VARCHAR2,
    oldmetadata    IN   REF SYS.XMLTYPE,
    newmetadata    IN   REF SYS.XMLTYPE)
```

Can be used to update schema-based metadata only. The new metadata must be schema-based or nonschema-based:

```sql
DBMS_XDB_REPOS.UPDATERESOURCEMETADATA(
    abspath        IN   VARCHAR2,
    oldmetadata    IN   REF SYS.XMLTYPE,
    newmetadata    IN   XMLTYPE);
```

Can be used for both schema-based and nonschema-based metadata:

```sql
DBMS_XDB_REPOS.UPDATERESOURCEMETADATA(
    abspath        IN   VARCHAR2,
    oldns          IN   VARCHAR2,
    oldname IN VARCHAR,
    newmetadata IN XMLTYPE);
```

Can be used for both schema-based or nonschema-based metadata. New metadata must be schema-based:

```sql
DBMS_XDB_REPOS.UPDATERESOURCEMETADATA(
    abspath       IN   VARCHAR2,
    oldns         IN   VARCHAR2,
    oldname       IN   VARCHAR,
    newmetadata   IN   REF SYS.XMLTYPE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>Absolute path of the resource</td>
</tr>
<tr>
<td>oldmetadata</td>
<td>REF to the old of metadata</td>
</tr>
<tr>
<td>newmetadata</td>
<td>REF to the new, replacement metadata (can be either schema-based or nonschema-based depending on the overload)</td>
</tr>
</tbody>
</table>
Table 194-37  (Cont.) UPDATERESOURCEMETADATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldns</td>
<td>Namespace identifying old metadata</td>
</tr>
<tr>
<td>oldname</td>
<td>Local name identifying old metadata</td>
</tr>
</tbody>
</table>

Usage Notes

In the case of `REF`, it stores the `REF` in the resource and the metadata is stored in a separate table. Uniqueness of `REFs` is enforced. In the case where the `XMLTYPE` is passed in, data is parsed to determine if it is schema-based or not and is stored accordingly.
Oracle XML DB versioning interfaces are found in the DBMS_XDB_VERSION package. Functions and procedures of DBMS_XDB_VERSION help to create a VCR and manage the versions in the version history.

This chapter contains the following topics:

- Security Model
- Summary of DBMS_XDB_VERSION Subprograms

See Also:
Oracle XML DB Developer's Guide

195.1 DBMS_XDB_VERSION Security Model

Owned by XDB, the DBMS_XDB_VERSION package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

195.2 Summary of DBMS_XDB_VERSION Subprograms

This table lists the DBMS_XDB_VERSION subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKIN Function</td>
<td>Checks in a checked-out VCR and returns the resource id of the newly-created version</td>
</tr>
<tr>
<td>CHECKOUT Procedure</td>
<td>Checks out a VCR before updating or deleting it</td>
</tr>
<tr>
<td>GETCONTENTSBLOBBYRESID Function</td>
<td>Obtain contents as a BLOB</td>
</tr>
<tr>
<td>GETCONTENTSCLOBBYRESID Function</td>
<td>Obtain contents as a CLOB</td>
</tr>
<tr>
<td>GETCONTENTSXMLBYRESID Function</td>
<td>Obtain contents as an XMLType</td>
</tr>
<tr>
<td>GETPREDECESSORS Function</td>
<td>Retrieves the list of predecessors by path name</td>
</tr>
<tr>
<td>GETPRESBYRESID Function</td>
<td>Retrieves the list of predecessors by resource id</td>
</tr>
<tr>
<td>GETRESOURCEBYRESID Function</td>
<td>Obtains the resource as an XMLType, given the resource object ID</td>
</tr>
<tr>
<td>GETSUCCESSORS Function</td>
<td>Retrieves the list of successors by path name</td>
</tr>
</tbody>
</table>
### Table 195-1  (Cont.) DBMS_XDB_VERSION Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETSUCCSBYRESID Function</td>
<td>Retrieves the list of successors by resource id</td>
</tr>
<tr>
<td>MAKEVERSIONED Function</td>
<td>Turns a regular resource whose path name is given into a version-controlled resource</td>
</tr>
<tr>
<td>UNCHECKOUT Function</td>
<td>Checks in a checked-out resource, returns the resource id of the version before the resource is checked out</td>
</tr>
</tbody>
</table>

#### 195.2.1 CHECKIN Function

This function checks in a checked-out VCR and returns the resource id of the newly-created version.

**Syntax**

```sql
DBMS_XDB_VERSION.CHECKIN(
    pathname VARCHAR2)
RETURN DBMS_XDB.resid_type;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathname</td>
<td>The path name of the checked-out resource.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This is not an auto-commit SQL operation. CHECKIN Function doesn't have to take the same path name that was passed to CHECKOUT Procedure operation. However, the CHECKIN Function path name and the CHECKOUT Procedure path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to CHECKIN Function because the old name is either invalid or is currently bound with a different resource. Exception is raised if the path name does not exist. If the path name has been changed, the new path name must be used to CHECKIN Function the resource.

#### 195.2.2 CHECKOUT Procedure

This procedure checks out a VCR before updating or deleting it.

**Syntax**

```sql
DBMS_XDB_VERSION.Checkout(
    pathname VARCHAR2);
```
Parameters

Table 195-3  CHECKOUT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathname</td>
<td>The path name of the VCR to be checked out.</td>
</tr>
</tbody>
</table>

Usage Notes

This is not an auto-commit SQL operation. Two users of the same workspace cannot CHECKOUT Procedure the same VCR at the same time. If this happens, one user must rollback. As a result, it is good practice to commit the CHECKOUT Procedure operation before updating a resource and avoid loss of the update if the transaction is rolled back. An exception is raised if the given resource is not a VCR, if the VCR is already checked out, if the resource doesn't exist.

195.2.3 GETCONTENTSBLOBBYRESID Function

This function obtain contents as a BLOB.

Syntax

```sql
DBMS_XDB_VERSION.GETCONTENTSBLOBBYRESID(
    resid     DBMS_XDB.resid_type)
RETURN BLOB;
```

Parameters

Table 195-4  GETCONTENTSBLOBBYRESID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resid</td>
<td>The resource id.</td>
</tr>
</tbody>
</table>

195.2.4 GETCONTENTSCLOBBYRESID Function

This function obtains contents as a CLOB.

Syntax

```sql
DBMS_XDB_VERSION.GETCONTENTSCLOBBYRESID(
    resid     DBMS_XDB.resid_type)
RETURN CLOB;
```

Parameters

Table 195-5  GETCONTENTSCLOBBYRESID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resid</td>
<td>The resource id.</td>
</tr>
</tbody>
</table>
195.2.5 GETCONTENTXMLBYRESID Function

This function obtains contents as an XMLType.

Syntax

```
DBMS_XDB_VERSION.GETCONTENTXMLBYRESID(
   resid   DBMS_XDB.resid_type)
RETURN XMLType;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resid</td>
<td>The resource id.</td>
</tr>
</tbody>
</table>

Return Values

If the contents are not valid XML, returns NULL.

195.2.6 GETPREDECESSORS Function

This function retrieves the list of predecessors by the path name.

Syntax

```
DBMS_XDB_VERSION.GETPREDECESSORS(
   pathname   VARCHAR2)
RETURN resid_list_type;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathname</td>
<td>The path name of the resource.</td>
</tr>
</tbody>
</table>

Return Values

An exception is raised if `pathname` is illegal.

195.2.7 GETPREDSDBYRESID Function

This function retrieves the list of predecessors by resource id.

Syntax

```
DBMS_XDB_VERSION.GETPREDSDBYRESID(
   resid   resid_type)
RETURN resid_list_type;
```
Parameters

Table 195-8  GETPREDSBYRESID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resid</td>
<td>The resource id.</td>
</tr>
</tbody>
</table>

Usage Notes

Getting predecessors by RESID is more efficient than by pathname.

Exceptions

An exception is raised if the RESID is illegal.

195.2.8 GETRESOURCEBYRESID Function

This function obtains the resource as an XMLType, given the resource object ID. Because the system does not create a path name for versions, this function is useful for retrieving the resource using its resource id.

Syntax

```
DBMS_XDB_VERSION.GETRESOURCEBYRESID(
    resid resid_type)
RETURN XMLType;
```

Parameters

Table 195-9  GETRESOURCEBYRESID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resid</td>
<td>The resource id.</td>
</tr>
</tbody>
</table>

195.2.9 GETSUCCESSORS Function

Given a version resource or a VCR, this function retrieves the list of the successors of the resource by the path name.

Syntax

```
DBMS_XDB_VERSION.GETSUCCESSORS(
    pathname VARCHAR2)
RETURN resid_list_type;
```

Parameters

Table 195-10  GETSUCCESSORS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathname</td>
<td>The path name of the resource.</td>
</tr>
</tbody>
</table>
Usage Notes
Getting successors by RESID is more efficient than by pathname.

Exceptions
An exception is raised if the pathname is illegal.

195.2.10 GETSUCCSBYRESID Function

This function retrieves the list of the successors of the resource by resource id using version resource or VCR.

Syntax

```sql
DBMS_XDB_VERSION.GETSUCCSBYRESID(
    resid    resid_type
) RETURN resid_list_type;
```

Parameters

Table 195-11  GETSUCCSBYRESID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resid</td>
<td>The resource id.</td>
</tr>
</tbody>
</table>

Usage Notes
Getting successors by RESID is more efficient than by pathname.

Exceptions
An exception is raised if the pathname is illegal.

195.2.11 MAKEVERSIONED Function

This function turns a regular resource whose path name is given into a version-controlled resource. This new resource is then put under version control. All other path names continue to refer to the original resource.

Syntax

```sql
DBMS_XDB_VERSION.MAKEVERSIONED(
    pathname   VARCHAR2
) RETURN DBMS_XDB.resid_type;
```

Parameters

Table 195-12  MAKEVERSIONED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathname</td>
<td>The path name of the resource to be put under version control.</td>
</tr>
</tbody>
</table>
Return Values
This function returns the resource ID of the first version, or root, of the VCR.

Usage Notes
If two or more path names are bound with the same resource, a copy of the resource is created, and the given path name is bound with the newly-created copy.

This is not an auto-commit SQL operation. An exception is raised if the resource doesn't exist.
- This call is legal for VCR, and neither exception nor warning is raised.
- This call is illegal for folder, version history, version resource, and ACL.
- No support for Schema-based resources is provided.

195.2.12 UNCHECKOUT Function
This function checks-in a checked-out resource and returns the resource id of the version before the resource is checked out.

Syntax
```sql
DBMS_XDB_VERSION.UNCHECKOUT(
    pathname    VARCHAR2)
RETURN DBMS_XDB.resid_type;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathname</td>
<td>The path name of the checked-out resource.</td>
</tr>
</tbody>
</table>

Usage Notes
This is not an auto-commit SQL operation. The UNCHECKOUT Function does not have to take the same path name that was passed to the operation by the CHECKOUT Procedure. However, the UNCHECKOUT Function path name and the CHECKOUT Procedure path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to UNCHECKOUT Function, because the old name is either invalid or is currently bound with a different resource. If the path name has been changed, the new path name must be used to UNCHECKOUT Function the resource.

Exceptions
An exception is raised if the path name doesn't exist.
The `DBMS_XDBRESOURCE` package provides the interface to operate on the resource's metadata and contents.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of `DBMS_XDBRESOURCE` Subprograms

**See Also:**
Oracle XML DB Developer's Guide for examples of "Using `DBMS_XDBRESOURCE`"

### 196.1 DBMS_XDBRESOURCE Overview

The `DBMS_XDBRESOURCE` package provides routines to get and set the resource's metadata and contents. To take advantage of the DOM traversal facility, provided in `DBMS_XMLDOM` package, an XDBResource instance could be converted to a DOMDocument type by using `DBMS_XDBRESOURCE.MAKEDOCUMENT` routine.

### 196.2 DBMS_XDBRESOURCE Security Model

Owned by XDB, the `DBMS_XDBRESOURCE` package must be created by SYS or XDB. The `EXECUTE` privilege is granted to `PUBLIC`. Subprograms in this package are executed using the privileges of the current user.

### 196.3 Summary of DBMS_XDBRESOURCE Subprograms

This table lists the `DBMS_XDBRESOURCE` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREERESOURCE Procedure</td>
<td>Frees any memory associated with an XDBResource</td>
</tr>
<tr>
<td>GETACL Function</td>
<td>Given an XDBResource, returns its ACL as string</td>
</tr>
<tr>
<td>GETACLDOCFROMRES Function</td>
<td>Returns the ACL Document for the given resource as XMLType</td>
</tr>
<tr>
<td>GETAUTHOR Function</td>
<td>Given an XDBResource, returns its author</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GETCHARACTERSET Function</td>
<td>Given an XDBResource, returns its character set</td>
</tr>
<tr>
<td>GETCOMMENT Function</td>
<td>Given an XDBResource, returns its comment</td>
</tr>
<tr>
<td>GETCONTENTBLOB Function</td>
<td>Returns the contents of the resource as a BLOB</td>
</tr>
<tr>
<td>GETCONTENTCLOB Function</td>
<td>Returns the contents of the resource as a CLOB</td>
</tr>
<tr>
<td>GETCONTENTREF Function</td>
<td>Returns the contents of the resource as an XMLTypeRef</td>
</tr>
<tr>
<td>GETCONTENTTYPE Function</td>
<td>Given an XDBResource, returns its content-type</td>
</tr>
<tr>
<td>GETCONTENTXML Function</td>
<td>Returns the contents of the resource as XML</td>
</tr>
<tr>
<td>GETCONTENTVARCHAR2 Function</td>
<td>Returns the contents of the resource as a string</td>
</tr>
<tr>
<td>GETCREATIONDATE Function</td>
<td>Given an XDBResource, returns its creation date</td>
</tr>
<tr>
<td>GETCREATOR Function</td>
<td>Given an XDBResource, returns its creator</td>
</tr>
<tr>
<td>GETCUSTOMMETADATA Function</td>
<td>Returns the requested custom metadata given the xpath and namespace to the metadata</td>
</tr>
<tr>
<td>GETDISPLAYNAME Function</td>
<td>Given an XDBResource, returns its display name</td>
</tr>
<tr>
<td>GETLANGUAGE Function</td>
<td>Given an XDBResource, returns its language</td>
</tr>
<tr>
<td>GETLASTMODIFIER Function</td>
<td>Given an XDBResource, returns its last modifier</td>
</tr>
<tr>
<td>GETMODIFICATIONDATE Function</td>
<td>Given an XDBResource, returns its modification date</td>
</tr>
<tr>
<td>GETOWNER Function</td>
<td>Given an XDBResource, returns its owner.</td>
</tr>
<tr>
<td>GETREFCOUNT Function</td>
<td>Given an XDBResource, returns its reference count</td>
</tr>
<tr>
<td>GETVERSIONID Function</td>
<td>Given an XDBResource, returns its version ID</td>
</tr>
<tr>
<td>HASACLCHANGED Function</td>
<td>Returns TRUE if the ACL of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASAUTHORCHANGED Function</td>
<td>Returns TRUE if the ACL of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCUSTOMMETADATACHANGED Function</td>
<td>Returns TRUE if custom-metadata for this XPath has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCHANGED Function</td>
<td>Returns TRUE if the element or attribute represented by the given XPath has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCHARACTERSETCHANGED Function</td>
<td>Returns TRUE if the character set of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCOMMENTCHANGED Function</td>
<td>Returns TRUE if the comment of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCONTENTCHANGED Function</td>
<td>Returns TRUE if the contents of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCONTENTTYPECHANGED Function</td>
<td>Returns TRUE if the content-type of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCREATIONDATECHANGED Function</td>
<td>Returns TRUE if the creation date of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCREATORCHANGED Function</td>
<td>Returns TRUE if the creator of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASCUSTOMMETADATACHANGED Function</td>
<td>Returns TRUE if custom-metadata for this XPath has changed, FALSE otherwise</td>
</tr>
</tbody>
</table>
### Table 196-1  (Cont.) DBMS_XDBRESOURCE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASDISPLAYNAMECHANGED Function</td>
<td>Returns TRUE if the display name of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASLANGUAGECHANGED Function</td>
<td>Returns TRUE if the language of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASLASTMODIFIERCHANGED Function</td>
<td>Returns TRUE if the last modifier of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASMODIFICATIONDATECHANGED Function</td>
<td>Returns TRUE if the modification date of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASOWNERCHANGED Function</td>
<td>Returns TRUE if the owner of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASREFCOUNTCHANGED Function</td>
<td>Returns TRUE if the reference count of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>HASVERSIONIDCHANGED Function</td>
<td>Returns TRUE if the version ID of the given resource has changed, FALSE otherwise</td>
</tr>
<tr>
<td>ISFOLDER Function</td>
<td>Returns TRUE if the given resource is a folder, FALSE otherwise</td>
</tr>
<tr>
<td>ISNULL Function</td>
<td>Returns TRUE if input resource is NULL, FALSE otherwise</td>
</tr>
<tr>
<td>MAKEDOCUMENT Function</td>
<td>Converts the XDBResource to a DOMDocument which can be operated on using the XMLDOM interface</td>
</tr>
<tr>
<td>SAVE Procedure</td>
<td>Updates the resource with any modifications</td>
</tr>
<tr>
<td>SETACL Procedure</td>
<td>Sets the ACL of the given XDBResource to the path specified</td>
</tr>
<tr>
<td>SETAUTHOR Procedure</td>
<td>Sets the author of the given XDBResource to the specified string</td>
</tr>
<tr>
<td>SETCHARACTERSET Procedure</td>
<td>Sets the character set of the given XDBResource to a specified character set</td>
</tr>
<tr>
<td>SETCOMMENT Procedure</td>
<td>Sets a comment associated with the given XDBResource</td>
</tr>
<tr>
<td>SETCONTENT Procedures</td>
<td>Replaces the contents of the given resource with the given CLOB</td>
</tr>
<tr>
<td>SETCONTENTTYPE Procedure</td>
<td>Sets the content-type of the given XDBResource</td>
</tr>
<tr>
<td>SETCUSTOMMETADATA Procedure</td>
<td>Sets the custom metadata specified by the XPath and namespace to new data</td>
</tr>
<tr>
<td>SETDISPLAYNAME Procedure</td>
<td>Sets the display name of the given XDBResource</td>
</tr>
<tr>
<td>SETLANGUAGE Procedure</td>
<td>Sets the language of the given XDBResource</td>
</tr>
<tr>
<td>SETOWNER Procedure</td>
<td>Sets the owner of the given XDBResource</td>
</tr>
</tbody>
</table>
196.3.1 FREERESOURCE Procedure

This procedure frees any memory associated with an XDBResource.

Syntax

DBMS_XDBRESOURCE.FREERESOURCE (
    res   IN    XDBResource)
RETURN VARCHAR2;

Parameters

Table 196-2 FREERESOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource to free</td>
</tr>
</tbody>
</table>

196.3.2 GETACL Function

Given an XDBResource, this function returns its ACL as string.

Syntax

DBMS_XDBRESOURCE.GETACL (
    res   IN    XDBResource)
RETURN VARCHAR2;

Parameters

Table 196-3 GETACL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.3 GETACLDOCFROMRES Function

This function returns the ACL Document for the given resource as XMLType.

Syntax

DBMS_XDBRESOURCE.GETACLDOCFROMRES (
    res   IN    XDBResource)
RETURN SYS.XMLTYPE;

Parameters

Table 196-4 GETACLDOCFROMRES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.4 GETAUTHOR Function

Given an XDBResource, this function returns its author.

Syntax

```sql
DBMS_XDBRESOURCE.GETAUTHOR (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.5 GETCHARACTERSET Function

Given an XDBResource, this function returns its characterset.

Syntax

```sql
DBMS_XDBRESOURCE.GETCHARACTERSET (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.6 GETCOMMENT Function

Given an XDBResource, this function returns its comment.

Syntax

```sql
DBMS_XDBRESOURCE.GETCOMMENT (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.7 GETCONTENTBLOB Function

This function returns the contents of the resource as a BLOB.

Syntax

```
DBMS_XDBRESOURCE.GETCONTENTBLOB (  
    res    IN     XDBResource,  
    csid   OUT    PLS_INTEGER)  
RETURN BLOB;
```

Parameters

Table 196-8  GETCONTENTBLOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>csid</td>
<td>Character set ID of the BLOB returned</td>
</tr>
</tbody>
</table>

196.3.8 GETCONTENTCLOB Function

This function returns the contents of the resource as a CLOB.

Syntax

```
DBMS_XDBRESOURCE.GETCONTENTCLOB (  
    res   IN    XDBResource)  
RETURN CLOB;
```

Parameters

Table 196-9  GETCONTENTCLOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.9 GETCONTENTREF Function

This function returns the contents of the resource as an XMLTypeRef.

Syntax

```
DBMS_XDBRESOURCE.GETCONTENTREF (  
    res   IN    XDBResource)  
RETURN VARCHAR2;
```
196.3.10 GETCONTENTTYPE Function

Given an XDBResource, this function returns its content-type.

Syntax

```
DBMS_XDBRESOURCE.GETCONTENTTYPE (
   res   IN    XDBResource)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.11 GETCONTENTXML Function

This function returns the contents of the resource as an XMLTypeRef.

Syntax

```
DBMS_XDBRESOURCE.GETCONTENTXML (
   res   IN    XDBResource)
RETURN XMLType;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.12 GETCONTENTVARCHAR2 Function

This function returns the contents of the resource as a string.

Syntax

```
DBMS_XDBRESOURCE.GETCONTENTVARCHAR2 (
   res   IN    XDBResource)
RETURN VARCHAR2;
```
Parameters

### Table 196-13 GETCONTENTVARCHAR2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

#### 196.3.13 GETCREATIONDATE Function

Given an XDBResource, this function returns its creation date.

**Syntax**

```sql
DBMS_XDBRESOURCE.GETCREATIONDATE (
    res IN XDBResource
) RETURN TIMESTAMP;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

#### 196.3.14 GETCREATOR Function

Given an XDBResource, this function returns its creator.

**Syntax**

```sql
DBMS_XDBRESOURCE.GETCREATOR (
    res IN XDBResource
) RETURN VARCHAR2;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

#### 196.3.15 GETCUSTOMMETADATA Function

This function returns the requested custom metadata given the xpath and namespace to the metadata.

**Syntax**

```sql
DBMS_XDBRESOURCE.GETCUSTOMMETADATA (
    res IN XDBResource, 
    xpath IN VARCHAR2, 
    namespace IN VARCHAR2 
) RETURN XDBResource;
```
Parameters

Table 196-16  GETCUSTOMMETADATA Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>xpath</td>
<td>XPath for custom metadata</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

Usage Notes

The first component of the XPath expression must be "Resource".

196.3.16 GETDISPLAYNAME Function

Given an XDBResource, this function returns its display name.

Syntax

```sql
DBMS_XDBRESOURCE.GETDISPLAYNAME (
    res   IN    XDBResource)
RETURN VARCHAR2;
```

Parameters

Table 196-17  GETDISPLAYNAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.17 GETLANGUAGE Function

Given an XDBResource, this function returns its language.

Syntax

```sql
DBMS_XDBRESOURCE.GETLANGUAGE (
    res   IN    XDBResource)
RETURN VARCHAR2;
```

Parameters

Table 196-18  GETLANGUAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.18 GETLASTMODIFIER Function

Given an XDBResource, this function returns its last modifier.

Syntax

```sql
DBMS_XDBRESOURCE.GETLASTMODIFIER (
    res IN XDBResource)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.19 GETMODIFICATIONDATE Function

Given an XDBResource, this function returns its modification date.

Syntax

```sql
DBMS_XDBRESOURCE.GETMODIFICATIONDATE (
    res IN XDBResource)
RETURN TIMESTAMP;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.20 GETOWNER Function

Given an XDBResource, this function returns its owner.

Syntax

```sql
DBMS_XDBRESOURCE.GETOWNER (
    res IN XDBResource)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.21 GETREFCOUNT Function

Given an XDBResource, this function returns its reference count.

Syntax

```
DBMS_XDBRESOURCE.GETREFCOUNT (  
    res IN XDBResource)  
RETURN PLS_INTEGER;
```

Parameters

Table 196-22  GETREFCOUNT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.22 GETVERSIONID Function

Given an XDBResource, this function returns its version ID.

Syntax

```
DBMS_XDBRESOURCE.GETVERSIONID (  
    res IN XDBResource)  
RETURN PLS_INTEGER;
```

Parameters

Table 196-23  GETVERSIONID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.23 HASACLCHANGED Function

This function returns **TRUE** if the ACL of the given resource has changed, **FALSE** otherwise.

Syntax

```
DBMS_XDBRESOURCE.HASACLCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

Parameters

Table 196-24  HASACLCHANGED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.24 HASAUTHORCHANGED Function

This function returns TRUE if the author of the given resource has changed, FALSE otherwise.

Syntax

```
DBMS_XDBRESOURCE.HASAUTHORCHANGED ( res   IN    XDBResource) RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.25 HASCHANGED Function

Given an XPath, this function determines whether the element or attribute represented by the XPath has changed.

Syntax

```
DBMS_XDBRESOURCE.HASCHANGED ( res         IN    XDBResource, xpath       IN    VARCHAR2, namespace   IN    VARCHAR2) RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>xpath</td>
<td>XPath to check</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace to use</td>
</tr>
</tbody>
</table>

196.3.26 HASCHARACTERSETCHANGED Function

This function returns TRUE if the character set of the given resource has changed, FALSE otherwise.

Syntax

```
DBMS_XDBRESOURCE.HASCHARACTERSETCHANGED ( res   IN    XDBResource) RETURN BOOLEAN;
```
196.3.27 HASCOMMENTCHANGED Function

This function returns TRUE if the comment of the given resource has changed, FALSE otherwise.

Syntax

```
DBMS_XDBRESOURCE.HASCOMMENTCHANGED (
    res   IN    XDBResource)
RETURN BOOLEAN;
```

Parameters

Table 196-28  HASCOMMENTCHANGED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.28 HASCONTENTCHANGED Function

This function returns TRUE if the contents of the given resource has changed, FALSE otherwise.

Syntax

```
DBMS_XDBRESOURCE.HASCONTENTCHANGED (
    res   IN    XDBResource)
RETURN BOOLEAN;
```

Parameters

Table 196-29  HASCONTENTCHANGED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.29 HASCONTENTTYPECHANGED Function

This function returns **TRUE** if the content-type of the given resource has changed, **FALSE** otherwise.

**Syntax**

```sql
DBMS_XDBRESOURCE.HASCONTENTTYPECHANGED (
    res   IN    XDBResource)
RETURN BOOLEAN;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.30 HASCREATIONDATECHANGED Function

This function returns **TRUE** if the creation date of the given resource has changed, **FALSE** otherwise.

**Syntax**

```sql
DBMS_XDBRESOURCE.HASCREATIONDATECHANGED (
    res   IN    XDBResource)
RETURN BOOLEAN;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.31 HASCREATORCHANGED Function

This function returns **TRUE** if the creator of the given resource has changed, **FALSE** otherwise.

**Syntax**

```sql
DBMS_XDBRESOURCE.HASCREATORCHANGED (
    res   IN    XDBResource)
RETURN BOOLEAN;
```
Parameters

Table 196-32  HASCREATORCHANGED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.32 HASCUSTOMMETADATACHANGED Function

This function checks whether the custom-metadata for a given resource has changed.

Syntax

```
DBMS_XDBRESOURCE.HASCUSTOMMETADATACHANGED (    
    res IN XDBResource)    
RETURN BOOLEAN;
```

Parameters

Table 196-33  HASCUSTOMMETADATACHANGED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.33 HASDISPLAYNAMECHANGED Function

This function returns TRUE if the display name of the given resource has changed, FALSE otherwise.

Syntax

```
DBMS_XDBRESOURCE.HASDISPLAYNAMECHANGED (    
    res IN XDBResource)    
RETURN BOOLEAN;
```

Parameters

Table 196-34  HASDISPLAYNAMECHANGED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.34 HASLANGUAGECHANGED Function

This function returns TRUE if the language of the given resource has changed, FALSE otherwise.

Syntax

```sql
DBMS_XDBRESOURCE.HASLANGUAGECHANGED (  
    res   IN    XDBResource)  
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.35 HASLASTMODIFIERCHANGED Function

This function returns TRUE if the last modifier of the given resource has changed, FALSE otherwise.

Syntax

```sql
DBMS_XDBRESOURCE.HASLASTMODIFIERCHANGED (  
    res   IN    XDBResource)  
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.36 HASMODIFICATIONDATECHANGED Function

This function returns TRUE if the modification date of the given resource has changed, FALSE otherwise.

Syntax

```sql
DBMS_XDBRESOURCE.HASMODIFICATIONDATECHANGED (  
    res   IN    XDBResource)  
RETURN BOOLEAN;
```
196.3.37 HASOWNERCHANGED Function

This function returns TRUE if the owner of the given resource has changed, FALSE otherwise.

Syntax

```sql
DBMS_XDBRESOURCE.HASOWNERCHANGED (  
  res IN XDBResource)  
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.38 HASREFCOUNTCHANGED Function

This function returns TRUE if the reference count of the given resource has changed, FALSE otherwise.

Syntax

```sql
DBMS_XDBRESOURCE.HASREFCOUNTCHANGED (  
  res IN XDBResource)  
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.39 HASVERSIONIDCHANGED Function

This function returns TRUE if the version ID of the given resource has changed, FALSE otherwise.

Syntax

DBMS_XDBRESOURCE.HASVERSIONIDCHANGED (  
   res IN XDBResource)  
RETURN BOOLEAN;

Parameters

Table 196-40 HASVERSIONIDCHANGED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.40 ISFOLDER Function

This function returns TRUE if the given resource is a folder, FALSE otherwise.

Syntax

DBMS_XDBRESOURCE.ISFOLDER (  
   res IN XDBResource)  
RETURN BOOLEAN;

Parameters

Table 196-41 ISFOLDER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.41 ISNULL Function

This function returns TRUE if input resource is NULL.

Syntax

DBMS_XDBRESOURCE.ISNULL (  
   res IN XDBResource)  
RETURN BOOLEAN;
Parameters

Table 196-42   ISNULL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>Input resource</td>
</tr>
</tbody>
</table>

196.3.42 MAKEDOCUMENT Function

This function converts the XDBResource to a DOMDocument which can be operated on using the XMLDOM interface.

See Also:
The DBMS_XMLDOM package

Syntax

```sql
DBMS_XDBRESOURCE.MAKEDOCUMENT (
    res   IN    XDBResource)
RETURN DBMS_XMLDOM.DOMDocument;
```

Parameters

Table 196-43   MAKEDOCUMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>

196.3.43 SAVE Procedure

This procedure updates the resource with any modifications.

Syntax

```sql
DBMS_XDBRESOURCE.SAVE (
    res   IN    XDBResource);
```

Parameters

Table 196-44   SAVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
</tbody>
</table>
196.3.44 SETACL Procedure

This procedure sets the ACL of the given XDBResource to the path specified.

Syntax

```
DBMS_XDBRESOURCE.SETACL (
    res      IN OUT  XDBResource,
    ACLPath  IN      VARCHAR2);
```

Parameters

Table 196-45 SETACL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>ACLPath</td>
<td>Absolute path of the new ACL</td>
</tr>
</tbody>
</table>

196.3.45 SETAUTHOR Procedure

This procedure sets the author of the given XDBResource to the specified string.

Syntax

```
DBMS_XDBRESOURCE.SETAUTHOR (
    res     IN OUT  XDBResource,
    author  IN      VARCHAR2);
```

Parameters

Table 196-46 SETAUTHOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>author</td>
<td>Author</td>
</tr>
</tbody>
</table>

196.3.46 SETCHARACTERSET Procedure

This procedure sets the character set of the given XDBResource to a specified character set.

Syntax

```
DBMS_XDBRESOURCE.SETCHARACTERSET (
    res      IN OUT  XDBResource,
    charSet  IN      VARCHAR2);
```
196.3.47 SETCOMMENT Procedure

This procedure sets a comment associated with the given XDBResource.

Syntax

```sql
DBMS_XDBRESOURCE.SETCOMMENT (    
    res      IN OUT  XDBResource,    
    comment  IN      VARCHAR2);
```

196.3.48 SETCONTENT Procedures

This procedure replaces the contents of the given resource with the given datatype.

Syntax

```sql
DBMS_XDBRESOURCE.SETCONTENT (    
    res      IN OUT  XDBResource,    
    data     IN      BFILE,    
    csid     IN      NUMBER);    
DBMS_XDBRESOURCE.SETCONTENT (    
    res      IN OUT  XDBResource,    
    data     IN      BLOB,    
    csid     IN      PLS_INTEGER);    
DBMS_XDBRESOURCE.SETCONTENT (    
    res      IN OUT  XDBResource,    
    data     IN      CLOB);    
DBMS_XDBRESOURCE.SETCONTENT (    
    res      IN OUT  XDBResource,    
    data     IN      REF SYS.XMLType,    
    sticky   IN      BOOLEAN := TRUE);    
DBMS_XDBRESOURCE.SETCONTENT (    
    res      IN OUT  XDBResource,    
    data     IN      REF SYS.XMLType,);    
```
DBMS_XDBRESOURCE.SETCONTENT
res IN OUT XDBResource,
data IN SYS.XMLType);

Parameters

Table 196-49  SETCONTENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>data</td>
<td>Data input as BFILE, BLOB, CLOB, string, XMLType</td>
</tr>
<tr>
<td>csid</td>
<td>Character set ID of the BFILE, BLOB</td>
</tr>
<tr>
<td>sticky</td>
<td>If True creates a sticky REF, otherwise non-sticky</td>
</tr>
</tbody>
</table>

196.3.49 SETCONTENTTYPE Procedure

This procedure sets the content-type of the given XDBResource.

Syntax

DBMS_XDBRESOURCE.SETCONTENTTYPE (  
res IN OUT XDBResource,  
conttype IN VARCHAR2);  

Parameters

Table 196-50  SETCONTENTTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>conttype</td>
<td>New content-type</td>
</tr>
</tbody>
</table>

196.3.50 SETCUSTOMMETADATA Procedure

This procedure sets the custom metadata specified by the xpath and namespace to new data.

Syntax

DBMS_XDBRESOURCE.SETCUSTOMMETADATA (  
res IN OUT XDBResource,  
xpath IN VARCHAR2,  
namespace IN VARCHAR2,  
newMetadata IN XMLType);  

Parameters
Parameters

Table 196-51  SETCUSTOMMETADATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>xpath</td>
<td>XPath to change</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace to use</td>
</tr>
<tr>
<td>newMetadata</td>
<td>New data that should replace the metadata at the given XPath</td>
</tr>
</tbody>
</table>

Usage Notes

The first component of the XPath expression must be "Resource".

196.3.51 SETDISPLAYNAME Procedure

This procedure sets the display name of the given XDBResource.

Syntax

```sql
DBMS_XDBRESOURCE.SETDISPLAYNAME (  
    res IN OUT XDBResource,  
    name IN VARCHAR2);
```

Parameters

Table 196-52  SETDISPLAYNAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>name</td>
<td>New display name</td>
</tr>
</tbody>
</table>

196.3.52 SETLANGUAGE Procedure

This procedure sets the language of the given XDBResource.

Syntax

```sql
DBMS_XDBRESOURCE.SETLANGUAGE (  
    res IN OUT XDBResource,  
    ACLPath IN VARCHAR2);
```

Parameters

Table 196-53  SETLANGUAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>ACLPath</td>
<td>New path</td>
</tr>
</tbody>
</table>
196.3.53 SETOWNER Procedure

This procedure sets the owner of the given XDBResource.

Syntax

DBMS_XDBRESOURCE.SETOWNER (  
    res IN OUT XDBResource,  
    owner IN VARCHAR2);  

Parameters

Table 196-54  SETOWNER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>XDBResource</td>
</tr>
<tr>
<td>owner</td>
<td>New owner</td>
</tr>
</tbody>
</table>

Usage Notes

The user must have the XDBADMIN privilege to call this subprogram.
The `DBMS_XDBT` package provides a convenient mechanism for administrators to set up a `CONTEXT` index on the Oracle XML DB hierarchy.

The package contains procedures to create default preferences, create the index and set up automatic synchronization of the `CONTEXT` index.

The `DBMS_XDBT` package also contains a set of package variables that describe the configuration settings for the index. These are intended to cover the basic customizations that installations may require, but is by no means a complete set.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Summary of DBMS_XDBT Subprograms

See Also:

Oracle XML DB Developer's Guide

197.1 DBMS_XDBT Overview

The `DBMS_XDBT` package contains a set of package variables that describe the configuration settings for the index.

The `DBMS_XDBT` package can be used in the following ways:

- Customize the package to set up the appropriate configuration.
- Use the `DROPREFERENCES Procedure` to drop any existing index preferences
- Create new index preferences using the `CREATEPREFERENCES Procedure` procedure
- Create the `CONTEXT` index using the `CREATEINDEX Procedure` procedure
- Set up automatic synchronization of the index using the `CONFIGUREAUTOSYNC Procedure`

197.2 DBMS_XDBT Security Model

Owned by XDB, the `DBMS_XDBT` package must be created by SYS or XDB. The `EXECUTE` privilege is granted to SYS or XDB. Subprograms in this package are executed using the privileges of the current user.
197.3 DBMS_XDBT Operational Notes

Configuration settings, or package variables, are available to customize the DBMS_XDBT package.

The DBMS_XDBT package can be customized by using a PL/SQL procedure or an anonymous block to set the relevant package variables, configuration settings, and then execute the procedures. A more general approach would be to introduce the appropriate customizations by modifying this package in place, or as a copy. The system must be configured to use job queues, and the jobs can be viewed through the USER_JOBS catalog views.

<table>
<thead>
<tr>
<th>Table 197-1 General Indexing Settings for Customizing DBMS_XDBT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>IndexName</td>
</tr>
<tr>
<td>IndexTablespace</td>
</tr>
<tr>
<td>IndexMemory</td>
</tr>
<tr>
<td>LogFile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 197-2 Filtering Settings for Customizing DBMS_XDBT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>SkipFilter_Types</td>
</tr>
<tr>
<td>NullFilter_Types</td>
</tr>
<tr>
<td>FilterPref</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 197-3 Stoplist Settings for Customizing DBMS_XDBT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>StoplistPref</td>
</tr>
<tr>
<td>StopWords</td>
</tr>
</tbody>
</table>
Table 197-4  Sectioning and Section Group Settings for Customizing DBMS_XDBT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SectionGroup</td>
<td>HTML_SECTION_GROUP</td>
<td>Default sectioner. Use PATH_SECTION_GROUP or AUTO_SECTION_GROUP if repository contains mainly XML documents.</td>
</tr>
<tr>
<td>SectiongroupPref</td>
<td>XDB$CI_SECTIONGROUP</td>
<td>Name of the section group.</td>
</tr>
</tbody>
</table>

Table 197-5  Other Index Preference Settings for Customizing DBMS_XDBT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatastorePref</td>
<td>XDB$CI_DATASTORE</td>
<td>Name of the datastore preference</td>
</tr>
<tr>
<td>StoragePref</td>
<td>XDB$CI_STORAGE</td>
<td>Name of the storage preference</td>
</tr>
<tr>
<td>WordlistPref</td>
<td>XDB$CI_WORDLIST</td>
<td>Name of the wordlist preference</td>
</tr>
<tr>
<td>DefaultLexerPref</td>
<td>XDB$CI_DEFAULT_LEXER</td>
<td>Name of the default lexer preference.</td>
</tr>
</tbody>
</table>

Table 197-6  SYNC (CONTEXT Synchronization) Settings for Customizing DBMS_XDBT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoSyncPolicy</td>
<td>SYNC_BY_PENDING_COUNT</td>
<td>Indicates when the index should be SYNCed. One of SYNC_BY_PENDING_COUNT, SYNC_BY_TIME, or SYNC_BY_PENDING_COUNT_AND_TIME.</td>
</tr>
<tr>
<td>MaxPendingCount</td>
<td>2</td>
<td>Maximum number of documents in the CTX_USER_PENDING queue before an index SYNC is triggered. Only if the AutoSyncPolicy is SYNC_BY_PENDING_COUNT or SYNC_BY_PENDING_COUNT_AND_TIME.</td>
</tr>
<tr>
<td>CheckPendingCountIn</td>
<td>10 minutes</td>
<td>How often, in minutes, the pending queue should be checked. Only if the AutoSyncPolicy is SYNC_BY_PENDING_COUNT or SYNC_BY_PENDING_COUNT_AND_TIME.</td>
</tr>
<tr>
<td>SyncInterval</td>
<td>60 minutes</td>
<td>Indicates how often, in minutes, the index should be SYNCed. Only if the AutoSyncPolicy is SYNC_BY_TIME or SYNC_BY_PENDING_COUNT_AND_TIME.</td>
</tr>
</tbody>
</table>

197.4 Summary of DBMS_XDBT Subprograms

This table lists the DBMS_XDBT subprograms in alphabetical order and briefly describes them.

Table 197-7  DBMS_XDBT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIGUREAUTOSYNC</td>
<td>Configures the CONTEXT index for automatic maintenance, SYNC</td>
</tr>
<tr>
<td>CREATEDATASTOREPREF</td>
<td>Creates a USER datastore preference for the CONTEXT index</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
</tr>
</tbody>
</table>

Oracle
### DBMS_XDBT Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEFILTERPREF Procedure</td>
<td>Creates a filter preference for the CONTEXT index</td>
</tr>
<tr>
<td>CREATEINDEX Procedure</td>
<td>Creates the CONTEXT index on the XML DB hierarchy</td>
</tr>
<tr>
<td>CREATELEXERPREF Procedure</td>
<td>Creates a lexer preference for the CONTEXT index</td>
</tr>
<tr>
<td>CREATEPREFERENCES Procedure</td>
<td>Creates preferences required for the CONTEXT index on the XML DB hierarchy</td>
</tr>
<tr>
<td>CREATESECTIONGROUPPREF Procedure</td>
<td>Creates a storage preference for the CONTEXT index</td>
</tr>
<tr>
<td>CREATESTOPLISTPREF Procedure</td>
<td>Creates a section group for the CONTEXT index</td>
</tr>
<tr>
<td>CREATESTORAGEPREF Procedure</td>
<td>Creates a wordlist preference for the CONTEXT index</td>
</tr>
<tr>
<td>CREATEWORLDLISTPREF Procedure</td>
<td>Creates a stoplist for the CONTEXT index</td>
</tr>
<tr>
<td>DROPREFERENCES Procedure</td>
<td>Drops any existing preferences</td>
</tr>
</tbody>
</table>

### 197.4.1 CONFIGUREAUTOSYNC Procedure

This procedure sets up jobs for automatic SYNCs of the CONTEXT index.

**Syntax**

```
DBMS_XDBT.CONFIGUREAUTOSYNC;
```

**Usage Notes**

- The system must be configured for job queues for automatic synchronization. The jobs can be viewed using the `USER_JOBS` catalog views.
- The configuration parameter `AutoSyncPolicy` can be set to choose an appropriate synchronization policy.

The synchronization can be based on one of the following:

<table>
<thead>
<tr>
<th>Sync Basis</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNC_BY_PENDING_COUNT</td>
<td>The SYNC is triggered when the number of documents in the pending queue is greater than a threshold (See the <code>MaxPendingCount</code> configuration setting). The pending queue is polled at regular intervals (See the <code>CheckPendingCountInterval</code> configuration parameter) to determine if the number of documents exceeds the threshold.</td>
</tr>
<tr>
<td>SYNC_BY_TIME</td>
<td>The SYNC is triggered at regular intervals. (See the <code>SyncInterval</code> configuration parameter).</td>
</tr>
<tr>
<td>SYNC_BY_PENDING_COUNT_AND_TIME</td>
<td>A combination of both of the preceding options.</td>
</tr>
</tbody>
</table>
197.4.2 CREATEDATASTOREPREF Procedure

This procedure creates a user datastore preference for the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATEDATASTOREPREF;
```

Usage Notes

- The name of the datastore preference can be modified; see the DatastorePref configuration setting.
- The default USER datastore procedure also filters the incoming document. The DBMS_XDBT package provides a set of configuration settings that control the filtering process.
- The SkipFilter_Types array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not indexed. Some of the properties of the document metadata, such as author, remain unindexed.
  - The NullFilter_Types array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not filtered; however, they are still indexed. This is intended to be used for documents that are text-based, such as HTML, XML and plain-text.
  - All other documents use the INSO filter through the IFILTER API.

197.4.3 CREATEFILTERPREF Procedure

This procedure creates a NULL filter preference for the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATEFILTERPREF;
```

Usage Notes

- The name of the filter preference can be modified; see FilterPref configuration setting.
- The USER datastore procedure filters the incoming document; see CREATEDATASTOREPREF Procedure for more details.

197.4.4 CREATEINDEX Procedure

This procedure creates the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATEINDEX;
```

Usage Notes

- The name of the index can be changed; see the IndexName configuration setting.
- Set the LogFile configuration parameter to enable ROWID logging during index creation.
• Set the IndexMemory configuration parameter to determine the amount of memory that index creation, and later SYNCs, will use.

### 197.4.5 CREATELEXERPREF Procedure

This procedure creates a BASIC lexer preference for the CONTEXT index on the XML DB hierarchy.

**Syntax**

```sql
DBMS_XDBT.CREATELEXERPREF;
```

**Usage Notes**

- The name of the lexer preference can be modified; see LexerPref configuration setting. No other configuration settings are provided.
- MultiLexer preferences are not supported.
- Base letter translation is turned on by default.

### 197.4.6 CREATEPREFERENCES Procedure

This procedure creates a set of default preferences based on the configuration settings.

**Syntax**

```sql
DBMS_XDBT.CREATEPREFERENCES;
```

### 197.4.7 CREATESECTIONGROUPPREF Procedure

This procedure creates a section group for the CONTEXT index on the XML DB hierarchy.

**Syntax**

```sql
DBMS_XDBT.CREATESECTIONGROUPPREF;
```

**Usage Notes**

- The name of the section group can be changed; see the SectiongroupPref configuration setting.
- The HTML sectioner is used by default. No zone sections are created by default. If the vast majority of documents are XML, consider using the AUTO_SECTION_GROUP or the PATH_SECTION_GROUP; see the SectionGroup configuration setting.

### 197.4.8 CREATESTOPLISTPREF Procedure

This procedure creates a stoplist for the CONTEXT index on the XML DB hierarchy.

**Syntax**

```sql
DBMS_XDBT.CREATESTOPLISTPREF;
```
Usage Notes

- The name of the stoplist can be modified; see the StoplistPref configuration setting.
- Numbers are not indexed.
- The StopWords array is a configurable list of stopwords. These are meant to be stopwords in addition to the set of stopwords in CTXSYS.DEFAULT_STOPLIST.

197.4.9 CREATESTORAGEPREF Procedure

This procedure creates a BASIC_STORAGE preference for the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATESTORAGEPREF;
```

Usage Notes

- The name of the storage preference can be modified; see the StoragePref configuration setting.
- A tablespace can be specified for the tables and indexes comprising the CONTEXT index; see the IndexTablespace configuration setting.
- Prefix and Substring indexing are not turned on by default.
- The I_INDEX_CLAUSE uses key compression.

197.4.10 CREATEWORLDLISTPREF Procedure

This procedure creates a word list preference for the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.CREATENWORLDLISTPREF;
```

Usage Notes

- The name of the word list preference can be modified; see the WordlistPref configuration setting. No other configuration settings are provided.
- FUZZY_MATCH and STEMMER attributes are set to AUTO (auto-language detection)

197.4.11 DROPPREFERENCES Procedure

This procedure drops any previously created preferences for the CONTEXT index on the XML DB hierarchy.

Syntax

```
DBMS_XDBT.DROPPREFERENCES;
```
The DBMS_XDBZ package controls the Oracle XML DB repository security, which is based on Access Control Lists (ACLs).

This chapter contains the following topics:

- Security Model
- Constants
- Summary of DBMS_XDBZ Subprograms

See Also:
Oracle XML DB Developer's Guide

198.1 DBMS_XDBZ Security Model

Owned by XDB, the DBMS_XDBZ package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

198.2 DBMS_XDBZ Constants

The DBMS_XDBZ package defines several constants that can be used for specifying parameter values.

The package uses the constants shown in following tables.

| Table 198-1  DBMS_XDBZ Constants - Name Format |
|-----------------|-----------------|-----------------|-----------------|
| Constant         | Type            | Value | Description                        |
| NAME_FORMAT_SHORT | PLS_INTEGER     | 1     | DB user name or LDAP nickname       |
| NAME_FORMAT_DISTINGUISHED | PLS_INTEGER     | 2     | LDAP distinguished name |

| Table 198-2  DBMS_XDBZ Constants - Enable Option |
|-----------------|-----------------|-----------------|-----------------|
| Constant         | Type            | Value | Description                                                                 |
| ENABLE_CONTENTS  | PLS_INTEGER     | 1     | Enables hierarchy for contents and is used by users when calling the ENABLE_HIERARCHY Procedure |
Table 198-2  (Cont.) DBMS_XDBZ Constants - Enable Option

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_RESMETADATA</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>Enables hierarchy for resource metadata, that is, this table will store schema based custom metadata for resources</td>
</tr>
</tbody>
</table>

Table 198-3  DBMS_XDBZ Constants - Enable Option Exercised

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_ENABLED_CONTENTS</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>If hierarchy was enabled for contents, that is, the ENABLE_HIERARCHY Procedure was called with hierarchy_type as ENABLE_CONTENTS</td>
</tr>
<tr>
<td>IS_ENABLED_RESMETADATA</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>If hierarchy was enabled for resource metadata, that is, the ENABLE_HIERARCHY Procedure was called with hierarchy_type as ENABLE_RESMETADATA</td>
</tr>
</tbody>
</table>

198.3 Summary of DBMS_XDBZ Subprograms

This table lists the DBMS_XDBZ subprograms and briefly describes them.

Table 198-4  DBMS_XDBZ Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE_HIERARCHY Procedure</td>
<td>Disables repository support for the specified XMLTYPE table or view</td>
</tr>
<tr>
<td>ENABLE_HIERARCHY Procedure</td>
<td>Enables repository support for the specified XMLType table or view</td>
</tr>
<tr>
<td>GET_ACLOID Function</td>
<td>Retrieves the ACL Object ID for the specified resource</td>
</tr>
<tr>
<td>GET_USERID Function</td>
<td>Retrieves the user ID for the specified user</td>
</tr>
<tr>
<td>IS_HIERARCHY_ENABLED Function</td>
<td>Determines if repository support for the specified XMLType table or view is enabled</td>
</tr>
<tr>
<td>PURGELDAPCACHE Function</td>
<td>Purges the LDAP nickname cache</td>
</tr>
</tbody>
</table>

198.3.1 CREATENONCEKEY Procedure

This procedure generates a nonce value for use in digest authentication.

Syntax

DBMS_XDBZ.CREATENONCEKEY;
198.3.2 DISABLE_HIERARCHY Procedure

This procedure disables repository support for a particular XMLType table or view.

Syntax

DBMS_XDBZ.DISABLE_HIERARCHY(
    object_schema IN VARCHAR2,
    object_name   IN VARCHAR2);

Parameters

Table 198-5  DISABLE_HIERARCHY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema name of the XMLType table or view</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the XMLType table or view</td>
</tr>
</tbody>
</table>

198.3.3 ENABLE_HIERARCHY Procedure

This procedure enables repository support for a particular XMLType table or view. This allows the use of a uniform ACL-based security model across all documents in the repository.

See Also:

Oracle XML DB Developer's Guide for more information about

Syntax

DBMS_XDBZ.ENABLE_HIERARCHY(
    object_schema   IN   VARCHAR2,
    object_name     IN   VARCHAR2,
    hierarchy_type  IN   PLS_INTEGER := DBMS_XDBZ.ENABLE_CONTENTS);

Parameters

Table 198-6  ENABLE_HIERARCHY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema name of the XMLType table or view</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the XMLType table or view</td>
</tr>
</tbody>
</table>
**Table 198-6**  (Cont.) ENABLE_HIERARCHY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hierarchy_type</td>
<td>How to enable the hierarchy.</td>
</tr>
<tr>
<td></td>
<td>- ENABLE CONTENTS - enable hierarchy for contents, that is, this table will store contents of resources in the repository</td>
</tr>
<tr>
<td></td>
<td>- ENABLE RESMETA DATA - enable hierarchy for resource metadata, that is, this table will store schema based custom metadata for resources</td>
</tr>
</tbody>
</table>

If this subprogram is called on a table, another call will have no effect. Note that you cannot enable hierarchy for both contents and resource metadata.

### 198.3.4 GET_ACLOID Function

This function retrieves the ACL Object ID for the specified resource, if the repository path is known.

**Syntax**

```sql
DBMS_XDBZ.GET_ACLOID(
    aclpath   IN   VARCHAR2,
    acloid    OUT  RAW)
RETURN BOOLEAN;
```

**Parameters**

**Table 198-7**  GET_ACLOID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclpath</td>
<td>ACL resource path for the repository</td>
</tr>
<tr>
<td>acloid</td>
<td>Returned Object ID</td>
</tr>
</tbody>
</table>

**Return Values**

Returns **TRUE** if successful.

### 198.3.5 GET_USERID Function

This function retrieves the user ID for the specified user name. The local database is searched first, and if found, the **USERID** is returned in 4-byte database format. Otherwise, the LDAP directory is searched, if available, and if found, the **USERID** is returned in 4-byte database format.

**Syntax**

```sql
DBMS_XDBZ.GET_USERID(
    username IN  VARCHAR2,
    userid   OUT RAW,
    format   IN  BINARY_INTEGER := NAME_FORMAT_SHORT)
RETURN BOOLEAN;
```
Parameters

Table 198-8  GET_USERID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the database or LDAP user.</td>
</tr>
<tr>
<td>userid</td>
<td>Return parameter for the matching user id.</td>
</tr>
<tr>
<td>format</td>
<td>Format of the specified user name; valid options are:</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDBZ.NAME_FORMAT_SHORT (default) -- DB user name or LDAP nickname</td>
</tr>
<tr>
<td></td>
<td>· DBMS_XDBZ.NAME_FORMAT_DISTINGUISHED -- LDAP distinguished name.</td>
</tr>
</tbody>
</table>

Return Values

Returns TRUE if successful.

198.3.6 IS_HIERARCHY_ENABLED Function

This function determines if repository support for the specified XMLType table or view is enabled.

Syntax

DBMS_XDBZ.IS_HIERARCHY_ENABLED(
  object_schema    IN  VARCHAR2,
  object_name      IN  VARCHAR2,
  hierarchy_type   IN  PLS_INTEGER := IS_ENABLED_CONTENTS)
RETURN BOOLEAN;

Parameters

Table 198-9  IS_HIERARCHY_ENABLED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_schema</td>
<td>Schema name of the XMLType table or view</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the XMLType table or view</td>
</tr>
<tr>
<td>hierarchy_type</td>
<td>Type of hierarchy to check for:</td>
</tr>
<tr>
<td></td>
<td>· IS_ENABLED_CONTENTS - if hierarchy was enabled for contents,</td>
</tr>
<tr>
<td></td>
<td>that is, the ENABLE_HIERARCHY Procedure was called with hierarchy_type as ENABLE_CONTENTS</td>
</tr>
<tr>
<td></td>
<td>· IS_ENABLED_RESMETADATA - if hierarchy was enabled for resource metadata, that is, the ENABLE_HIERARCHY Procedure was called with hierarchy_type as ENABLE_RESMETADATA</td>
</tr>
</tbody>
</table>

Return Values

Returns TRUE if the given XMLTYPE table or view has the XDB Hierarchy enabled with the specified type.
198.3.7 PURGELDAPCACHE Function

This function purges the LDAP nickname cache.

It returns TRUE if successful.

Syntax

DBMS_XDBZ.PURGELDAPCACHE
RETURN BOOLEAN;
The DBMS_XEVENT package provides event-related types and supporting subprograms.

This chapter contains the following topics:

- Security Model
- Constants
- Subprogram Groups
- Summary of DBMS_XEVENT Subprograms

See Also:
Oracle XML DB Developer's Guide for more information about "Oracle XML DB Repository Events"

### 199.1 DBMS_XEVENT Security Model

Owned by XDB, the DBMS_XEVENT package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

### 199.2 DBMS_XEVENT Constants

The DBMS_XEVENT package defines several constants that can be used for specifying parameter values.

The constants are defined in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENDER_EVENT</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PRE_CREATE_EVENT</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>POST_CREATE_EVENT</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>PRE_DELETE_EVENT</td>
<td>PLS_INTEGER</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>POST_DELETEEVENT</td>
<td>PLS_INTEGER</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>PRE_UPDATE_EVENT</td>
<td>PLS_INTEGER</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>POST_UPDATE_EVENT</td>
<td>PLS_INTEGER</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>PRE_LOCK_EVENT</td>
<td>PLS_INTEGER</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Table 199-1  (Cont.) DBMS_XEVENT Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST_LOCK_EVENT</td>
<td>PLS_INTEGER</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>PRE_UNLOCK_EVENT</td>
<td>PLS_INTEGER</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>POST_UNLOCK_EVENT</td>
<td>PLS_INTEGER</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>PRE_LINKIN_EVENT</td>
<td>PLS_INTEGER</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>POST_LINKIN_EVENT</td>
<td>PLS_INTEGER</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>PRE_LINKTO_EVENT</td>
<td>PLS_INTEGER</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>POST_LINKTO_EVENT</td>
<td>PLS_INTEGER</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PRE_UNLINKIN_EVENT</td>
<td>PLS_INTEGER</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>POST_UNLINKIN_EVENT</td>
<td>PLS_INTEGER</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>PRE_UNLINKFROM_EVENT</td>
<td>PLS_INTEGER</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>POST_UNLINKFROM_EVENT</td>
<td>PLS_INTEGER</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>PRE_CHECKIN_EVENT</td>
<td>PLS_INTEGER</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>POST_CHECKIN_EVENT</td>
<td>PLS_INTEGER</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>PRE_CHECKOUT_EVENT</td>
<td>PLS_INTEGER</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>POST_CHECKOUT_EVENT</td>
<td>PLS_INTEGER</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>PRE_UNCHECKOUT_EVENT</td>
<td>PLS_INTEGER</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>POST_UNCHECKOUT_EVENT</td>
<td>PLS_INTEGER</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>PRE_VERSIONCONTROL_EVENT</td>
<td>PLS_INTEGER</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>POST_VERSIONCONTROL_EVENT</td>
<td>PLS_INTEGER</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>PRE_OPEN_EVENT</td>
<td>PLS_INTEGER</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>POST_OPEN_EVENT</td>
<td>PLS_INTEGER</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>PRE_INCONSISTENT_UPDATE_EVENT</td>
<td>PLS_INTEGER</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>POST_INCONSISTENT_UPDATE_EVENT</td>
<td>PLS_INTEGER</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>POST_CHECKIN_EVENT</td>
<td>PLS_INTEGER</td>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>

199.3 Subprogram Groups

The subprograms in the DBMS_XEVENT package can be divided in groups.

- XDBEvent Type Subprograms
- XDBRepositoryEvent Type Subprograms
- XDBHandlerList Type Subprograms
- XDBHandler Type Subprograms
- XDBPath Type Subprograms
- XDBLink Type Subprograms
199.3.1 DBMS_XEVENT XDBEvent Type Subprograms

This subprogram group provides an interface for use with the XDBEvent type.

Table 199-2  XDBEvent Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETCURRENTUSER Function</td>
<td>Returns the name of the user executing the operation that triggers the event</td>
</tr>
<tr>
<td>GETEVENT Function</td>
<td>Returns a value identifying the triggering event</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Returns TRUE if input argument is NULL</td>
</tr>
</tbody>
</table>

The Summary of DBMS_XEVENT Subprograms contains a complete listing of all subprograms in the package.

199.3.2 DBMS_XEVENT XDBRepositoryEvent Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBRepositoryEvent type.

Table 199-3  XDBRepositoryEvent Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETAPPLICATIONDATA Function</td>
<td>Returns the <code>&lt;applicationData&gt;</code> element extracted from the resource configuration that defines the invoking handler</td>
</tr>
<tr>
<td>GETHANDLERLIST Function</td>
<td>Returns an XDBHandlerList object containing the list of handlers that will be executed after the currently executing handler</td>
</tr>
<tr>
<td>GETINTERFACE Function</td>
<td>Returns the top-level interface used to initiate the operation that triggered the event</td>
</tr>
<tr>
<td>GETLINK Function</td>
<td>Returns an XDBLink object for the target resource</td>
</tr>
<tr>
<td>GETLOCK Function</td>
<td>Returns the lock object corresponding to the current operation</td>
</tr>
<tr>
<td>GETOLDRESOURCE Function</td>
<td>Returns the original XDBResource object before the operation was executed</td>
</tr>
<tr>
<td>GETOPENACCESSMODE Function</td>
<td>Returns the access mode for the open operation</td>
</tr>
<tr>
<td>GETOPENDENYMODE Function</td>
<td>Returns the deny mode for the open operation</td>
</tr>
<tr>
<td>GETOUTPUTSTREAM Function</td>
<td>Returns the output BLOB in which the handler can write the rendered data</td>
</tr>
<tr>
<td>GETPARAMETER Function</td>
<td>Returns the value of a request or session-specific parameter</td>
</tr>
<tr>
<td>GETPARENT Function</td>
<td>Returns the resource object corresponding to a parent folder of the target resource</td>
</tr>
<tr>
<td>GETPATH Function</td>
<td>Returns the XDBPath object representing the path of the resource for which the event was fired</td>
</tr>
</tbody>
</table>
Table 199-3 (Cont.) XDBRepositoryEvent Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETRESOURCE Function</td>
<td>Returns an XDBResource object that provides methods to access and modify the contents and metadata of the target resource</td>
</tr>
<tr>
<td>GETUPDATEBYTECOUNT Function</td>
<td>If the current operation is a byte-range write, returns the byte count</td>
</tr>
<tr>
<td>GETUPDATEBYTEOFFSET Function</td>
<td>If the current operation is a byte-range write, function returns the byte offset at which the range begins</td>
</tr>
<tr>
<td>GETXDBEVENT Function</td>
<td>Converts an XDBRepositoryEvent object to an XDBEvent type</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Returns TRUE if input argument is NULL</td>
</tr>
<tr>
<td>SETRENDERPATH Procedure</td>
<td>Specifies the path of the resource that contains the rendered contents</td>
</tr>
<tr>
<td>SETRENDERSTREAM Procedure</td>
<td>sets the BLOB from which the rendered contents can be read</td>
</tr>
</tbody>
</table>

The Summary of DBMS_XEVENT Subprograms contains a complete listing of all subprograms in the package.

199.3.3 DBMS_XEVENT XDBHandlerList Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBHandlerList type.

Table 199-4 XDBHandlerList Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR Procedure</td>
<td>Clears the handler list</td>
</tr>
<tr>
<td>GETFIRST Function</td>
<td>Returns the first handler in the list</td>
</tr>
<tr>
<td>GETNAME Function</td>
<td>Returns the next handler in the list</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Returns TRUE if input argument is NULL</td>
</tr>
<tr>
<td>REMOVE Procedure</td>
<td>Removes the specified handler from the handler list</td>
</tr>
</tbody>
</table>

The Summary of DBMS_XEVENT Subprograms contains a complete listing of all subprograms in the package.

199.3.4 DBMS_XEVENT XDBHandler Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBHandler type.

Table 199-5 XDBHandler Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETLANGUAGE Function</td>
<td>Returns the implementation language of the handler</td>
</tr>
</tbody>
</table>
Table 199-5  (Cont.) XDBHandler Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETSCHEMA Function</td>
<td>Returns the schema of the handler's source</td>
</tr>
<tr>
<td>GETSOURCE Function</td>
<td>Returns the name of the Java class, PL/SQL package or object type implementing the handler</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Returns TRUE if input argument is NULL</td>
</tr>
</tbody>
</table>

The Summary of DBMS_XEVENT Subprograms contains a complete listing of all subprograms in the package.

199.3.5 DBMS_XEVENT XDBPath Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBPath type.

Table 199-6  XDBPath Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETNAME Function</td>
<td>Returns the string representation of the path</td>
</tr>
<tr>
<td>GETPARENTPATH Function</td>
<td>Returns the parent's path</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Returns TRUE if input argument is NULL</td>
</tr>
</tbody>
</table>

The Summary of DBMS_XEVENT Subprograms contains a complete listing of all subprograms in the package.

199.3.6 DBMS_XEVENT XDBLink Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBLink type.

Table 199-7  XDBLink Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETCHILDOID Function</td>
<td>Returns the OID of the resource to which the link is pointing</td>
</tr>
<tr>
<td>GETPARENTNAME Function</td>
<td>Returns the link's parent folder's name</td>
</tr>
<tr>
<td>GETPARENTOID Function</td>
<td>Returns the link's parent folder's OID</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Returns TRUE if input argument is NULL</td>
</tr>
</tbody>
</table>

The Summary of DBMS_XEVENT Subprograms contains a complete listing of all subprograms in the package.

199.4 Summary of DBMS_XEVENT Subprograms

This table lists the DBMS_XEVENT subprograms and briefly describes them.
<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR Procedure</td>
<td>Clears the handler list</td>
<td>XDBHandlerList Type Subprograms</td>
</tr>
<tr>
<td>GETAPPLICATIONDATA Function</td>
<td>Returns the <code>&lt;applicationData&gt;</code> element extracted from the resource configuration that defines the invoking handler</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETCHILDROID Function</td>
<td>Returns the OID of the resource to which the link is pointing</td>
<td>XDBLink Type Subprograms</td>
</tr>
<tr>
<td>GETCURRENTUSER Function</td>
<td>Returns the name of the user executing the operation that triggers the event</td>
<td>XDBEvent Type Subprograms</td>
</tr>
<tr>
<td>GETEVENT Function</td>
<td>Returns a value identifying the triggering event</td>
<td>XDBEvent Type Subprograms</td>
</tr>
<tr>
<td>GETFIRST Function</td>
<td>Returns the first handler in the list</td>
<td>XDBHandlerList Type Subprograms</td>
</tr>
<tr>
<td>GETHANDLERLIST Function</td>
<td>Returns an XDBHandlerList object containing the list of handlers that will be executed after the currently executing handler</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETINTERFACE Function</td>
<td>Returns the top-level interface used to initiate the operation that triggered the event</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETLANGUAGE Function</td>
<td>Returns the implementation language of the handler</td>
<td>XDBHandler Type Subprograms</td>
</tr>
<tr>
<td>GETLINK Function</td>
<td>Returns an XDBLink object for the target resource</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETLOCK Function</td>
<td>Returns the lock object corresponding to the current operation</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETNAME Function</td>
<td>Returns the string representation of the path</td>
<td>XDBPath Type Subprograms</td>
</tr>
<tr>
<td>GETNAME Function</td>
<td>Returns the next handler in the list</td>
<td>XDBHandlerList Type Subprograms</td>
</tr>
<tr>
<td>GETOLDRESOURCE Function</td>
<td>Returns the original XDBResource object before the operation was executed</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETOPENACCESSMODE Function</td>
<td>Returns the access mode for the open operation</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETOPENDENYMODE Function</td>
<td>Returns the deny mode for the open operation</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETOUTPUTSTREAM Function</td>
<td>Returns the output BLOB in which the handler can write the rendered data</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETPARAMETER Function</td>
<td>Returns the value of a request or session-specific parameter</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETPARENT Function</td>
<td>Returns the resource object corresponding to a parent folder of the target resource</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>GETPARENTNAME Function</td>
<td>Returns the link's parent folder's name</td>
<td>XDBLink Type Subprograms</td>
</tr>
<tr>
<td>GETPARENTOID Function</td>
<td>Returns the link's parent folder's OID</td>
<td>XDBLink Type Subprograms</td>
</tr>
<tr>
<td>GETPARENTNAME Function</td>
<td>Returns the parent's path</td>
<td>XDBPath Type Subprograms</td>
</tr>
<tr>
<td>GETPATH Function</td>
<td>Returns the XDBPath object representing the path of the resource for which the event was fired</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETRESOURCE Function</td>
<td>Returns an XDBResource object that provides methods to access and modify the contents and metadata of the target resource</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETSCHEMA Function</td>
<td>Returns the schema of the handler's source</td>
<td>XDBHandler Type Subprograms</td>
</tr>
<tr>
<td>GETSOURCE Function</td>
<td>Returns the name of the Java class, PL/SQL package or object type implementing the handler</td>
<td>XDBHandler Type Subprograms</td>
</tr>
<tr>
<td>GETUPDATEBYTECOUNT Function</td>
<td>If the current operation is a byte-range write, returns the byte count</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETUPDATEBYTEOFFSET Function</td>
<td>If the current operation is a byte-range write, function returns the byte offset at which the range begins</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>GETXDBEVENT Function</td>
<td>Converts an XDBRepositoryEvent object to an XDBEvent type</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Returns TRUE if input argument is NULL</td>
<td>XDBEvent Type Subprograms</td>
</tr>
<tr>
<td>REMOVE Procedure</td>
<td>Removes the specified handler from the handler list</td>
<td>XDBHandlerList Type Subprograms</td>
</tr>
<tr>
<td>SETRENDERPATH Procedure</td>
<td>Specifies the path of the resource that contains the rendered contents</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
<tr>
<td>SETRENDERSTREAM Procedure</td>
<td>sets the BLOB from which the rendered contents can be read</td>
<td>XDBRepositoryEvent Type Subprograms</td>
</tr>
</tbody>
</table>
199.4.1 CLEAR Procedure

This procedure clears the handler list.

See Also:
XDBHandlerList Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.CLEAR (hl IN OUT XDBHandlerList);
```

Parameters

Table 199-9 CLEAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hl</td>
<td>Handler list</td>
</tr>
</tbody>
</table>

199.4.2 GETAPPLICATIONDATA Function

This function returns the `<applicationData>` element extracted from the resource configuration that defines the invoking handler.

See Also:
XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETAPPLICATIONDATA (ev IN XDBRepositoryEvent RETURN XMLType;
```

Parameters

Table 199-10 GETAPPLICATIONDATA Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>
199.4.3 GETCHILDOID Function

This function returns the OID of the resource to which the link is pointing.

[See Also:](#)

[XDBLink Type Subprograms](#) for other subprograms in this group

**Syntax**

```sql
DBMS_XEVENT.GETCHILDOID (  
    Link IN XDBLink)  
RETURN RAW;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Link</td>
</tr>
</tbody>
</table>

199.4.4 GETCURRENTUSER Function

This function returns the name of the user executing the operation that triggers the event.

[See Also:](#)

[XDBEvent Type Subprograms](#) for other subprograms in this group

**Syntax**

```sql
DBMS_XEVENT.GETCURRENTUSER (  
    ev IN XDBEvent)  
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBEvent type</td>
</tr>
</tbody>
</table>
199.4.5 GETEVENT Function

This function returns the name of the user executing the operation that triggers the event.

See Also:
XDBEvent Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETEVENT (ev IN XDBEvent)
RETURN XDBEventID;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBEvent type</td>
</tr>
</tbody>
</table>

199.4.6 GETFIRST Function

This function returns the first handler in the list.

See Also:
XDBHandlerList Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETFIRST (hl IN XDBHandlerList)
RETURN XDBHandler;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hl</td>
<td>Handler list</td>
</tr>
</tbody>
</table>
199.4.7 GETHANDLERLIST Function

This function returns an XDBHandlerList object containing the list of handlers that will be executed after the currently executing handler.

The current handler can then filter out some of the subsequent handlers if necessary, subject to security checks. An insufficient privilege exception is raised if the executing user does not have the required access privilege to any of the resource configuration associating with a handler in the list.

See Also:
XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETHANDLERLIST (
   ev   IN   XDBRepositoryEvent)
RETURN XDBHandlerList;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

199.4.8 GETINTERFACE Function

This function returns the top-level interface used to initiate the operation that triggered the event. This could be HTTP, FTP or SQL.

See Also:
XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETINTERFACE (
   ev   IN   XDBRepositoryEvent)
RETURN VARCHAR2;
```
Parameters

Table 199-16  GETINTERFACE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

199.4.9 GETLANGUAGE Function

This function returns the implementation language of the handler.

See Also:

XDBHandler Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETLANGUAGE (
    handler   IN   XDBHandler)
RETURN VARCHAR2;
```

Parameters

Table 199-17  GETLANGUAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>Handler</td>
</tr>
</tbody>
</table>

199.4.10 GETLINK Function

This function returns an XDBLink object for the target resource.

For a link* or unlink* event, this will be the link involved in the operation. For other events, an error is returned. Using this object the handler can access link properties, such as, ParentName, ParentOID, ChildOID and LinkName.

See Also:

XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETLINK (
    ev   IN   XDBRepositoryEvent)
RETURN XDBLink;
```
199.4.11 GETLINKNAME Function

This function returns the name of the link.

<table>
<thead>
<tr>
<th>See Also:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDBLink Type Subprograms for other subprograms in this group</td>
</tr>
</tbody>
</table>

**Syntax**

```sql
DBMS_XEVENT.GETLINKNAME (  
    link IN XDBLink)  
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Link</td>
</tr>
</tbody>
</table>

199.4.12 GETLOCK Function

This function returns the lock object corresponding to the current operation. It is only valid for lock and unlock events.

<table>
<thead>
<tr>
<th>See Also:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDBRepositoryEvent Type Subprograms for other subprograms in this group</td>
</tr>
</tbody>
</table>

**Syntax**

```sql
DBMS_XEVENT.GETLOCK (  
    ev IN XDBRepositoryEvent)  
RETURN XDBLock;
```
199.4.13 GETLANGUAGE Function

This function returns the implementation language of the handler.

**See Also:**
XDBHandler Type Subprograms for other subprograms in this group

**Syntax**

```sql
DBMS_XEVENT.GETLANGUAGE (
    handler   IN   XDBHandler
) RETURN VARCHAR2;
```

199.4.14 GETNAME Function

This function returns the string representation of the path.

**See Also:**
XDBPath Type Subprograms for other subprograms in this group

**Syntax**

```sql
DBMS_XEVENT.GETNAME (
    path   IN   XDBPath
) RETURN VARCHAR2;
```
Parameters

Table 199-22  GETNAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path</td>
</tr>
</tbody>
</table>

199.4.15 GETNEXT Function

This function returns the next handler in the list.

See Also:

XDBHandlerList Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.GETNEXT (
   hl   IN   XDBHandlerList)
RETURN XDBHandler;

Parameters

Table 199-23  GETNEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hl</td>
<td>Handler list</td>
</tr>
</tbody>
</table>

199.4.16 GETOLDRESOURCE Function

This function returns the original XDBResource object before the operation was executed.

This method applies only to update event. For other events, an error is returned. This is a read-only object, and consequently none of the modifier methods will work on this object.

See Also:

XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.GETOLDRESOURCE (
   ev   IN   XDBRepositoryEvent)
RETURN XDBResource;
Parameters

Table 199-24  GETOLDRESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

199.4.17 GETOPENACCESSMODE Function

This function returns the access mode for the open operation.

See Also:
XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

```plsql
DBMS_XEVENT.GETOPENACCESSMODE (
    ev   IN   XDBRepositoryEvent)
RETURN PLS_INTEGER;
```

Parameters

Table 199-25  GETOPENACCESSMODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

Return Values

- XDBRepositoryEvent.OPEN_ACCESS_READ (value 1)
- XDBRepositoryEvent.OPEN_ACCESS_WRITE (value 2)
- XDBRepositoryEvent.OPEN_ACCESS_READ_WRITE (value 3)

199.4.18 GETOPENDENYMODE Function

This function returns the deny mode for the open operation. It is only valid for the open event.

See Also:
XDBRepositoryEvent Type Subprograms for other subprograms in this group
Syntax

DBMS_XEVENT.GETOPENDENYMODE (ev IN XDBRepositoryEvent) RETURN PLS_INTEGER;

Parameters

Table 199-26  GETOPENDENYMODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

Return Values

- XDBRepositoryEvent.OPEN_DENY_NONE (value 0)
- XDBRepositoryEvent.OPEN_DENY_READ (value 1)
- XDBRepositoryEvent.OPEN_DENY_READ_WRITE (value 2)

199.4.19 GETOUTPUTSTREAM Function

This function returns the output BLOB in which the handler can write the rendered data. It is only valid for the render event.

See Also:

XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.GETOUTPUTSTREAM (ev IN XDBRepositoryEvent) RETURN BLOB;

Parameters

Table 199-27  GETOUTPUTSTREAM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>
199.4.20 GETPARAMETER Function

This function returns the value of a request or session-specific parameter. The definition of the `key` parameter can be found in RFC 2616 (HTTP/1.1). They will be mapped to equivalent SQL session parameters (if any).

See Also:

- `XDBRepositoryEvent Type Subprograms` for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETPARAMETER (  
   ev   IN   XDBRepositoryEvent,  
   key  IN   VARCHAR2)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
<tr>
<td>key</td>
<td>Supported parameters:</td>
</tr>
<tr>
<td></td>
<td>- ACCEPT</td>
</tr>
<tr>
<td></td>
<td>- ACCEPT-LANGUAGE</td>
</tr>
<tr>
<td></td>
<td>- ACCEPT-CHARSET</td>
</tr>
<tr>
<td></td>
<td>- ACCEPT_ENCODING</td>
</tr>
</tbody>
</table>

199.4.21 GETPARENT Function

This function returns the resource object corresponding to a parent folder of the target resource. Note that this could be any folder that contains a link to the target resource. This is a read-only object, and consequently none of the modifier methods will work on this object. For a link* or unlink* event, this method returns the link's parent folder.

See Also:

- `XDBRepositoryEvent Type Subprograms` for other subprograms in this group

Syntax

```sql
DBMS_XEVENT.GETPARENT (  
   ev   IN   XDBRepositoryEvent)  
RETURN XDBResource;
```
Parameters

Table 199-29 GETPARENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

199.4.22 GETPARENTNAME Function

This function returns the link's parent folder's name.

See Also:

XDBLink Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.GETPARENTNAME (
    link IN XDBLink)
RETURN VARCHAR2;

Parameters

Table 199-30 GETPARENTNAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Link</td>
</tr>
</tbody>
</table>

199.4.23 GETPARENTOID Function

This function returns the link's parent folder's OID.

See Also:

XDBLink Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.GETPARENTOID (
    link IN XDBLink)
RETURN RAW;
199.4.24 GETPARENTPATH Function

This function returns the parent's path. The level indicates the number of levels up the hierarchy. This value must be greater than zero. Level 1 means the immediate parent. If level exceeds the height of the tree then a NULL is returned.

See Also:

XDBPath Type Subprograms for other subprograms in this group

Syntax

```
DBMS_XEVENT.GETPARENTPATH (
    path   IN   XDBPath,
    level   IN   INTEGER)
RETURN XDBPath;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path</td>
</tr>
<tr>
<td>level</td>
<td>Number of levels up the hierarchy</td>
</tr>
</tbody>
</table>

199.4.25 GETPATH Function

This function returns the XDBPath object representing the path of the resource for which the event was fired. From this object, functions are provided to get the different path segments.

See Also:

XDBRepositoryEvent Type Subprograms for other subprograms in this group
Syntax

DBMS_XEVENT.GETPATH (
    ev   IN   XDBRepositoryEvent)
RETURN XDBPath;

Parameters

Table 199-33  GETPATH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

199.4.26 GETRESOURCE Function

This function returns an XDBResource object that provides methods to access and modify the contents and metadata of the target resource. This object reflects any changes made by previous handlers to the resource.

The modifier methods will work only in the pre-create and pre-update event handlers. For a link* or unlink* event, this method returns the resource that the link is pointing to. For a create event, this method returns the resource that is being created.

See Also:
XDBRepositoryEvent Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.GETRESOURCE (
    ev   IN   XDBRepositoryEvent)
RETURN XDBResource;

Parameters

Table 199-34  GETRESOURCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

199.4.27 GETSCHEMA Function

This function returns the schema of the handler's source.

See Also:
XDBHandler Type Subprograms for other subprograms in this group
Syntax

\[
\text{DBMS\_XEVENT\_GETSCHEMA (}
\quad \text{handler IN XDBHandler)}
\text{RETURN VARCHAR2;}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>Handler</td>
</tr>
</tbody>
</table>

199.4.28 GETSOURCE Function

This function returns the name of the Java class, PL/SQL package or object type implementing the handler.

See Also:

- [XDBHandler Type Subprograms](#) for other subprograms in this group

Syntax

\[
\text{DBMS\_XEVENT\_GETSOURCE (}
\quad \text{handler IN XDBHandler)}
\text{RETURN VARCHAR2;}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>Handler</td>
</tr>
</tbody>
</table>

199.4.29 GETUPDATEBYTECOUNT Function

If the current operation is a byte-range write, the GETUPDATEBYTECOUNT function returns the byte count. It is only valid for the inconsistent-update event.

See Also:

- [XDBRepositoryEvent Type Subprograms](#) for other subprograms in this group
Syntax

```
DBMS_XEVENT.GETUPDATEBYTECOUNT (  
      ev   IN   XDBRepositoryEvent)  
RETURN NUMBER;
```

Parameters

**Table 199-37  GETUPDATEBYTECOUNT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ev</code></td>
<td>Event of <code>XDBRepositoryEvent</code> type</td>
</tr>
</tbody>
</table>

**199.4.30 GETUPDATEBYTEOFFSET Function**

If the current operation is a byte-range write, the `GETUPDATEBYTEOFFSET` function returns the byte offset at which the range begins. It is only valid for the inconsistent-update event.

**See Also:**

*XDBRepositoryEvent Type Subprograms* for other subprograms in this group

Syntax

```
DBMS_XEVENT.GETUPDATEBYTEOFFSET (  
      ev   IN   XDBRepositoryEvent)  
RETURN NUMBER;
```

Parameters

**Table 199-38  GETUPDATEBYTEOFFSET Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ev</code></td>
<td>Event of <code>XDBRepositoryEvent</code> type</td>
</tr>
</tbody>
</table>

**199.4.31 GETXDBEVEVENT Function**

This function converts an `XDBRepositoryEvent` object to an `XDBEvent` type.

**See Also:**

*XDBRepositoryEvent Type Subprograms* for other subprograms in this group
Syntax

DBMS_XEVENT.GETXDBEVENT (
    ev   IN   XDBRepositoryEvent)
RETURN XDBEvent;

Parameters

Table 199-39  GETXDBEVENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of XDBRepositoryEvent type</td>
</tr>
</tbody>
</table>

199.4.32 ISNULL Functions

This function returns TRUE if input argument is NULL.

See Also:

- XDBEvent Type Subprograms for other subprograms in this group
- XDBRepositoryEvent Type Subprograms for other subprograms in this group
- XDBHandlerList Type Subprograms for other subprograms in this group
- XDBHandler Type Subprograms for other subprograms in this group
- XDBPath Type Subprograms for other subprograms in this group
- XDBLink Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.ISNULL (
    ev   IN   XDBEvent)
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
    ev   IN   XDBRepositoryEvent)
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
    hl   IN   XDBHandlerList)
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
    handler   IN   XDBHandler)
RETURN BOOLEAN;
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
    path   IN   XDBPath)
RETURN BOOLEAN;
DBMS_XEVENT.ISNULL (
    link       IN   XDBLink)
RETURN BOOLEAN;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>Event of specified type</td>
</tr>
<tr>
<td>hl</td>
<td>Handler list</td>
</tr>
<tr>
<td>handler</td>
<td>Handler</td>
</tr>
<tr>
<td>path</td>
<td>Path</td>
</tr>
</tbody>
</table>

199.4.33 REMOVE Procedure

This procedure removes the specified handler from the handler list.

See Also:
XDBHandlerList Type Subprograms for other subprograms in this group

Syntax

DBMS_XEVENT.REMOVE (
    hl       IN OUT   XDBHandlerList,
    handler  IN       XDBHandler);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hl</td>
<td>Handler list</td>
</tr>
<tr>
<td>handler</td>
<td>Handler</td>
</tr>
</tbody>
</table>

199.4.34 SETRENDERPATH Procedure

This procedure specifies the path of the resource that contains the rendered contents.

This should not be called after the stream returned by GETOUTPUTSTREAM Function is written to or after the SETRENDERSTREAM Procedure is called; doing so will result in an error. This is only valid for the render event.

Syntax

DBMS_XEVENT.SETRENDERPATH (
    ev         IN   XDBRepositoryEvent,  
    path       IN   VARCHAR2);
Parameters

Table 199-42  SETRENDERPATH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>XDB Repository Event object</td>
</tr>
<tr>
<td>path</td>
<td>Path of the resource containing the rendered contents</td>
</tr>
</tbody>
</table>

199.4.35 SETRENDERSTREAM Procedure

This procedure sets the BLOB from which the rendered contents can be read.

This should not be called after the stream returned by GETOUTPUTSTREAM is written to or after SETRENDERPATH is called; doing so will result in an error. This is only valid for the render event.

Syntax

```
DBMS_XEVENT.SETRENDERSTREAM (  
    ev    IN   XDBRepositoryEvent,  
    istr  IN   BLOB);  
```

Parameters

Table 199-43  SETRENDERSTREAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev</td>
<td>XDBRepositoryEvent object</td>
</tr>
<tr>
<td>istr</td>
<td>Input stream from which to get the rendered contents</td>
</tr>
</tbody>
</table>
The `DBMS_XMLDOM` package is used to access `XMLType` objects, and implements the Document Object Model (DOM), an application programming interface for HTML and XML documents.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Types
- Exceptions
- Subprogram Groups
- Summary of `DBMS_XMLDOM` Subprograms

See Also:

*Oracle XML Developer's Kit Programmer's Guide*

### 200.1 DBMS_XMLDOM Overview

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents, and the manner in which they are accessed and manipulated.

In the DOM specification, the term "document" is used in the broad sense. XML is being increasingly used to represent many different kinds of information that may be stored in diverse systems. This information has been traditionally be seen as "data"; nevertheless, XML presents this data as documents, and the `DBMS_XMLDOM` package allows you access to both schema-based and non schema-based documents.

**Note:**

Read-from and write-to files must be on the server file system.

With DOM, anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions. In particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.
One important objective of the W3C DOM specification is to provide a standard programming interface that can be used in a wide variety of environments, programming languages, and applications. Because the DOM standard is object-oriented while PL/SQL is essentially a procedural language, some changes had to be made:

- Various DOM interfaces such as Node, Element, and others have equivalent PL/SQL types DOMNode, DOMElement, respectively.
- Various DOMException codes such as WRONG_DOCUMENT_ERR, HIERARCHY_REQUEST_ERR, and others, have similarly named PL/SQL exceptions.
- Various DOM Node type codes such as ELEMENT_NODE, ATTRIBUTE_NODE, and others, have similarly named PL/SQL constants.
- Subprograms defined on a DOM type become functions or procedures that accept it as a parameter. For example, to perform APPENDCHILD Function on a DOMNode, the APPENDCHILD FunctionPL/SQL function is provided.
- To perform setAttribute on a DOMElement elem, setATTRIBUTE Procedures, use PL/SQL procedure.

DOM defines an inheritance hierarchy. For example, Document, Element, and Attr are defined to be subtypes of Node (see Figure 200-1). Thus, a method defined in the Node interface should be available in these as well. Since such inheritance is not supported in PL/SQL, it is implemented through direct invocation of the MAKENODE function. Calling MAKENODE on various DOM types converts these types into a DOMNode. The appropriate functions or procedures that accept DOMNodes can then be called to operate on these types. If, subsequently, type specific functionality is desired, the DOMNode can be converted back into the original type by the makeXXX functions, where DOMXXX is the desired DOM type.

**Figure 200-1**  Inheritance Diagram for DOM Types

The implementation of this interface follows the REC-DOM-Level-1-19981001.
200.2 DBMS_XMLDOM Security Model

Owned by XDB, the DBMS_XMLDOM package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC.

Subprograms in this package are executed using the privileges of the current user.

200.3 DBMS_XMLDOM Constants

The DBMS_XMLDOM package defines several constants that can be used for specifying parameter values.

These constants are listed in the following table.

<table>
<thead>
<tr>
<th>Table 200-1 Defined Constants for DBMS_XMLDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
</tr>
<tr>
<td>---------------------------------</td>
</tr>
<tr>
<td>ELEMENT_NODE</td>
</tr>
<tr>
<td>ATTRIBUTE_NODE</td>
</tr>
<tr>
<td>TEXT_NODE</td>
</tr>
<tr>
<td>CDATA_SECTION_NODE</td>
</tr>
<tr>
<td>ENTITY_REFERENCE_NODE</td>
</tr>
<tr>
<td>ENTITY_NODE</td>
</tr>
<tr>
<td>PROCESSING_INSTRUCTION_NODE</td>
</tr>
<tr>
<td>COMMENT_NODE</td>
</tr>
<tr>
<td>DOCUMENT_NODE</td>
</tr>
<tr>
<td>DOCUMENT_TYPE_NODE</td>
</tr>
<tr>
<td>DOCUMENT_FRAGMENT_NODE</td>
</tr>
<tr>
<td>NOTATION_NODE</td>
</tr>
</tbody>
</table>

200.4 DBMS_XMLDOM Types

This table lists and briefly describes the types for the DBMS_XMLDOM.DOMTYPE package.

<table>
<thead>
<tr>
<th>Table 200-2 XDB_XMLDOM Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>DOMATTR</td>
</tr>
<tr>
<td>DOMCDATASECTION</td>
</tr>
<tr>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>DOMCOMMENT</td>
</tr>
</tbody>
</table>
### Table 200-2  (Cont.) XDB.XMLDOM Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMDOCUMENT</td>
<td>Implements the DOM Document interface.</td>
</tr>
<tr>
<td>DOMDOCUMENTFRAGMENT</td>
<td>Implements the DOM DocumentFragment interface.</td>
</tr>
<tr>
<td>DOMDOCUMENTTYPE</td>
<td>Implements the DOM Document Type interface.</td>
</tr>
<tr>
<td>DOMELEMENT</td>
<td>Implements the DOM Element interface.</td>
</tr>
<tr>
<td>DOMENTITY</td>
<td>Implements the DOM Entity interface.</td>
</tr>
<tr>
<td>DOMENTITYREFERENCE</td>
<td>Implements the DOM EntityReference interface.</td>
</tr>
<tr>
<td>DOMIMPLEMENTATION</td>
<td>Implements the DOM Implementation interface.</td>
</tr>
<tr>
<td>DOMNAMEDNODEMAP</td>
<td>Implements the DOM Named Node Map interface.</td>
</tr>
<tr>
<td>DOMNODE</td>
<td>Implements the DOM Node interface.</td>
</tr>
<tr>
<td>DOMNODELIST</td>
<td>Implements the DOM NodeList interface.</td>
</tr>
<tr>
<td>DOMNOTATION</td>
<td>Implements the DOM Notation interface.</td>
</tr>
<tr>
<td>DOMPROCESSINGINSTRUCTION</td>
<td>Implements the DOM Processing instruction interface.</td>
</tr>
<tr>
<td>DOMTEXT</td>
<td>Implements the DOM Text interface.</td>
</tr>
</tbody>
</table>

### 200.5 DBMS.XMLDOM Exceptions

`DBMS.XMLDOM` generates an exception when it encounters an issue.

This table lists the exceptions defined for `DBMS(XMLDOM)`:

#### Table 200-3  Exceptions for DBMS.XMLDOM

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMSTRING_SIZE_ERR</td>
<td>If the specified range of text does not fit into a <code>DOMString</code>.</td>
</tr>
<tr>
<td>HIERARCHY_REQUEST_ERR</td>
<td>If any node is inserted somewhere it doesn't belong.</td>
</tr>
<tr>
<td>INDEX_SIZE_ERR</td>
<td>If index or size is negative, or greater than the allowed value.</td>
</tr>
<tr>
<td>INUSE_ATTRIBUTE_ERR</td>
<td>If an attempt is made to add an attribute that is already in use elsewhere.</td>
</tr>
<tr>
<td>INVALID_CHARACTER_ERR</td>
<td>If an invalid or illegal character is specified, such as in a name. See production 2 in the XML specification for the definition of a legal character, and production 5 for the definition of a legal name character.</td>
</tr>
<tr>
<td>NO_DATA_ALLOWED_ERROR</td>
<td>If data is specified for a node that does not support data.</td>
</tr>
<tr>
<td>NOT_FOUND_ERR</td>
<td>If an attempt is made to reference a node in a context where it does not exist.</td>
</tr>
<tr>
<td>NO_MODIFICATION_ALLOWED_ERR</td>
<td>If an attempt is made to modify an object where modifications are not allowed.</td>
</tr>
<tr>
<td>NOT_SUPPORTED_ERR</td>
<td>If the implementation does not support the requested type of object or operation.</td>
</tr>
<tr>
<td>WRONG_DOCUMENT_ERR</td>
<td>If a node is used in a different document than the one that created it (that doesn't support it).</td>
</tr>
</tbody>
</table>
200.6 DBMS_XMLDOM Subprogram Groups

DBMS_XMLDOM subprograms are divided into groups according to W3C Interfaces.

- DOMNode Subprograms
- DOMAttr Subprograms
- DOMCDataSection Subprograms
- DOMCharacterData Subprograms
- DOMComment Subprograms
- DOMDocument Subprograms
- DOMDocumentFragment Subprograms
- DOMDocumentType Subprograms
- DOMElememt Subprograms
- DOMEntity Subprograms
- DOMEntityReference Subprograms
- DOMImplementation Subprograms
- DOMNamedNodeMap Subprograms
- DOMNodeList Subprograms
- DOMNotation Subprograms
- DOMProcessingInstruction Subprograms
- DOMText Subprograms

200.6.1 DBMS_XMLDOM DOMNode Subprograms

This table lists and briefly describes the DOMNode subprograms of DBMS_XMLDOM.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADOPTNODE Function</td>
<td>Adopts a node from another document</td>
</tr>
<tr>
<td>APPENDCHILD Function</td>
<td>Appends a new child to the node</td>
</tr>
<tr>
<td>CLONENODE Function</td>
<td>Clones the node</td>
</tr>
<tr>
<td>FREENODE Procedure</td>
<td>Frees all resources associated with the node</td>
</tr>
<tr>
<td>GETATTRIBUTES Function</td>
<td>Retrieves the attributes of the node</td>
</tr>
<tr>
<td>GETCHILDNODES Function</td>
<td>Retrieves the children of the node</td>
</tr>
<tr>
<td>GETEXPANDEDNAME Procedure and Functions</td>
<td>Retrieves the expanded name of the node</td>
</tr>
<tr>
<td>GETFIRSTCHILD Function</td>
<td>Retrieves the first child of the node</td>
</tr>
<tr>
<td>GETLASTCHILD Function</td>
<td>Retrieves the last child of the node</td>
</tr>
<tr>
<td>GETLOCALNAME Procedure and Functions</td>
<td>Retrieves the local part of the qualified name</td>
</tr>
<tr>
<td>GETNAMESPACE Procedure and Functions</td>
<td>Retrieves the node's namespace URI</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>GETNEXTSIBLING Function</td>
<td>Retrieves the next sibling of the node</td>
</tr>
<tr>
<td>GETNODENAME Function</td>
<td>Retrieves the Name of the Node</td>
</tr>
<tr>
<td>GETNODETYPE Function</td>
<td>Retrieves the Type of the node</td>
</tr>
<tr>
<td>GETNODEVALUE Function</td>
<td>Retrieves the Value of the Node</td>
</tr>
<tr>
<td>GETNODEVALUEASBINARYSTREAM Function &amp; Procedure</td>
<td>Retrieves Node Value as binary stream</td>
</tr>
<tr>
<td>GETNODEVALUEASCHARACTERSTREAM Function &amp; Procedure</td>
<td>Retrieves Node Value as character stream</td>
</tr>
<tr>
<td>GETOWNERDOCUMENT Function</td>
<td>Retrieves the owner document of the node</td>
</tr>
<tr>
<td>GETPARENTNODE Function</td>
<td>Retrieves the parent of this node</td>
</tr>
<tr>
<td>GETPREFIX Function</td>
<td>Retrieves the namespace prefix</td>
</tr>
<tr>
<td>GETPREVIOUSSIBLING Function</td>
<td>Retrieves the previous sibling of the node</td>
</tr>
<tr>
<td>GETSCHEMANODE Function</td>
<td>Retrieves the associated schema URI</td>
</tr>
<tr>
<td>HASATTRIBUTES Function</td>
<td>Tests if the node has attributes</td>
</tr>
<tr>
<td>HASCHILDNODES Function</td>
<td>Tests if the node has child nodes</td>
</tr>
<tr>
<td>IMPORTNODE Function</td>
<td>Imports a node from another document</td>
</tr>
<tr>
<td>INSERTBEFORE Function</td>
<td>Inserts a child before the reference child</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the node is NULL</td>
</tr>
<tr>
<td>MAKEATTR Function</td>
<td>Casts the node to an Attribute</td>
</tr>
<tr>
<td>MAKECDATASECTION Function</td>
<td>Casts the node to a CData Section</td>
</tr>
<tr>
<td>MAKECHARACTERDATA Function</td>
<td>Casts the node to Character Data</td>
</tr>
<tr>
<td>MAKECOMMENT Function</td>
<td>Casts the node to a Comment</td>
</tr>
<tr>
<td>MADEDOCUMENT Function</td>
<td>Casts the node to a DOM Document</td>
</tr>
<tr>
<td>MADEDOCUMENTFRAGMENT Function</td>
<td>Casts the node to a DOM Document Fragment</td>
</tr>
<tr>
<td>MADEDOCUMENTTYPE Function</td>
<td>Casts the node to a DOM Document Type</td>
</tr>
<tr>
<td>MAKEELEMENT Function</td>
<td>Casts the node to a DOM Element</td>
</tr>
<tr>
<td>MAKEENTITY Function</td>
<td>Casts the node to a DOM Entity</td>
</tr>
<tr>
<td>MAKEENTITYREFERENCE Function</td>
<td>Casts the node to a DOM Entity Reference</td>
</tr>
<tr>
<td>MAKENOTATION Function</td>
<td>Casts the node to a DOM Notation</td>
</tr>
<tr>
<td>MAKEPROCESSINGINSTRUCTION Function</td>
<td>Casts the node to a DOM Processing Instruction</td>
</tr>
<tr>
<td>Maketext Function</td>
<td>Casts the node to a DOM Text</td>
</tr>
<tr>
<td>REMOVECHILD Function</td>
<td>Removes a specified child from a node</td>
</tr>
<tr>
<td>REPLACECHILD Function</td>
<td>Replaces the old child with a new child</td>
</tr>
<tr>
<td>SETNODEVALUE Procedure</td>
<td>Sets the Value of the node</td>
</tr>
<tr>
<td>SETNODEVALUEASBINARYSTREAM Function &amp; Procedure</td>
<td>Sets the Node Value as binary stream</td>
</tr>
<tr>
<td>SETNODEVALUEASCHARACTERSTREAM Function &amp; Procedure</td>
<td>Sets the Node Value as a character stream</td>
</tr>
<tr>
<td>SETPREFIX Procedure</td>
<td>Sets the namespace prefix</td>
</tr>
</tbody>
</table>
Table 200-4  (Cont.) Summary of DOMNode Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USEBINARYSTREAM Function</td>
<td>Establishes that the stream is valid</td>
</tr>
<tr>
<td>WRITETOBUFFER Procedures</td>
<td>Writes the contents of the node to a buffer</td>
</tr>
<tr>
<td>WRITETOCLOB Procedures</td>
<td>Writes the contents of the node to a CLOB</td>
</tr>
<tr>
<td>WRITETOFILE Procedures</td>
<td>Writes the contents of the node to a file</td>
</tr>
</tbody>
</table>

200.6.2 DBMS_XMLDOM DOMAttr Subprograms

This table lists the DOMAttr subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.

Table 200-5  Summary of DOMAttr Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETEXPANDEDNAME Procedure and Functions</td>
<td>Retrieves the expanded name of the attribute</td>
</tr>
<tr>
<td>GETLOCALNAME Procedure and Functions</td>
<td>Retrieves the local name of the attribute</td>
</tr>
<tr>
<td>GETNAME Functions</td>
<td>Retrieves the name of the attribute</td>
</tr>
<tr>
<td>GETNAMESPACE Procedure and Functions</td>
<td>Retrieves the NS URI of the attribute</td>
</tr>
<tr>
<td>GETOWNERELEMENT Function</td>
<td>Retrieves the Element node, parent of the attribute</td>
</tr>
<tr>
<td>GETQUALIFIEDNAME Functions</td>
<td>Retrieves the Qualified Name of the attribute</td>
</tr>
<tr>
<td>GETSPECIFIED Function</td>
<td>Tests if attribute was specified in the element</td>
</tr>
<tr>
<td>GETVALUE Function</td>
<td>Retrieves the value of the attribute</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the Attribute node is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the Attribute to a node</td>
</tr>
<tr>
<td>SETVALUE Procedure</td>
<td>Sets the value of the attribute</td>
</tr>
</tbody>
</table>

200.6.3 DBMS_XMLDOM DOMCDataSection Subprograms

This table lists the DOMCdata subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.

Table 200-6  Summary of DOMCdata Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the CDataSection is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the CDataSection to a node</td>
</tr>
</tbody>
</table>
200.6.4 DBMS_XMLDOM DOMCharacterData Subprograms

This table lists the DOMCharacterData subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDDATA Procedure</td>
<td>Appends the specified data to the node data</td>
</tr>
<tr>
<td>DELETEDATA Procedure</td>
<td>Deletes the data from the specified offsets</td>
</tr>
<tr>
<td>GETDATA Functions</td>
<td>Retrieves the data of the node</td>
</tr>
<tr>
<td>GETLENGTH Functions</td>
<td>Retrieves the length of the data</td>
</tr>
<tr>
<td>INSERTDATA Procedure</td>
<td>Inserts the data in the node at the specified offsets</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the CharacterData is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the CharacterData to a node</td>
</tr>
<tr>
<td>REPLACEDATA Procedure</td>
<td>Changes a range of characters in the node</td>
</tr>
<tr>
<td>SETDATA Procedures</td>
<td>Sets the data to the node</td>
</tr>
<tr>
<td>SUBSTRINGDATA Function</td>
<td>Retrieves the substring of the data</td>
</tr>
</tbody>
</table>

200.6.5 DBMS_XMLDOM DOMComment Subprograms

The table lists the DOMComment subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the comment is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the Comment to a node</td>
</tr>
</tbody>
</table>

200.6.6 DBMS_XMLDOM DOMDocument Subprograms

This table lists the DOMDocument subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEATTRIBUTE Functions</td>
<td>Creates an Attribute</td>
</tr>
<tr>
<td>CREATECDATASECTION Function</td>
<td>Creates a CDATASECTION node</td>
</tr>
<tr>
<td>CREATECOMMENT Function</td>
<td>Creates a Comment node</td>
</tr>
<tr>
<td>CREATEDOCUMENT Function</td>
<td>Creates a new Document</td>
</tr>
<tr>
<td>CREATEDOCUMENTFRAGMENT Function</td>
<td>Creates a new Document Fragment</td>
</tr>
<tr>
<td>CREATEELEMENT Functions</td>
<td>Creates a new Element</td>
</tr>
</tbody>
</table>
### Table 200-9 (Cont.) Summary of DOMDocument Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEENTITYREFERENCE Function</td>
<td>Creates an Entity reference</td>
</tr>
<tr>
<td>CREATEPROCESSINGINSTRUCTION Function</td>
<td>Creates a Processing Instruction</td>
</tr>
<tr>
<td>CREATETEXTNODE Function</td>
<td>Creates a Text node</td>
</tr>
<tr>
<td>FREEDOCFRAG Procedure</td>
<td>Frees the document fragment</td>
</tr>
<tr>
<td>FREEDOCUMENT Procedure</td>
<td>Frees the document</td>
</tr>
<tr>
<td>GETCHARSET Function</td>
<td>Retrieves the character set of the DOM document</td>
</tr>
<tr>
<td>GETDOCTYPE Function</td>
<td>Retrieves the DTD of the document</td>
</tr>
<tr>
<td>GETDOCUMENTELEMENT Function</td>
<td>Retrieves the root element of the document</td>
</tr>
<tr>
<td>GETELEMENTSBYTAGNAME Functions</td>
<td>Retrieves</td>
</tr>
<tr>
<td>GETIMPLEMENTATION Function</td>
<td>Retrieves the DOM implementation</td>
</tr>
<tr>
<td>GETSTANDALONE Function</td>
<td>Retrieves the standalone property of the document</td>
</tr>
<tr>
<td>GETVERSION Function</td>
<td>Retrieves the version of the document</td>
</tr>
<tr>
<td>GETXMLTYPE Function</td>
<td>Retrieves the XMLType associated with the DOM Document</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the document is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the document to a node</td>
</tr>
<tr>
<td>NEWDOMDOCUMENT Functions</td>
<td>Creates a new document</td>
</tr>
<tr>
<td>SETCHARSET Procedure</td>
<td>Sets the character set of the DOM document</td>
</tr>
<tr>
<td>SETDOCTYPE Procedure</td>
<td>Sets the DTD of the document</td>
</tr>
<tr>
<td>SETSTANDALONE Procedure</td>
<td>Sets the standalone property of the document</td>
</tr>
<tr>
<td>SETVERSION Procedure</td>
<td>Sets the version of the document</td>
</tr>
<tr>
<td>WRITETOBUFFER Procedures</td>
<td>Writes the document to a buffer</td>
</tr>
<tr>
<td>WRITETOFILE Procedures</td>
<td>Writes the document to a file</td>
</tr>
<tr>
<td>WRITETOCLOB Procedures</td>
<td>Writes the document to a CLOB</td>
</tr>
</tbody>
</table>

### 200.6.7 DBMS_XMLDOM DOMDocumentFragment Subprograms

This table lists the DOMDocumentFragment subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.
Table 200-10  Summary of DOMDocumentFragment Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREEDOCFRAG Procedure</td>
<td>Frees the specified document fragment</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the DocumentFragment is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the Document Fragment to a node</td>
</tr>
<tr>
<td>WRISETBUFFER Procedures</td>
<td>Writes the contents of a document fragment into a buffer</td>
</tr>
</tbody>
</table>

200.6.8 DBMS_XMLDOM DOMDocumentType Subprograms

This table lists the DOMDocumentType subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.

Table 200-11  Summary of DOMDocumentType Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FINDENTITY Function</td>
<td>Finds the specified entity in the document type</td>
</tr>
<tr>
<td>FINDNOTATION Function</td>
<td>Finds the specified notation in the document type</td>
</tr>
<tr>
<td>GETENTITIES Function</td>
<td>Retrieves the nodemap of entities in the Document type</td>
</tr>
<tr>
<td>GETNAME Functions</td>
<td>Retrieves the name of the Document type</td>
</tr>
<tr>
<td>GETNOTATIONS Function</td>
<td>Retrieves the nodemap of the notations in the Document type</td>
</tr>
<tr>
<td>GETPUBLICID Functions</td>
<td>Retrieves the public ID of the document type</td>
</tr>
<tr>
<td>GETSYSTEMID Functions</td>
<td>Retrieves the system ID of the document type</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the Document Type is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the document type to a node</td>
</tr>
</tbody>
</table>

200.6.9 DBMS_XMLDOM DOMElement Subprograms

This table lists the DOMElement subprograms of DBMS_XMLDOM in alphabetical order and briefly describes them.

Table 200-12  Summary of DOMElement Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREEELEMENT Procedure</td>
<td>Frees memory allocated to a DOMElement handle</td>
</tr>
<tr>
<td>GETATTRIBUTE Functions</td>
<td>Retrieves the attribute node by name</td>
</tr>
<tr>
<td>GETATTRIBUTENODE Functions</td>
<td>Retrieves the attribute node by name</td>
</tr>
<tr>
<td>GETCHILDRENBYTAGNAME Functions</td>
<td>Retrieves children of the element by tag name</td>
</tr>
</tbody>
</table>
### Table 200-12  (Cont.) Summary of DOMElement Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| GETELEMENTSBYTAGNAME Functions        | Retrieves  
  - the elements in the DOMNODELIST by tag name  
  - elements in the subtree of a DOMNODELIST by tagname                                                                                                   |
| GETEXPANDEDNAME Procedure and Functions | Retrieves the expanded name of the element                                                                                           |
| GETLOCALNAME Procedure and Functions   | Retrieves the local name of the element                                                                                              |
| GETNAMESPACE Procedure and Functions   | Retrieves the NS URI of the element                                                                                                    |
| GETQUALIFIEDNAME Functions            | Retrieves the qualified name of the element                                                                                           |
| GETTAGNAME Function                   | Retrieves the Tag name of the element                                                                                                 |
| HASATTRIBUTE Functions                | Tests if an attribute exists                                                                                                           |
| ISNULL Functions                      | Tests if the Element is NULL                                                                                                           |
| MAKENODE Functions                    | Casts the Element to a node                                                                                                           |
| NORMALIZE Procedure                   | Normalizes the text children of the element                                                                                            |
| REMOVEATTRIBUTE Procedures            | Removes the attribute specified by the name                                                                                                |
| REMOVEATTRIBUTENODE Function          | Removes the attribute node in the element                                                                                            |
| RESOLVENAMESPACEPREFIX Function       | Resolve the prefix to a namespace URI                                                                                                 |
| SETATTRIBUTE Procedures               | Sets the attribute specified by the name                                                                                              |
| SETATTRIBUTENODE Functions            | Sets the attribute node in the element                                                                                               |

### 200.6.10 DBMS_XMLDOM DOMEntity Subprograms

This table lists and briefly describes the DOMEntity subprograms of DBMS_XMLDOM.

### Table 200-13  Summary of DOMEntity Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETNOTATIONNAME Function</td>
<td>Retrieves the notation name of the entity</td>
</tr>
<tr>
<td>GETPUBLICID Functions</td>
<td>Retrieves the public Id of the entity</td>
</tr>
<tr>
<td>GETSYSTEMID Functions</td>
<td>Retrieves the system Id of the entity</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the Entity is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the Entity to a node</td>
</tr>
</tbody>
</table>

### 200.6.11 DBMS_XMLDOM DOMEntityReference Subprograms

This table lists and briefly describes the DOMEntityReference subprograms of DBMS_XMLDOM.
Table 200-14  Summary of DOMEntityReference Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the DOMEntityReference is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the DOMEntityReference to NULL</td>
</tr>
</tbody>
</table>

200.6.12 DBMS_XMLDOM DOMImplementation Subprograms

This table lists and briefly describes the DOMImplementation subprograms of DBMS_XMLDOM.

Table 200-15  Summary of DOMImplementation Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the DOMImplementation node is NULL</td>
</tr>
<tr>
<td>HASFEATURE</td>
<td>Tests if the DOMImplementation implements a feature</td>
</tr>
</tbody>
</table>

200.6.13 DBMS_XMLDOM DOMNamedNodeMap Subprograms

This table lists and briefly describes the DOMNamedNodeMap subprograms of DBMS_XMLDOM.

Table 200-16  Summary of DOMNamedNodeMap Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETLENGTH Functions</td>
<td>Retrieves the number of items in the map</td>
</tr>
<tr>
<td>GETNAMEDITEM Function</td>
<td>Retrieves the item specified by the name</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the NamedNodeMap is NULL</td>
</tr>
<tr>
<td>ITEM Functions</td>
<td>Retrieves the item given the index in the map</td>
</tr>
<tr>
<td>REMOVENAMEDITEM Function</td>
<td>Removes the item specified by name</td>
</tr>
<tr>
<td>SETNAMEDITEM Function</td>
<td>Sets the item in the map specified by the name</td>
</tr>
</tbody>
</table>

200.6.14 DBMS_XMLDOMDOMNodeList Subprograms

This table lists and briefly describes theDOMNodeList subprograms of DBMS_XMLDOM.

Table 200-17  Summary ofDOMNodeList Subprograms; DBMS_XMLDOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREENODELIST Procedure</td>
<td>Frees all resources associated with a nodelist</td>
</tr>
<tr>
<td>GETLENGTH Functions</td>
<td>Retrieves the number of items in the list</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the NodeList is NULL</td>
</tr>
<tr>
<td>ITEM Functions</td>
<td>Retrieves the item given the index in the list</td>
</tr>
</tbody>
</table>
200.6.15 DBMS_XMLDOM DOMNotation Subprograms

This table lists and briefly describes the DOMNotation subprograms of DBMS_XMLDOM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPUBLICID Functions</td>
<td>Retrieves the public Id of the notation</td>
</tr>
<tr>
<td>GETSYSTEMID Functions</td>
<td>Retrieves the system Id of the notation</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the Notation is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the notation to a node</td>
</tr>
</tbody>
</table>

200.6.16 DBMS_XMLDOM DOMProcessingInstruction Subprograms

This table lists and briefly describes the DOMProcessingInstruction subprograms of DBMS_XMLDOM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETDATA Functions</td>
<td>Retrieves the data of the processing instruction</td>
</tr>
<tr>
<td>GETTARGET Function</td>
<td>Retrieves the target of the processing instruction</td>
</tr>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the Processing Instruction is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the Processing Instruction to a node</td>
</tr>
<tr>
<td>SETDATA Procedures</td>
<td>Sets the data of the processing instruction</td>
</tr>
</tbody>
</table>

200.6.17 DBMS_XMLDOM DOMText Subprograms

This table lists and briefly describes the DOMText subprograms of DBMS_XMLDOM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISNULL Functions</td>
<td>Tests if the text is NULL</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts the text to a node</td>
</tr>
<tr>
<td>SPLITTEXT Function</td>
<td>Splits the contents of the text node into 2 text nodes</td>
</tr>
</tbody>
</table>

200.7 Summary of DBMS_XMLDOM Subprograms

This table lists the DBMS_XMLDOM subprograms and briefly describes them.
<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADOPTNODE Function</td>
<td>Adopts a node from another document</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>APPENDCHILD Function</td>
<td>Appends a new child to the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>APPENDDATA Procedure</td>
<td>Appends the specified data to the node data</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td>CLONENODE Function</td>
<td>Clones the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>CREATEATTRIBUTE Functions</td>
<td>Creates an Attribute</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATECDATASECTION Function</td>
<td>Creates a CDataSection node</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATECOMMENT Function</td>
<td>Creates a Comment node</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATEDOCUMENT Function</td>
<td>Creates a new Document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATEDOCUMENTFRAGMENT Function</td>
<td>Creates a new Document Fragment</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATEELEMENT Functions</td>
<td>Creates a new Element</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATEENTITYREFERENCE Function</td>
<td>Creates an Entity reference</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATEPROCESSINGINSTRUCTION Function</td>
<td>Creates a Processing Instruction</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>CREATETEXTNODE Function</td>
<td>Creates a Text node</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>DELETEDATA Procedure</td>
<td>Deletes the data from the specified offsets</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td>FINDENTITY Function</td>
<td>Finds the specified entity in the document type</td>
<td>DOMDocumentType Subprograms</td>
</tr>
<tr>
<td>FINDNOTATION Function</td>
<td>Finds the specified notation in the document type</td>
<td>DOMDocumentType Subprograms</td>
</tr>
<tr>
<td>FREEDOCFRAG Procedure</td>
<td>Frees the document fragment</td>
<td>DOMDocument Subprograms and DOMDocumentFragment Subprograms</td>
</tr>
<tr>
<td>FREEDOCUMENT Procedure</td>
<td>Frees the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>FREEELEMENT Procedure</td>
<td>Frees memory allocated to a DOMElement handle</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td>FREENODE Procedure</td>
<td>Frees all resources associated with the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>FREENODELIST Procedure</td>
<td>Frees all resources associated with a nodelist</td>
<td>DOMNodeList Subprograms</td>
</tr>
<tr>
<td>GETATTRIBUTE Functions</td>
<td>Retrieves the attribute node by name</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td>GETATTRIBUTENODE Functions</td>
<td>Retrieves the attribute node by name</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>GETATTRIBUTES Function</td>
<td>Retrieves the attributes of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETCHARSET Function</td>
<td>Retrieves the character set of the DOM document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>GETCHILDNODES Function</td>
<td>Retrieves the children of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETCHILDRENBYTAGNAME Functions</td>
<td>Retrieves children of the element by tag name</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td>GETDATA Functions</td>
<td>Retrieves  · the data of the node  · the data of the processing instruction</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td>GETDOCTYPE Function</td>
<td>Retrieves the DTD of the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>GETDOCUMENTELEMENT Function</td>
<td>Retrieves the root element of the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>GETELEMENTSBYTAGNAME Functions</td>
<td>Retrieves  · the elements in the DOMNODELIST by tag name  · elements in the subtree of a DOMNODELIST by tagname</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>GETENTITIES Function</td>
<td>Retrieves the nodemap of entities in the Document type</td>
<td>DOMDocumentType Subprograms</td>
</tr>
<tr>
<td>GETEXPANDEDNAME Procedure and Functions</td>
<td>Retrieves  · the expanded name of the node  · the expanded name of the attribute  · the expanded name of the element</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETFIRSTCHILD Function</td>
<td>Retrieves the first child of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETIMPLEMENTATION Function</td>
<td>Retrieves the DOM implementation</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>GETLASTCHILD Function</td>
<td>Retrieves the last child of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETLENGTH Functions</td>
<td>Retrieves  · the length of the data  · the number of items in the map  · the number of items in the list</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>GETLOCALNAME Procedure and Functions</td>
<td>Retrieves</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td></td>
<td>the local part of the qualified name</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td></td>
<td>the local name of the attribute</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td></td>
<td>the local name of the element</td>
<td></td>
</tr>
<tr>
<td>GETNAME Functions</td>
<td>Retrieves</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td></td>
<td>the name of the attribute</td>
<td>DOMDocumentType Subprograms</td>
</tr>
<tr>
<td></td>
<td>the name of the Document type</td>
<td></td>
</tr>
<tr>
<td>GETNAMEDITEM Function</td>
<td>Retrieves</td>
<td>DOMNamedNodeMap Subprograms</td>
</tr>
<tr>
<td></td>
<td>an item specified by name and namespace URI</td>
<td>DOMNamedNodeMap Subprograms</td>
</tr>
<tr>
<td>GETNAMESPACE Procedure and Functions</td>
<td>Retrieves</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td></td>
<td>the node's namespace URI</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td></td>
<td>the NS URI of the attribute</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td></td>
<td>the NS URI of the element</td>
<td></td>
</tr>
<tr>
<td>GETNEXTSIBLING Function</td>
<td>Retrieves the next sibling of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETNODENAME Function</td>
<td>Retrieves the Name of the Node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETNODETYPE Function</td>
<td>Retrieves the Type of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETNODEVALUE Function</td>
<td>Retrieves the Value of the Node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETNODEVALUEASBINARYSTREAM Function &amp; Procedure</td>
<td>Retrieves the Node Value as binary stream</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETNODEVALUEASECHARACTERSTREAM Function &amp; Procedure</td>
<td>Retrieves the Node Value as character stream</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETNOTATIONNAME Function</td>
<td>Retrieves the notation name of the entity</td>
<td>DOMEntity Subprograms</td>
</tr>
<tr>
<td>GETNOTATIONS Function</td>
<td>Retrieves the nodemap of the notations in the Document type</td>
<td>DOMDocumentType Subprograms</td>
</tr>
<tr>
<td>GETTARGET Function</td>
<td>Retrieves the target of the processing instruction</td>
<td>DOMProcessingInstruction Subprograms</td>
</tr>
<tr>
<td>GETOWNERDOCUMENT Function</td>
<td>Retrieves the owner document of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETOWNERELEMENT Function</td>
<td>Retrieves the Element node, parent of the attribute</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td>GETPARENTNODE Function</td>
<td>Retrieves the parent of this node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETPREFIX Function</td>
<td>Retrieves the namespace prefix</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETPREVIOUSSIBLING Function</td>
<td>Retrieves the previous sibling of the node</td>
<td>DOMNode Subprograms</td>
</tr>
</tbody>
</table>
## Table 200-21  (Cont.) Summary of DBMS_XMLDOM Package Subprogram

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPUBLICID Functions</td>
<td>Retrieves</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• the public ID of the document type</td>
<td>• DOMDocumentType Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the public Id of the entity</td>
<td>• DOMEntity Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the public Id of the notation</td>
<td>• DOMNotation Subprograms</td>
</tr>
<tr>
<td></td>
<td>GETQUALIFIEDNAME Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retrieves</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• the Qualified Name of the attribute</td>
<td>• DOMAttr Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the qualified name of the element</td>
<td>• DOMElement Subprograms</td>
</tr>
<tr>
<td>GETSCHEMANODE Function</td>
<td>Retrieves the associated schema URI</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>GETSPECIFIED Function</td>
<td>Tests if attribute was specified in the element.</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td>GETSTANDALONE Function</td>
<td>Retrieves the standalone property of the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>GETSYSTEMID Functions</td>
<td>Retrieves</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• the system ID of the document type</td>
<td>• DOMDocumentType Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the system Id of the entity</td>
<td>• DOMEntity Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the system Id of the notation</td>
<td>• DOMNotation Subprograms</td>
</tr>
<tr>
<td>GETTAGNAME Function</td>
<td>Retrieves the Tag name of the element</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td>GETVALUE Function</td>
<td>Retrieves the value of the attribute</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td>GETVERSION Function</td>
<td>Retrieves the version of the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>GETXMLTYPE Function</td>
<td>Retrieves the XMLType associated with the DOM Document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>HASATTRIBUTES Function</td>
<td>Tests if the node has attributes</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>HASATTRIBUTE Functions</td>
<td>Tests if an attribute exists</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td>HASCHILDNODES Function</td>
<td>Tests if the node has child nodes</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>HASFEATURE Function</td>
<td>Tests if the DOMImplementation implements a feature</td>
<td>DOMImplementation Subprograms</td>
</tr>
<tr>
<td>IMPORTNODE Function</td>
<td>Imports a node from another document</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>INSERTBEFORE Function</td>
<td>Inserts a child before the reference child</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>INSERTDATA Procedure</td>
<td>Inserts the data in the node at the specified offSets</td>
<td>DOMCharacterData Subprograms</td>
</tr>
</tbody>
</table>

---

**Summary of DBMS_XMLDOM Subprograms**

Chapter 200
### Table 200-21  (Cont.) Summary of DBMS_XMLDOM Package Subprogram

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ISNULL Functions</strong></td>
<td>Tests</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· if the node is NULL</td>
<td>· DOMNode Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the Attribute node is NULL</td>
<td>· DOMAttr Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the CDataSection is NULL</td>
<td>· DOMCDataSection Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the CharacterData is NULL</td>
<td>· DOMCharacterData Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the comment is NULL</td>
<td>· DOMComment Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the document is NULL</td>
<td>· DOMDocument Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the DocumentFragment is NULL</td>
<td>· DOMDocumentFragment Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the Document Type is NULL</td>
<td>· DOMDocumentType Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the Element is NULL</td>
<td>· DOMElement Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the Entity is NULL</td>
<td>· DOMEntity Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the DOMEntityReference is NULL</td>
<td>· DOMEntityReference Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the DOMImplementation node is NULL</td>
<td>· DOMImplementation Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the NamedNodeMap is NULL</td>
<td>· DOMNamedNodeMap Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the NodeList is NULL</td>
<td>· DOMNodeList Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the Notation is NULL</td>
<td>· DOMNotation Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the Processing Instruction is NULL</td>
<td>· DOMProcessingInstruction Subprograms</td>
</tr>
<tr>
<td></td>
<td>· if the text is NULL</td>
<td>· DOMText Subprograms</td>
</tr>
<tr>
<td><strong>ITEM Functions</strong></td>
<td>Retrieves</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· the item given the index in the map</td>
<td>· DOMNamedNodeMap Subprograms</td>
</tr>
<tr>
<td></td>
<td>· the item given the index in the NodeList</td>
<td>· DOMNodeList Subprograms</td>
</tr>
<tr>
<td><strong>MAKEATTR Function</strong></td>
<td>Casts the node to an Attribute</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>MAKECDATASECTION Function</strong></td>
<td>Casts the node to a CData Section</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>MAKECHARACTERDATA Function</strong></td>
<td>Casts the node to Character Data</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>MAKECOMMENT Function</strong></td>
<td>Casts the node to a Comment</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>MAKEDOCUMENT Function</strong></td>
<td>Casts the node to a DOM Document</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>MAKEDOCUMENTFRAGMENT Function</strong></td>
<td>Casts the node to a DOM Document Fragment</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>MADEDOCUMENTTYPE Function</td>
<td>Casts the node to a DOM Document Type</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>MAKEELEMENT Function</td>
<td>Casts the node to a DOM Element</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>MAKEENTITY Function</td>
<td>Casts the node to a DOM Entity</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>MAKEENTITYREFERENCE Function</td>
<td>Casts the node to a DOM Entity Reference</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>MAKENODE Functions</td>
<td>Casts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• the Attribute to a node</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the CDatasection to a node</td>
<td>DOMCDatasection Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the CharacterData to a node</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the Comment to a node</td>
<td>DOMComment Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the document to a node</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the Document Fragment to a node</td>
<td>DOMDocumentFragment Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the document type to a node</td>
<td>DOMDocumentType Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the Element to a node</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the Entity to a node</td>
<td>DOMEntity Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the DOMEntityReference to NULL</td>
<td>DOMEntityReference Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the notation to a node</td>
<td>DOMNotation Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the Processing Instruction to a node</td>
<td>DOMProcessingInstruction Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the text to a node</td>
<td>DOMText Subprograms</td>
</tr>
<tr>
<td>MAKENOTATION Function</td>
<td>Casts the node to a DOM Notation</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>MAKEPROCESSINGINSTRUCTION Function</td>
<td>Casts the node to a DOM Processing Instruction</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>MAKETEXT Function</td>
<td>Casts the node to a DOM Text</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td>NEWDOMDOCUMENT Functions</td>
<td>Creates a new document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td>NORMALIZE Procedure</td>
<td>Normalizes the text children of the element</td>
<td>DOMElemet Subprograms</td>
</tr>
<tr>
<td>REMOVEATTRIBUTE Procedures</td>
<td>Removes the attribute specified by the name</td>
<td>DOMElemet Subprograms</td>
</tr>
<tr>
<td>REMOVEATTRIBUTENODE Function</td>
<td>Removes the attribute node in the element</td>
<td>DOMElemet Subprograms</td>
</tr>
<tr>
<td>REMOVECHILD Function</td>
<td>Removes a specified child from a node</td>
<td>DOMNode Subprograms</td>
</tr>
</tbody>
</table>
### Table 200-21  (Cont.) Summary of DBMS_XMLDOM Package Subprogram

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REMOVENAMEITEM Function</strong></td>
<td>Removes the item specified by name</td>
<td>DOMNamedNodeMap Subprograms</td>
</tr>
<tr>
<td><strong>REPLACECHILD Function</strong></td>
<td>Replaces the old child with a new child</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>REPLACEDATA Procedure</strong></td>
<td>Changes a range of characters in the node</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td><strong>RESOLVENAMESPACEPREFIX Function</strong></td>
<td>Resolve the prefix to a namespace URI</td>
<td>DOMElemnt Subprograms</td>
</tr>
<tr>
<td><strong>SETATTRIBUTE Procedures</strong></td>
<td>Sets the attribute specified by the name</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td><strong>SETATTRIBUTENODE Functions</strong></td>
<td>Sets the attribute node in the element</td>
<td>DOMElement Subprograms</td>
</tr>
<tr>
<td><strong>SETCHARSET Procedure</strong></td>
<td>Sets the characterset of the DOM document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td><strong>SETDATA Procedures</strong></td>
<td>Sets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• the data to the node</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td></td>
<td>• the data of the processing instruction</td>
<td>DOMProcessingInstruction Subprograms</td>
</tr>
<tr>
<td><strong>SETDOCTYPE Procedure</strong></td>
<td>Sets the DTD of the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td><strong>SETNAMEDITEM Function</strong></td>
<td>Sets the item in the map specified by the name</td>
<td>DOMNamedNodeMap Subprograms</td>
</tr>
<tr>
<td><strong>SETNODEVALUE Procedure</strong></td>
<td>Sets the Value of the node</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>SETNODEVALUEASBINARYSTREAM Function &amp; Procedure</strong></td>
<td>Sets the Node Value as a binary stream</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>SETNODEVALUEASCHARACTERSTREAM Function &amp; Procedure</strong></td>
<td>Sets the Node Value as a character stream</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>SETPREFIX Procedure</strong></td>
<td>Sets the namespace prefix</td>
<td>DOMNode Subprograms</td>
</tr>
<tr>
<td><strong>SETSTANDALONE Procedure</strong></td>
<td>Sets the standalone property of the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td><strong>SETVALUE Procedure</strong></td>
<td>Sets the value of the attribute</td>
<td>DOMAttr Subprograms</td>
</tr>
<tr>
<td><strong>SETVERSION Procedure</strong></td>
<td>Sets the version of the document</td>
<td>DOMDocument Subprograms</td>
</tr>
<tr>
<td><strong>SPLITTEXT Function</strong></td>
<td>Splits the contents of the text node into 2 text nodes</td>
<td>DOMText Subprograms</td>
</tr>
<tr>
<td><strong>SUBSTRINGDATA Function</strong></td>
<td>Retrieves the substring of the data</td>
<td>DOMCharacterData Subprograms</td>
</tr>
<tr>
<td><strong>USEBINARYSTREAM Function</strong></td>
<td>Strabismus that the stream is valid for use</td>
<td>DOMNode Subprograms</td>
</tr>
</tbody>
</table>
### Table 200-21 (Cont.) Summary of DBMS_XMLDOM Package Subprogram

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
</table>
| WRITETOBUFFER Procedures | Writes  
  - the contents of the node to a buffer  
  - the document to a buffer  
  - the contents of a document fragment into a buffer | DOMNode Subprograms  
  DOMDocument Subprograms  
  DOMDocumentFragment Subprograms |
| WRITETOCLOB Procedures | Writes  
  - the contents of the node to a CLOB  
  - the document to a CLOB | DOMNode Subprograms  
  DOMDocument Subprograms |
| WRITETOFILE Procedures | Writes  
  - the contents of the node to a file  
  - the document to a file | DOMNode Subprograms  
  DOMDocument Subprograms |

#### 200.7.1 ADOPTNODE Function

This function adopts a node from another document, and returns this new node.

> See Also:

DOMNode Subprograms for other subprograms in this group

**Syntax**

```sql
DBMS_XMLDOM.ADOPTNODE(
    doc            IN   DOMDocument,
    importedNode   IN   DOMNode)
RETURN DOMNODE;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>Document that is adopting the node</td>
</tr>
<tr>
<td>importedNode</td>
<td>Node to adopt</td>
</tr>
</tbody>
</table>

**Usage Notes**

Note that the ADOPTNODE Function removes the node from the source document while the IMPORTNODE Function clones the node in the source document.
200.7.2 APPENDCHILD Function

This function adds the node `newchild` to the end of the list of children of this node, and returns the newly added node. If the `newchild` is already in the tree, it is first removed.

See Also:

DOMNode Subprograms

Syntax

```sql
DBMS_XMLDOM.APPENDCHILD(
    n          IN    DOMNode,
    newchild   IN    DOMNode)
RETURN DOMNODE;
```

Parameters

Table 200-23  APPENDCHILD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNode</td>
</tr>
<tr>
<td>newchild</td>
<td>The child to be appended to the list of children of node <code>n</code></td>
</tr>
</tbody>
</table>

200.7.3 APPENDDATA Procedure

This procedure appends the string to the end of the character data of the node. Upon success, `data` provides access to the concatenation of `data` and the specified string argument.

See Also:

DOMCharacterData Subprograms

Syntax

```sql
DBMS_XMLDOM.APPENDDATA(
    cd      IN    DOMCHARACTERDATA,
    arg     IN    VARCHAR2);
```

Parameters

Table 200-24  APPENDDATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
</tbody>
</table>
Table 200-24  (Cont.) APPENDDATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg</td>
<td>The data to append to the existing data</td>
</tr>
</tbody>
</table>

200.7.4 CLONENODE Function

This function returns a duplicate of this node, and serves as a generic copy constructor for nodes. The duplicate node has no parent, its parent node is NULL.

See Also:

DOMNode Subprograms

Syntax

```sql
DBMS_XMLDOM.CLONENODE(
  n       IN    DOMNODE,
  deep    IN    BOOLEAN)
RETURN DOMNODE;
```

Parameters

Table 200-25  CLONENODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>deep</td>
<td>Determines if children are to be cloned</td>
</tr>
</tbody>
</table>

Usage Notes

- Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node.
- Cloning an Attribute directly, as opposed to be cloned as part of an Element cloning operation, returns a specified attribute (specified is TRUE).
- Cloning any other type of node simply returns a copy of this node.

200.7.5 CREATEATTRIBUTE Functions

This function creates a DOMATTR node.

See Also:

DOMDocument Subprograms
Syntax

Creates a DOMATTR with the specified name:

```sql
DBMS_XMLDOM.CREATEATTRIBUTE(
    doc     IN    DOMDOCUMENT,
    name    IN    VARCHAR2)
RETURN DOMATTR;
```

Creates a DOMATTR with the specified name and namespace URI:

```sql
DBMS_XMLDOM.CREATEATTRIBUTE(
    doc     IN    DOMDOCUMENT,
    qname   IN    VARCHAR2,
    ns      IN     VARCHAR2)
RETURN DOMATTR;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>qname</td>
<td>New attribute qualified name</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

### 200.7.6 CREATECDATASECTION Function

This function creates a DOMCDATASECTION node.

#### See Also:

- DOMDocument Subprograms

Syntax

```sql
DBMS_XMLDOM.CREATECDATASECTION(
    doc     IN    DOMDOCUMENT,
    data    IN    VARCHAR2)
RETURN DOMCDATASECTION;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>data</td>
<td>Content of the DOMCDATASECTION node</td>
</tr>
</tbody>
</table>
200.7.7 CREATECOMMENT Function

This function creates a DOMCOMMENT node.

See Also:
DOMDocument Subprograms

Syntax

```sql
DBMS_XMLDOM.CREATECOMMENT(
    doc      IN      DOMDOCUMENT,
    data     IN      VARCHAR2)
RETURN DOMCOMMENT;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>data</td>
<td>Content of the DOMComment node</td>
</tr>
</tbody>
</table>

200.7.8 CREATEDOCUMENT Function

This function creates a DOMDOCUMENT with specified namespace URI, root element name, DTD.

See Also:
DOMDocument Subprograms

Syntax

```sql
DBMS_XMLDOM.CREATEDOCUMENT(
    namespaceURI      IN     VARCHAR2,
    qualifiedName     IN     VARCHAR2,
    doctype           IN     DOMTYPE := NULL)
RETURN DOMDOCUMENT;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Namespace URI</td>
</tr>
</tbody>
</table>
Table 200-29 (Cont.) CREATEDOCUMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qualifiedName</td>
<td>Root element name</td>
</tr>
<tr>
<td>doctype</td>
<td>Document type</td>
</tr>
</tbody>
</table>

200.7.9 CREATEDOCUMENTFRAGMENT Function

This function creates a DOMDOCUMENTFRAGMENT.

See Also:
DOMDocument Subprograms

Syntax

```
DBMS_XMLDOM.CREATEDOCUMENTFRAGMENT(
    doc IN DOMDOCUMENT)
RETURN DOMDOCUMENTFRAGMENT;
```

Parameters

Table 200-30 CREATEDOCUMENTFRAGMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDocument</td>
</tr>
</tbody>
</table>

200.7.10 CREATEELEMENT Functions

This function creates a DOMELEMENT.

See Also:
DOMDocument Subprograms

Syntax

Creates a DOMElement with specified name:

```
DBMS_XMLDOM.CREATEELEMENT(
    doc        IN      DOMDOCUMENT,
    tagName    IN      VARCHAR2)
RETURN DOMELEMENT;
```

Creates a DOMElement with specified name and namespace URI:
DBMS_XMLDOM.CREATEELEMENT(  
  doc        IN DOMDOCUMENT,  
  tagName    IN VARCHAR2,  
  ns         IN VARCHAR2)  
RETURN DOMELEMENT;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>tagName</td>
<td>Tagname for new DOMELEMENT</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

200.7.11 CREATEENTITYREFERENCE Function

This function creates a DOMENTITYREFERENCE node.

See Also:
DOMDocument Subprograms

Syntax

DBMS_XMLDOM.CREATEENTITYREFERENCE(  
  doc        IN DOMDOCUMENT,  
  name       IN VARCHAR2)  
RETURN DOMENTITYREFERENCE;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>name</td>
<td>New entity reference name</td>
</tr>
</tbody>
</table>

200.7.12 CREATEPROCESSINGINSTRUCTION Function

This function creates a DOMPROCESSINGINSTRUCTION node.

See Also:
DOMDocument Subprograms
Syntax

```
DBMS_XMLDOM.CREATEPROCESSINGINSTRUCTION(
    doc     IN     DOMDocument,
    target  IN     VARCHAR2,
    data    IN     VARCHAR2)
RETURN DOMPROCESSINGINSTRUCTION;
```

Parameters

Table 200-33  CREATEPROCESSINGINSTRUCTION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>target</td>
<td>Target of the new processing instruction</td>
</tr>
<tr>
<td>data</td>
<td>Content data of the new processing instruction</td>
</tr>
</tbody>
</table>

200.7.13 CREATETEXTNODE Function

This function creates a DOMTEXT node.

See Also:

DOMDocument Subprograms

Syntax

```
DBMS_XMLDOM.CREATETEXTNODE(
    doc     IN     DOMDocument,
    data    IN     VARCHAR2)
RETURN DOMTEXT;
```

Parameters

Table 200-34  CREATETEXTNODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>data</td>
<td>Content of the DOMText node</td>
</tr>
</tbody>
</table>
200.7.14 DELETEDATA Procedure

This procedure removes a range of characters from the node. Upon success, data and length reflect the change.

**See Also:**
DOMCharacterData Subprograms

**Syntax**

```
DBMS_XMLDOM.DELETEDATA(
    cd        IN     DOMCHARACTERDATA,
    offset    IN     NUMBER,
    cnt       IN     NUMBER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>offset</td>
<td>The offset from which to delete the data</td>
</tr>
<tr>
<td>cnt</td>
<td>The number of characters (starting from offset) to delete</td>
</tr>
</tbody>
</table>

200.7.15 FINDENTITY Function

This function finds an entity in the specified DTD, and returns that entity if found.

**See Also:**
DOMDocumentType Subprograms

**Syntax**

```
DBMS_XMLDOM.FINDENTITY(
    dt     IN     DOMDOCUMENTTYPE,
    name   IN     VARCHAR2,
    par    IN     BOOLEAN) RETURN  DOMENTITY;
```
Parameters

Table 200-36 FINDENTITY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>The DTD</td>
</tr>
<tr>
<td>name</td>
<td>Entity to find</td>
</tr>
<tr>
<td>par</td>
<td>Flag to indicate type of entity; TRUE for parameter entity and FALSE for normal entity</td>
</tr>
</tbody>
</table>

200.7.16 FINDNOTATION Function

This function finds the notation in the specified DTD, and returns it, if found.

See Also:

DOMDocumentType Subprograms

Syntax

DBMS_XMLDOM.FINDNOTATION(
    dt IN DOMDocumentType,
    name IN VARCHAR2)
RETURN DOMNOTATION;

Parameters

Table 200-37 FINDNOTATION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>The DTD</td>
</tr>
<tr>
<td>name</td>
<td>The notation to find</td>
</tr>
</tbody>
</table>

200.7.17 FREEDOCFRAG Procedure

This procedure frees the specified document fragment.

See Also:

DOMDocument Subprograms and DOMDocumentFragment Subprograms

Syntax

DBMS_XMLDOM.FREEDOCFRAG(
    df IN DOMDOCUMENTFRAGMENT);
200.7.18 FREEDOCUMENT Procedure

This procedure frees DOMDOCUMENT object.

See Also:

DOMDocument Subprograms

Syntax

DBMS_XMLDOM.FREEDOCUMENT(
    doc     IN     DOMDOCUMENT);

Parameters

Table 200-39  FREEDOCUMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
</tbody>
</table>

200.7.19 FREEELEMENT Procedure

This procedure frees memory allocated to a DOMElement handle.

See Also:

DBMS_XMLDOM DOMElement Subprograms

Syntax

DBMS_XMLDOM.FREEELEMENT(
    elem IN DOMELEMENT);
Parameters

Table 200-40  FREEELEMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>Of type DOMELEMENT</td>
</tr>
</tbody>
</table>

200.7.20 FREENODE Procedure

This procedure frees all resources associated with a DOMNODE.

See Also:

DOMNode Subprograms

Syntax

DBMS_XMLDOM.FREENODE(
    n  IN  DOMNODE);

Parameters

Table 200-41  FREENODE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

200.7.21 FREENODELIST Procedure

This procedure frees all resources associated with a nodelist.

See Also:

DBMS_XMLDOM DOMNodeList Subprograms

Syntax

DBMS_XMLDOM.FREENODELIST(
    nl  IN  DOMNodeList);

### Parameters

#### Table 200-42  FREENODELIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>Of type DOMNODELIST</td>
</tr>
</tbody>
</table>

### 200.7.22 GETATTRIBUTE Functions

This function returns the value of an attribute of a DOMELEMENT by name.

#### Syntax

**Returns the value of a DOMELEMENT’s attribute by name:**

```sql
DBMS_XMLDOM.GETATTRIBUTE(
    elem  IN  DOMELEMENT,
    name  IN  VARCHAR2)
RETURN VARCHAR2;
```

**Returns the value of a DOMELEMENT’s attribute by name and namespace URI:**

```sql
DBMS_XMLDOM.GETATTRIBUTE(
    elem  IN  DOMELEMENT,
    name  IN  VARCHAR2,
    ns    IN  VARCHAR2)
RETURN VARCHAR2;
```

#### Parameters

#### Table 200-43  GETATTRIBUTE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>name</td>
<td>Attribute name</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

### See Also:

DOMElement Subprograms
200.7.23 GETATTRIBUTENODE Functions

This function returns an attribute node from the DOMELEMENT by name. The function is overloaded. The specific forms of functionality are described along with the syntax declarations.

See Also:

DOMElement Subprograms

Syntax

Returns an attribute node from the DOMELEMENT by name:

```sql
DBMS_XMLDOM.GETATTRIBUTENODE(
    elem      IN     DOMElement,
    name      IN     VARCHAR2)
RETURN DOMATTR;
```

Returns an attribute node from the DOMELEMENT by name and namespace URI:

```sql
DBMS_XMLDOM.GETATTRIBUTENODE(
    elem      IN     DOMElement,
    name      IN     VARCHAR2,
    ns        IN     VARCHAR2)
RETURN DOMATTR;
```

Parameters

**Table 200-44  GETATTRIBUTENODE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>name</td>
<td>Attribute name; * matches any attribute</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

200.7.24 GETATTRIBUTES Function

This function retrieves a NAMEDNODEMAP containing the attributes of this node (if it is an Element) or NULL otherwise.

See Also:

DOMNode Subprograms
Syntax

DBMS_XMLDOM.GETATTRIBUTES(
    n    IN    DOMNode)
RETURN DOMNAMEDNODEMAP;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

200.7.25 GETCHARSET Function

This function retrieves the characterset of the DOM document.

See Also:

DOMDocument Subprograms

Syntax

DBMS_XMLDOM.GETCHARSET(
    doc    IN    DOMDocument)
RETURN VARCHAR2;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOM document</td>
</tr>
</tbody>
</table>

Usage Notes

For a newly parsed document, we return the database characterset. Once the SETCHARSET Procedure is called with a non-NULL value for charset, that charset is returned.

200.7.26 GETCHILDNODES Function

This function retrieves a DOMNODELIST that contains all children of this node. If there are no children, this is a DOMNODELIST containing no nodes.

See Also:

DOMNode Subprograms
Syntax

DBMS_XMLDOM.GETCHILDNODES(
    n    IN    DOMNode)
RETURN DOMNodeList;

Parameters

Table 200-47  GETCHILDNODES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNode</td>
</tr>
</tbody>
</table>

200.7.27 GETCHILDRENBYTAGNAME Functions

This function returns the children of the DOMElement.

See Also:

DOMElement Subprograms

Syntax

Returns children of the DOMElement given the tag name:

DBMS_XMLDOM.GETCHILDRENBYTAGNAME(
    elem    IN    DOMElement,
    name    IN    VARCHAR2)
RETURN DOMNODELIST;

Returns children of the DOMElement given the tag name and namespace:

DBMS_XMLDOM.GETCHILDRENBYTAGNAME(
    elem    IN    DOMElement,
    name    IN    VARCHAR2,
    ns      IN    VARCHAR2)
RETURN DOMNODELIST;

Parameters

Table 200-48  GETCHILDRENBYTAGNAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>DOMElement</td>
</tr>
<tr>
<td>name</td>
<td>Tag name</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>
200.7.28 GETDATA Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Gets the character data of the node that implements this interface (See Also: DOMCharacterData Subprograms):

```sql
DBMS_XMLDOM.GETDATA(
    cd IN DOMCHARACTERDATA
) RETURN VARCHAR2;
```

Returns the content data of the DOMProcessingInstruction (See Also: DOMProcessingInstruction Subprograms):

```sql
DBMS_XMLDOM.GETDATA(
    pi IN DOMPROCESSINGINSTRUCTION
) RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>pi</td>
<td>DOMPROCESSINGINSTRUCTION</td>
</tr>
</tbody>
</table>

200.7.29 GETDOCTYPE Function

This function returns the DTD associated to the DOMDOCUMENT.

See Also:
DOMDocument Subprograms

Syntax

```sql
DBMS_XMLDOM.GETDOCTYPE(
    doc IN DOMDOCUMENT
) RETURN DOMDOCUMENTTYPE;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
</tbody>
</table>
200.7.30 GETDOCUMENTELEMENT Function

This function returns the root element of the DOMDOCUMENT.

Syntax

```
DBMS_XMLDOM.GETDOCUMENTELEMENT(
    doc IN DOMDOCUMENT)
RETURN DOMELEMENT;
```

Parameters

Table 200-51  GETDOCUMENTELEMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
</tbody>
</table>

200.7.31 GETELEMENTSBYTAGNAME Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns a DOMNODELIST of all the elements with a specified tagname (See Also: DOMDocument Subprograms):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME(
    doc IN DOMDOCUMENT,
    tagname IN VARCHAR2)
RETURN DOMNODELIST;
```

Returns the element children of the DOMELEMENT given the tag name (See Also: DOMEElement Subprograms):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME(
    elem IN DOMELEMENT,
    name IN VARCHAR2)
RETURN DOMNODELIST;
```

Returns the element children of the DOMELEMENT given the tag name and namespace (See Also: DOMEElement Subprograms):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME(
    elem IN DOMELEMENT,
    name IN VARCHAR2,
    ns IN VARCHAR2)
RETURN DOMNODELIST;
```
Parameters

Table 200-52  GETELEMENTSBYTAGNAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>tagname</td>
<td>Name of the tag to match on</td>
</tr>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>name</td>
<td>Tag name; using a wildcard(*) would match any tag</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

200.7.32 GETENTITIES Function

This function retrieves a DOMNAMEDNODEMAP containing the general entities, both external and internal, declared in the DTD.

See Also:
DOMDocumentType Subprograms

Syntax

```
DBMS_XMLDOM.GETENTITIES(
    dt      IN     DOMDocumentType)
RETURN DOMNAMEDNODEMAP;
```

Parameters

Table 200-53  GETENTITIES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>DOMDOCUMENTTYPE</td>
</tr>
</tbody>
</table>

200.7.33 GETEXPANDEDNAME Procedure and Functions

This subprogram is overloaded as a procedure and two functions. The specific forms of functionality are described along with the syntax declarations.

Syntax

```
Retrieves the expanded name of the Node if is in an Element or Attribute type; otherwise, returns NULL (See Also: DOMNode Subprograms)
```

```
DBMS_XMLDOM.GETEXPANDEDNAME(
    n       IN      DOMNODE
    data    OUT     VARCHAR);
```

Returns the expanded name of the DOMAttr (See Also: DOMAttr Subprograms):
DBMS_XMLDOM.GETEXPANDEDNAME(
    a       IN     DOMAttr)
RETURN VARCHAR2;

Returns the expanded name of the DOMElement (See Also: DOMElement Subprograms):

DBMS_XMLDOM.GETEXPANDEDNAME(
    elem      IN    DOMELEMENT)
RETURN VARCHAR2;

Parameters

Table 200-54  GETEXPANDEDNAME Procedure and Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>data</td>
<td>Returned expanded name of the Node</td>
</tr>
<tr>
<td>a</td>
<td>DOMATTR</td>
</tr>
<tr>
<td>elem</td>
<td>DOMELEMENT</td>
</tr>
</tbody>
</table>

200.7.34 GETFIRSTCHILD Function

This function retrieves the first child of this node. If there is no such node, this returns NULL.

See Also:

DOMNode Subprograms

Syntax

DBMS_XMLDOM.GETFIRSTCHILD(
    n      IN      DOMNODE)
RETURN DOMNODE;

Parameters

Table 200-55  GETFIRSTCHILD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
200.7.35 GETIMPLEMENTATION Function

This function returns the DOMIMPLEMENTATION object that handles this DOMDOCUMENT.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
</tbody>
</table>

See Also:
DOMDocument Subprograms

Syntax

DBMS_XMLDOM.GETIMPLEMENTATION(
   doc IN DOMDOCUMENT)
RETURN DOMIMPLEMENTATION;

Parameters

Table 200-56 GETIMPLEMENTATION Function Parameters

200.7.36 GETLASTCHILD Function

This function retrieves the last child of this node. If there is no such node, this returns NULL.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.GETLASTCHILD(
   n IN DOMNODE)
RETURN DOMNODE;

Parameters

Table 200-57 GETLASTCHILD Function Parameters
200.7.37 GETLENGTH Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Gets the number of characters in the data. This may have the value zero, because CharacterData nodes may be empty (See Also: DOMCharacterData Subprograms):

```
DBMS_XMLDOM.GETLENGTH(
    cd     IN     DOMCHARACTERDATA)
RETURN NUMBER;
```

Gets the number of nodes in this map. The range of valid child node indexes is 0 to length-1, inclusive (See Also: DOMNamedNodeMap Subprograms):

```
DBMS_XMLDOM.GETLENGTH(
    nnm     IN     DOMNAMEDNODEMAP)
RETURN NUMBER;
```

Gets the number of nodes in the list. The range of valid child node indexes is 0 to length-1, inclusive (See Also: DOMNodeList Subprograms):

```
DBMS_XMLDOM.GETLENGTH(
    nl     IN    DOMNODELIST)
RETURN NUMBER;
```

Parameters

Table 200-58  GETLENGTH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>nnm</td>
<td>DOMNAMEDNODEMAP</td>
</tr>
<tr>
<td>nl</td>
<td>DOMNODELIST</td>
</tr>
</tbody>
</table>

200.7.38 GETLOCALNAME Procedure and Functions

This function is overloaded as a procedure and two functions. The specific forms of functionality are described alongside the syntax declarations.

Syntax

Retrieves the local part of the node’s qualified name (See Also: DOMNode Subprograms):

```
DBMS_XMLDOM.GETLOCALNAME(
    n       IN     DOMNODE,
    data    OUT    VARCHAR2);
```

Returns the local name of the DOMAttr (See Also: DOMAttr Subprograms):

```
```

ORACLE
DBMS_XMLDOM.GETLOCALNAME (  
a       IN     DOMATTR)  
RETURN VARCHAR2;

Returns the local name of the DOMElemet (See Also: DOMElemet Subprograms)

DBMS_XMLDOM.GETLOCALNAME (  
elem       IN     DOMELEMENT)  
RETURN VARCHAR2;

**Parameters**

**Table 200-59  GETLOCALNAME Procedure and Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNode</td>
</tr>
<tr>
<td>data</td>
<td>Returned local name.</td>
</tr>
<tr>
<td>a</td>
<td>DOMAttr.</td>
</tr>
<tr>
<td>elem</td>
<td>DOMElemet.</td>
</tr>
</tbody>
</table>

### 200.7.39 GETNAME Functions

This function is overloaded. The specific forms of functionality are described with the syntax declarations.

**Syntax**

Returns the name of this attribute (See Also: DOMAttr Subprograms):

DBMS_XMLDOM.GETNAME (  
a       IN     DOMATTR)  
RETURN VARCHAR2;

Retrieves the name of DTD, or the name immediately following the DOCTYPE keyword (See Also: DOMDocumentType Subprograms):

DBMS_XMLDOM.GETNAME (  
dt       IN     DOMDOCUMENTTYPE)  
RETURN VARCHAR2;

**Parameters**

**Table 200-60  GETNAME Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>DOMATTR</td>
</tr>
<tr>
<td>dt</td>
<td>DOMDOCUMENTTYPE</td>
</tr>
</tbody>
</table>
200.7.40 GETNAMEDITEM Function

GETNAMEDITEM retrieves a node specified by name.

Syntax

Retrieves a node specified by name:

```sql
DBMS_XMLDOM.GETNAMEDITEM(
    nnm IN DOMNAMEDNODEMAP,
    name IN VARCHAR2)
RETURN DOMNODE;
```

Retrieves a node specified by name and namespace URI:

```sql
DBMS_XMLDOM.GETNAMEDITEM(
    nnm IN DOMNAMEDNODEMAP,
    name IN VARCHAR2,
    ns IN VARCHAR2)
RETURN DOMNODE;
```

Parameters

Table 200-61  GETNAMEDITEM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnm</td>
<td>DOMNAMEDNODEMAP</td>
</tr>
<tr>
<td>name</td>
<td>Name of the item to be retrieved</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

200.7.41 GETNAMESPACE Procedure and Functions

This subprogram is overloaded as a procedure and two functions. The specific forms of functionality are described alongside the syntax declarations.

Syntax

Retrieves the namespace URI associated with the node (See Also: DOMNode Subprograms):

```sql
DBMS_XMLDOM.GETNAMESPACE(
    n IN DOMNODE,
    data OUT VARCHAR2);
```

Retrieves the namespace of the DOMATTR (See Also: DOMAttr Subprograms):
Retrieves the namespace of the DOMElement (See Also: DOMElement Subprograms):

```
DBMS_XMLDOM.GETNAMESPACE(
   elem       IN     DOMELEMENT)
RETURN VARCHAR2;
```

Parameters

Table 200-62 GETNAMESPACE Procedure and Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>data</td>
<td>Returned namespace URI</td>
</tr>
<tr>
<td>a</td>
<td>DOMATTR</td>
</tr>
<tr>
<td>elem</td>
<td>DOMELEMENT</td>
</tr>
</tbody>
</table>

200.7.42 GETNEXTSIBLING Function

This function retrieves the node immediately following this node. If there is no such node, this returns NULL.

See Also:

DOMNode Subprograms

Syntax

```
DBMS_XMLDOM.GETNEXTSIBLING(
   n       IN     DOMNODE)
RETURN DOMNode;
```

Parameters

Table 200-63 GETNEXTSIBLING Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
200.7.43 GETNODETYPE Function

This function retrieves a code representing the type of the underlying object.

See Also:
DOMNode Subprograms

Syntax

```
DBMS_XMLDOM.GETNODETYPE(
   n       IN     DOMNODE)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

200.7.44 GETNODENAME Function

This function gets the name of the node depending on its type.

See Also:
DOMNode Subprograms

Syntax

```
DBMS_XMLDOM.GETNODENAME(
   n       IN     DOMNODE)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
200.7.45 GETNODEVALUE Function

This function gets the value of this node, depending on its type.

Syntax

```sql
DBMS_XMLDOM.GETNODEVALUE(
    n       IN     DOMNODE
RETURN VARCHAR2;*
```

Parameters

Table 200-66  GETNODEVALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

200.7.46 GETNODEVALUEASBINARYSTREAM Function & Procedure

The operation of these subprograms is described with each syntax implementation.

Syntax

This function returns an instance of the PL/SQL XMLBinaryInputStream. The node datatype must be RAW or BLOB – if not an exception is raised.

```sql
DBMS_XMLDOM.GETNODEVALUEASBINARYSTREAM (
    n      IN     DOMNODE
RETURN SYS.UTL_BINARYINPUTSTREAM;
```

Using this procedure, the application passes an implementation of SYS.UTL_BINARYOUTPUTSTREAM into which XDB writes the contents of the node. The datatype of the node must be RAW or CLOB – if not an exception is raised.

```sql
DBMS_XMLDOM.GETNODEVALUEASBINARYSTREAM (
    n        in   DOMNODE,
    value    in   SYS.UTL_BINARYOUTPUTSTREAM);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>value</td>
<td>BINARYOUTPUTSTREAM</td>
</tr>
</tbody>
</table>

**200.7.47 GETNODEVALUEASCHARACTERSTREAM Function & Procedure**

The operation of these subprograms is described with each syntax implementation.

**See Also:**

DOMNode Subprograms

**Syntax**

This function returns an instance of the PL/SQL XMLCharacterInputStream. If the node data is character it is converted to the current session character set. If the node data is not character data, it is first converted to character data.

```sql
DBMS_XMLDOM.GETNODEVALUEASCHARACTERSTREAM (n IN DOMNODE) RETURN SYS.UTL_CHARACTERINPUTSTREAM;
```

Using this procedure, the node data is converted, as necessary, to the session character set and then "pushed" into the SYS.UTL_CHARACTEROUTPUTSTREAM.

```sql
DBMS_XMLDOM.GETNODEVALUEASCHARACTERSTREAM (n IN DOMNODE, value IN SYS.UTL_CHARACTEROUTPUTSTREAM);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>value</td>
<td>CHARACTEROUTPUTSTREAM</td>
</tr>
</tbody>
</table>
200.7.48 GETNOTATIONNAME Function

This function returns the notation name of the DOMENTITY.

See Also:
DOMEntity Subprograms

Syntax

DBMS_XMLDOM.GETNOTATIONNAME(
    ent       IN     DOMENTITY)
RETURN VARCHAR2;

Parameters

Table 200-69  GETNOTATIONNAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ent</td>
<td>DOMENTITY</td>
</tr>
</tbody>
</table>

200.7.49 GETNOTATIONS Function

This function retrieves a DOMNAMEDNODEMAP containing the notations declared in the DTD.

See Also:
DOMDocumentType Subprograms

Syntax

DBMS_XMLDOM.GETNOTATIONS(
    dt       IN     DOMDOCUMENTTYPE)
RETURN DOMNAMEDNODEMAP;

Parameters

Table 200-70  GETNOTATIONS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>DOMDOCUMENTTYPE</td>
</tr>
</tbody>
</table>
200.7.50 GETTARGET Function

This function returns the target of the DOMPROCESSINGINSTRUCTION.

See Also:
DOMProcessingInstruction Subprograms

Syntax

DBMS_XMLDOM.GETTARGET(
    pi       IN     DOMPROCESSINGINSTRUCTION)
RETURN VARCHAR2;

Parameters

Table 200-71 GETTARGET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>DOMPROCESSINGINSTRUCTION</td>
</tr>
</tbody>
</table>

200.7.51 GETOWNERDOCUMENT Function

This function retrieves the Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document or a Document Type that is not used with any Document yet, this is NULL.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.GETOWNERDOCUMENT(
    n       IN     DOMNODE)
RETURN DOMDOCUMENT;

Parameters

Table 200-72 GETOWNERDOCUMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
200.7.52 GETOWNERELEMENT Function

This function retrieves the Element node to which the specified Attribute is attached.

```sql
DBMS_XMLDOM.GETOWNERELEMENT(
    a       IN     DOMATTR)
RETURN DOMElement;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Attribute</td>
</tr>
</tbody>
</table>

200.7.53 GETPARENTNODE Function

This function retrieves the parent of this node. All nodes, except Attr, Document, DocumentFragment, Entity, and Notation may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is NULL.

```sql
DBMS_XMLDOM.GETPARENTNODE(
    n       IN     DOMNODE)
RETURN DOMNODE;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
200.7.54 GETPREFIX Function

This function retrieves the namespace prefix of the node.

See Also:

DOMNode Subprograms

Syntax

DBMS_XMLDOM.GETPREFIX(
   n       IN     DOMNODE)
RETURN VARCHAR2;

Parameters

Table 200-75  GETPREFIX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

200.7.55 GETPREVIOUSSIBLING Function

This function retrieves the node immediately preceding this node. If there is no such node, this returns NULL.

See Also:

DOMNode Subprograms

Syntax

DBMS_XMLDOM.GETPREVIOUSSIBLING(
   n       IN     DOMNODE)
RETURN DOMNODE;

Parameters

Table 200-76  GETPREVIOUSSIBLING Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
200.7.56 GETPUBLICID Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the public identifier of the specified DTD (See Also: DOMDocumentType Subprograms):

```sql
DBMS_XMLDOM.GETPUBLICID(
    dt   IN     DOMDOCUMENTTYPE)
RETURN VARCHAR2;
```

Returns the public identifier of the DOMENTITY (See Also: DOMEntity Subprograms):

```sql
DBMS_XMLDOM.GETPUBLICID(
    ent  IN     DOMENTITY)
RETURN VARCHAR2;
```

Returns the public identifier of the DOMNOTATION (See Also: DOMNotation Subprograms):

```sql
DBMS_XMLDOM.GETPUBLICID(
    n    IN     DOMNOTATION)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>The DTD</td>
</tr>
<tr>
<td>ent</td>
<td>DOMENTITY</td>
</tr>
<tr>
<td>n</td>
<td>DOMNOTATION</td>
</tr>
</tbody>
</table>

200.7.57 GETQUALIFIEDNAME Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the qualified name of the DOMATTR (See Also: DOMAttr Subprograms):

```sql
DBMS_XMLDOM.GETQUALIFIEDNAME(
    a    IN     DOMATTR)
RETURN VARCHAR2;
```

Returns the qualified name of the DOMElement (See Also: DOMElement Subprograms):

```sql
DBMS_XMLDOM.GETQUALIFIEDNAME(
    elem IN     DOMELEMENT)
RETURN VARCHAR2;
```
### Parameters

#### Table 200-78  GETQUALIFIEDNAME Functions Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>DOMATTR</td>
</tr>
<tr>
<td>elem</td>
<td>DOMELEMENT</td>
</tr>
</tbody>
</table>

### 200.7.58 GETSCHEMANODE Function

This function retrieves the schema URI associated with the node.

#### See Also:

DOMNode Subprograms

#### Syntax

```sql
DBMS_XMLDOM.GETSCHEMANODE(
    n     IN     DOMNODE)
RETURN DOMNODE;
```

#### Parameters

#### Table 200-79  GETSCHEMANODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

### 200.7.59 GETSPECIFIED Function

If this attribute was explicitly specified, a value in the original document, this is true; otherwise, it is false.

#### See Also:

DOMAttr Subprograms

#### Syntax

```sql
DBMS_XMLDOM.GETSPECIFIED(
    a     IN     DOMATTR)
RETURN BOOLEAN;
```
Parameters

Table 200-80  GETSPECIFIED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>DOMATTR</td>
</tr>
</tbody>
</table>

200.7.60 GETSTANDALONE Function

This function returns the standalone property associated with the DOMDOCUMENT.

See Also:
DOMDocument Subprograms

Syntax

DBMS_XMLDOM.GETSTANDALONE(
  doc IN DOMDOCUMENT)
RETURN VARCHAR2;

Parameters

Table 200-81  GETSTANDALONE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
</tbody>
</table>

200.7.61 GETSYSTEMID Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the system id of the specified DTD (See Also: DOMDocumentType Subprograms):

DBMS_XMLDOM.GETSYSTEMID(
  dt IN DOMDOCUMENTTYPE)
RETURN VARCHAR2;

Returns the system identifier of the DOMENTITY (See Also: DOMEntity Subprograms):

DBMS_XMLDOM.GETSYSTEMID(
  ent IN DOMENTITY)
RETURN VARCHAR2;

Returns the system identifier of the DOMNOTATION (See Also: DOMNotation Subprograms):
DBMS_XMLDOM.GETSYSTEMID(
    n    IN    DOMNOTATION)
RETURN VARCHAR2;

Parameters

Table 200-82  GETSYSTEMID Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>The DTD.</td>
</tr>
<tr>
<td>ent</td>
<td>DOMEntity.</td>
</tr>
<tr>
<td>n</td>
<td>DOMNotation.</td>
</tr>
</tbody>
</table>

200.7.62 GETTAGNAME Function

This function returns the name of the DOMELEMENT.

See Also:

DOMElement Subprograms

Syntax

DBMS_XMLDOM.GETTAGNAME(
    elem    IN    DOMELEMENT)
RETURN VARCHAR2;

Parameters

Table 200-83  GETTAGNAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
</tbody>
</table>

200.7.63 GETVALUE Function

This function retrieves the value of the attribute.

See Also:

DOMAttr Subprograms

Syntax

DBMS_XMLDOM.GETVALUE(
    a    IN    DOMATTR)
RETURN VARCHAR2;
### Parameters

#### Table 200-84  GETVALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>DOMATTR</td>
</tr>
</tbody>
</table>

### 200.7.64 GETVERSION Function

This function returns the version of the `DOMDOCUMENT`.

#### Syntax

```sql
DBMS_XMLDOM.GETVERSION(
    doc       IN     DOMDOCUMENT)
RETURN VARCHAR2;
```

#### Parameters

#### Table 200-85  GETVERSION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
</tbody>
</table>

### 200.7.65 GETXMLTYPE Function

This function returns the `XMLType` associated with the `DOMDOCUMENT`.

#### See Also:

DOMDocument Subprograms

#### Syntax

```sql
DBMS_XMLDOM.GETXMLTYPE(
    doc       IN     DOMDOCUMENT)
RETURN SYS.XMLTYPE;
```
200.7.66 HASATTRIBUTE Functions

Verifies whether an attribute has been defined for DOMELEMENT, or has a default value.

See Also:

DOMElement Subprograms

Syntax

Verifies whether an attribute with the specified name has been defined for DOMELEMENT:

```sql
DBMS_XMLDOM.HASATTRIBUTE(
    elem     IN  DOMELEMENT,
    name     IN  VARCHAR2)
RETURN VARCHAR2;
```

Verifies whether an attribute with specified name and namespace URI has been defined for DOMELEMENT; namespace enabled:

```sql
DBMS_XMLDOM.HASATTRIBUTE(
    elem     IN  DOMELEMENT,
    name     IN  VARCHAR2,
    ns       IN  VARCHAR2)
RETURN VARCHAR2;
```
200.7.67 HASATTRIBUTES Function

This function returns whether this node has any attributes.

Syntax

DBMS_XMLDOM.HASATTRIBUTES(
    n       IN     DOMNODE)
RETURN BOOLEAN;

Parameters

Table 200-88  HASATTRIBUTES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

200.7.68 HASCHILDNODES Function

This function determines whether this node has any children.

Syntax

DBMS_XMLDOM.HASCHILDNODES(
    n       IN     DOMNODE)
RETURN BOOLEAN;

Parameters

Table 200-89  HASCHILDNODES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
200.7.69 HASFEATURE Function

This function tests if the DOMIMPLEMENTATION implements a specific feature.

See Also:
DOMImplementation Subprograms

Syntax

DBMS_XMLDOM.HASFEATURE(
    di       IN     DOMIMPLEMENTATION,
    feature  IN     VARCHAR2,
    version  IN     VARCHAR2)
RETURN BOOLEAN;

Parameters

Table 200-90  HASFEATURE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>di</td>
<td>DOMIMPLEMENTATION</td>
</tr>
<tr>
<td>feature</td>
<td>The feature to check for</td>
</tr>
<tr>
<td>version</td>
<td>The version of the DOM to check in</td>
</tr>
</tbody>
</table>

200.7.70 IMPORTNODE Function

This function imports a node from an external document and returns this new node.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.IMPORTNODE(
    doc            IN  DOMDOCUMENT,
    importedNode   IN  DOMNODE,
    deep           IN  BOOLEAN)
RETURN DOMNODE;
Parameters

**Table 200-91  IMPORTNODE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>Document from which the node is imported</td>
</tr>
<tr>
<td>importedNode</td>
<td>Node to import</td>
</tr>
<tr>
<td>deep</td>
<td>Setting for recursive import.</td>
</tr>
<tr>
<td></td>
<td>• If this value is TRUE, the entire subtree of the node will be imported with the node.</td>
</tr>
<tr>
<td></td>
<td>• If this value is FALSE, only the node itself will be imported.</td>
</tr>
</tbody>
</table>

Usage Notes

Note that the **ADOPTNODE Function** removes the node from the source document while the **IMPORTNODE Function** clones the node in the source document.

### 200.7.71 INSERTBEFORE Function

This function inserts the node `newchild` before the existing child node `refchild`. If `refchild` is NULL, insert `newchild` at the end of the list of children.

If `newchild` is a `DOCUMENTFRAGMENT` object, all of its children are inserted, in the same order, before `refchild`. If the `newchild` is already in the tree, it is first removed.

**See Also:**

DOMNode Subprograms

**Syntax**

```
DBMS_XMLDOM.INSERTBEFORE(
    n          IN     DOMNODE,
    newchild   IN     DOMNODE,
    refchild   IN     DOMNODE)
RETURN DOMNode;
```

**Parameters**

**Table 200-92  INSERTBEFORE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>newChild</td>
<td>The child to be inserted in the DOMNODE</td>
</tr>
<tr>
<td>refChild</td>
<td>The reference node before which the <code>newchild</code> is to be inserted</td>
</tr>
</tbody>
</table>
200.7.72 INSERTDATA Procedure

This procedure inserts a string at the specified character offset.

See Also:

DOMCharacterData Subprograms

Syntax

```
DBMS_XMLDOM.INSERTDATA(
    cd       IN     DOMCHARACTERDATA,
    offset   IN     NUMBER,
    arg      IN     VARCHAR2);
```

Parameters

Table 200-93  INSERTDATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>offset</td>
<td>The offset at which to insert the data</td>
</tr>
<tr>
<td>arg</td>
<td>The value to be inserted</td>
</tr>
</tbody>
</table>

200.7.73 ISNULL Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Checks if the specified DOMNODE is NULL. Returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNode Subprograms):

```
DBMS_XMLDOM.ISNULL(
    n        IN     DOMNODE)
RETURN BOOLEAN;
```

Checks that the specified DOMATTR is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMAttr Subprograms):

```
DBMS_XMLDOM.ISNULL(
    a        IN     DOMATTR)
RETURN BOOLEAN;
```

Checks that the specified DOMCDATASECTION is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMCDATASECTION Subprograms):

```
DBMS_XMLDOM.ISNULL(
    cds      IN     DOMCDATASECTION)
RETURN BOOLEAN;
```
Checks that the specified DOMCharacterData is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMCharacterData Subprograms):

DBMS_XMLDOM.ISNULL(
    cd IN DOMCharacterData)
RETURN BOOLEAN;

Checks that the specified DOMComment is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMComment Subprograms):

DBMS_XMLDOM.ISNULL(
    com IN DOMComment)
RETURN BOOLEAN;

Checks that the specified DOMDocument is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMDocument Subprograms):

DBMS_XMLDOM.ISNULL(
    doc IN DOMDocument)
RETURN BOOLEAN;

Checks that the specified DOMDocumentFragment is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMDocumentFragment Subprograms):

DBMS_XMLDOM.ISNULL(
    df IN DOMDocumentFragment)
RETURN BOOLEAN;

Checks that the specified DOMDocumentType is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMDocumentType Subprograms):

DBMS_XMLDOM.ISNULL(
    dt IN DOMDocumentType)
RETURN BOOLEAN;

Checks that the specified DOMElement is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMElement Subprograms):

DBMS_XMLDOM.ISNULL(
    elem IN DOMElement)
RETURN BOOLEAN;

Checks that the specified DOMEntity is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMEntity Subprograms):

DBMS_XMLDOM.ISNULL(
    ent IN DOMEntity)
RETURN BOOLEAN;

Checks that the specified DOMEntityReference is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMEntityReference Subprograms):

DBMS_XMLDOM.ISNULL(
    EREF IN DOMEntityReference)
RETURN BOOLEAN;

Checks that the specified DOMImplementation is NULL; returns TRUE if it is NULL (See Also: DOMImplementation Subprograms):

DBMS_XMLDOM.ISNULL(
    NULL)
RETURN BOOLEAN;
DBMS_XMLDOM.ISNULL(
    di       IN     DOMIMPLEMENTATION)
RETURN BOOLEAN;

Checks that the specified DOMNAMEDNODEMAP is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNamedNodeMap Subprograms):

DBMS_XMLDOM.ISNULL(
    nmm      IN     DOMNAMEDNODEMAP)
RETURN BOOLEAN;

Checks that the specified DOMNODELIST is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNodeList Subprograms):

DBMS_XMLDOM.ISNULL(
    nl       IN     DOMNODELIST)
RETURN BOOLEAN;

Checks that the specified DOMNOTATION is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMNotation Subprograms):

DBMS_XMLDOM.ISNULL(
    n       IN     DOMNOTATION)
RETURN BOOLEAN;

Checks that the specified DOMPROCESSINGINSTRUCTION is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMProcessingInstruction Subprograms):

DBMS_XMLDOM.ISNULL(
    pi       IN     DOMPROCESSINGINSTRUCTION)
RETURN BOOLEAN;

Checks that the specified DOMTEXT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: DOMText Subprograms):

DBMS_XMLDOM.ISNULL(
    t       IN     DOMTEXT)
RETURN BOOLEAN;

Parameters

Table 200-94  ISNULL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to check</td>
</tr>
<tr>
<td>a</td>
<td>DOMATTR to check</td>
</tr>
<tr>
<td>cds</td>
<td>DOMCDATASECTION to check</td>
</tr>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA to check</td>
</tr>
<tr>
<td>com</td>
<td>DOMCOMMENT to check</td>
</tr>
<tr>
<td>doc</td>
<td>DOMDOCUMENT to check</td>
</tr>
<tr>
<td>dF</td>
<td>DOMDOCUMENTFRAGMENT to check</td>
</tr>
<tr>
<td>dt</td>
<td>DOMDOCUMENTTYPE to check</td>
</tr>
<tr>
<td>elem</td>
<td>DOMELEMENT to check</td>
</tr>
<tr>
<td>ent</td>
<td>DOMENTITY to check</td>
</tr>
</tbody>
</table>
Table 200-94  (Cont.) ISNULL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eref</td>
<td>DOMENTITYREFERENCE to check</td>
</tr>
<tr>
<td>di</td>
<td>DOMIMPLEMENTATION to check</td>
</tr>
<tr>
<td>nnm</td>
<td>DOMNAMENODEMAP to check</td>
</tr>
<tr>
<td>nl</td>
<td>DOMNODELIST to check</td>
</tr>
<tr>
<td>n</td>
<td>DOMNOTATION to check</td>
</tr>
<tr>
<td>pi</td>
<td>DOMPROCESSINGINSTRUCTION to check</td>
</tr>
<tr>
<td>t</td>
<td>DOMTEXT to check</td>
</tr>
</tbody>
</table>

200.7.74 ITEM Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Returns the item in the map which corresponds to the \texttt{INDEX} parameter. If \texttt{INDEX} is greater than or equal to the number of nodes in this map, this returns \texttt{NULL} (See Also: \texttt{DOMNamedNodeMap Subprograms}):

\begin{verbatim}
DBMS_XMLDOM.ITEM(
    nnm    IN     DOMNAMEDNODEMAP,
    index IN     NUMBER)
RETURN DOMNODE;
\end{verbatim}

Returns the item in the collection which corresponds to the \texttt{INDEX} parameter. If index is greater than or equal to the number of nodes in the list, this returns \texttt{NULL} (See Also: \texttt{DOMNodeList Subprograms}):

\begin{verbatim}
DBMS_XMLDOM.ITEM(
    nl     IN     DOMNODELIST,
    index IN     NUMBER)
RETURN DOMNODE;
\end{verbatim}

Parameters

Table 200-95  ITEM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnm</td>
<td>DOMNAMEDNODEMAP</td>
</tr>
<tr>
<td>index</td>
<td>The index in the node map at which the item is to be retrieved</td>
</tr>
<tr>
<td>nl</td>
<td>DOMNODELIST</td>
</tr>
<tr>
<td>index</td>
<td>The index in the NodeList used to retrieve the item</td>
</tr>
</tbody>
</table>
200.7.75 MAKEATTR Function

This function casts a specified DOMNODE to a DOMATTR, and returns the DOMATTR.

See Also:
DOMNode Subprograms

Syntax

```
DBMS_XMLDOM.MAKEATTR(
    n       IN     DOMNODE)
RETURN DOMATTR;
```

Parameters

Table 200-96 MAKEATTR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

200.7.76 MAKECDATASECTION Function

This function casts a specified DOMNODE to a DOMCDATASECTION.

See Also:
DOMNode Subprograms

Syntax

```
DBMS_XMLDOM.MAKECDATASECTION(
    n       IN     DOMNODE)
RETURN DOMCDATASECTION;
```

Parameters

Table 200-97 MAKECDATASECTION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>
200.7.77 MAKECHARACTERDATA Function

This function casts a specified DOMNODE to a DOMCHARACTERDATA, and returns the DOMCHARACTERDATA.

Syntax

```
DBMS_XMLDOM.MAKECHARACTERDATA(
    n       IN     DOMNode)
RETURN DOMCharacterData;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

200.7.78 MAKECOMMENT Function

This function casts a specified DOMNODE to a DOMCOMMENT, and returns the DOMCOMMENT.

Syntax

```
DBMS_XMLDOM.MAKECOMMENT(
    n       IN     DOMNode)
RETURN DOMCOMMENT;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>
200.7.79 MAKEDOCUMENT Function

This function casts a specified DOMNODE to a DOMDOCUMENT, and returns the DOMDOCUMENT.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKEDOCUMENT(
  n    IN     DOMNODE)
RETURN DOMDocument;

Parameters

Table 200-100  MAKEDOCUMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

200.7.80 MAKEDOCUMENTFRAGMENT Function

This function casts a specified DOMNODE to a DOMDOCUMENTFRAGMENT, and returns the DOMDOCUMENTFRAGMENT.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKEDOCUMENTFRAGMENT(
  n    IN     DOMNODE)
RETURN DOMDOCUMENTFRAGMENT;

Parameters

Table 200-101  MAKEDOCUMENTFRAGMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>
200.7.81 MAKEDOCUMENTTYPE Function

This function casts a specified DOMNODE to a DOMDOCUMENTTYPE and returns the DOMDOCUMENTTYPE.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKEDOCUMENTTYPE(
  n       IN     DOMNODE)
RETURN DOMDOCUMENTTYPE;

Parameters

Table 200-102  MAKEDOCUMENTTYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

200.7.82 MAKEELEMENT Function

This function casts a specified DOMNODE to a DOMELEMENT, and returns the DOMELEMENT.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKEELEMENT(
  n       IN     DOMNODE)
RETURN DOMELEMENT;

Parameters

Table 200-103  MAKEELEMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>
200.7.83 MAKEENTITY Function

This function casts a specified DOMNODE to a DOMENTITY, and returns the DOMENTITY.

See Also:

DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKEENTITY(
    n       IN     DOMNODE)
RETURN DOMENTITY;

Parameters

Table 200-104 MAKEENTITY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

200.7.84 MAKEENTITYREFERENCE Function

This function casts a specified DOMNODE to a DOMENTITYREFERENCE, and returns the DOMENTITYREFERENCE.

See Also:

DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKEENTITYREFERENCE(
    n       IN     DOMNODE)
RETURN DOMENTITYREFERENCE;

Parameters

Table 200-105 MAKEENTITYREFERENCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>
200.7.85 MAKE NODE Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

Syntax

Casts specified DOMATTR to a DOMNODE, and returns the DOMNODE (See Also: DOMAttr Subprograms):

DBMS_XMLDOM.MAKENODE(
    a  IN  DOMATTR)
RETURN DOMNODE;

Casts the DOMCDATASECTION to a DOMNODE, and returns that DOMNODE (See Also: DOMCDataSection Subprograms):

DBMS_XMLDOM.MAKENODE(
    cds  IN  DOMCDATASECTION)
RETURN DOMNODE;

Casts the specified DOMCHARACTERDATA as a DOMNODE, and returns that DOMNODE (See Also: DOMCharacterData Subprograms):

DBMS_XMLDOM.MAKENODE(
    cd  IN  DOMCHARACTERDATA)
RETURN DOMNODE;

Casts the specified DOMCOMMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMComment Subprograms):

DBMS_XMLDOM.MAKENODE(
    com  IN  DOMCOMMENT)
RETURN DOMNODE;

Casts the DOMDOCUMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMDocument Subprograms):

DBMS_XMLDOM.MAKENODE(
    doc  IN  DOMDOCUMENT)
RETURN DOMNODE;

Casts the specified DOMDOCUMENTFRAGMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMDocumentFragment Subprograms):

DBMS_XMLDOM.MAKENODE(
    df  IN  DOMDOCUMENTFRAGMENT)
RETURN DOMNode;

Casts the specified DOMDOCUMENTTYPE to a DOMNODE, and returns that DOMNODE (See Also: DOMDocumentType Subprograms):

DBMS_XMLDOM.MAKENODE(
    dt  IN  DOMDOCUMENTTYPE)
RETURN DOMNODE;

Casts the specified DOMELEMENT to a DOMNODE, and returns that DOMNODE (See Also: DOMElement Subprograms):
DBMS_XMLDOM.MAKENODE(
   elem       IN     DOMELEMENT)
RETURN DOMNODE;

Casts specified DOMENTITY to a DOMNODE, and returns that DOMNODE (See Also: DOMEntity Subprograms):

DBMS_XMLDOM.MAKENODE(
   ent       IN     DOMENTITY)
RETURN DOMNODE;

Casts the DOMENTITYREFERENCE to a DOMNODE, and returns that DOMNODE (See Also: DOMEntityReference Subprograms):

DBMS_XMLDOM.MAKENODE(
   eref       IN     DOMENTITYREFERENCE)
RETURN DOMNODE;

Casts the DOMNOTATION to a DOMNODE, and returns that DOMNODE (See Also: DOMNotation Subprograms):

DBMS_XMLDOM.MAKENODE(
   n       IN     DOMNOTATION)
RETURN DOMNODE;

Casts the DOMPROCESSINGINSTRUCTION to a DOMNODE, and returns the DOMNODE (See Also: DOMProcessingInstruction Subprograms):

DBMS_XMLDOM.MAKENODE(
   pi       IN     DOMPROCESSINGINSTRUCTION)
RETURN DOMNODE;

Casts the DOMTEXT to a DOMNODE, and returns that DOMNODE (See Also: DOMText Subprograms):

DBMS_XMLDOM.MAKENODE(
   t       IN     DOMTEXT)
RETURN DOMNODE;

Parameters

Table 200-106  MAKENODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>DOMATTR to cast</td>
</tr>
<tr>
<td>cds</td>
<td>DOMCDATASECTION to cast</td>
</tr>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA to cast</td>
</tr>
<tr>
<td>com</td>
<td>DOMCOMMENT to cast</td>
</tr>
<tr>
<td>doc</td>
<td>DOMDOCUMENT to cast</td>
</tr>
<tr>
<td>df</td>
<td>DOMDOCUMENTFRAGMENT to cast</td>
</tr>
<tr>
<td>dt</td>
<td>DOMDOCUMENTTYPE to cast</td>
</tr>
<tr>
<td>elem</td>
<td>DOMELEMENT to cast</td>
</tr>
<tr>
<td>ent</td>
<td>DOMENTITY to cast</td>
</tr>
<tr>
<td>eref</td>
<td>DOMENTITYREFERENCE to cast</td>
</tr>
</tbody>
</table>
Table 200-106  (Cont.) MAKENODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNOTATION to cast</td>
</tr>
<tr>
<td>pi</td>
<td>DOMPROCESSINGINSTRUCTION to cast</td>
</tr>
<tr>
<td>t</td>
<td>DOMTEXT to cast</td>
</tr>
</tbody>
</table>

200.7.86 MAKENOTATION Function

This function casts a specified DOMNODE to a DOMNOTATION, and returns the DOMNOTATION.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKENOTATION(
   n       IN     DOMNODE)
RETURN DOMNOTATION;

Parameters

Table 200-107  MAKENOTATION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

200.7.87 MAKEPROCESSINGINSTRUCTION Function

This function casts a specified DOMNODE to a DOMPROCESSINGINSTRUCTION, and returns the Domprocessinginstruction.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.MAKEPROCESSINGINSTRUCTION(
   n       IN     DOMNODE)
RETURN DOMPROCESSINGINSTRUCTION;
### Parameters

#### Table 200-108 MAKEPROCESSINGINSTRUCTION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

### 200.7.88 MAKETEXT Function

This function casts a specified DOMNODE to a DOMTEXT, and returns the DOMTEXT.

**See Also:**

- DOMNode Subprograms

#### Syntax

```sql
DBMS_XMLDOM.MAKETEXT(
    n           IN     DOMNODE)
RETURN DOMTEXT;
```

### Parameters

#### Table 200-109 MAKETEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE to cast</td>
</tr>
</tbody>
</table>

### 200.7.89 NEWDOMDOCUMENT Functions

This function returns a new DOMDOCUMENT instance.

**See Also:**

- DOMDocument Subprograms

#### Syntax

**Returns a new DOMDOCUMENT instance:**

```sql
DBMS_XMLDOM.NEWDOMDOCUMENT
RETURN DOMDOCUMENT;
```

**Returns a new DOMDOCUMENT instance created from the specified XMLType object:**

```sql
```
DBMS_XMLDOM.NEWDOMDOCUMENT(
    xmldoc    IN SYS.XMLTYPE)
RETURN DOMDOCUMENT;

Returns a new DOMDOCUMENT instance created from the specified CLOB:

DBMS_XMLDOM.NEWDOMDOCUMENT(
    cl       IN    CLOB)
RETURN DOMDOCUMENT;

Parameters

Table 200-110  NEWDOMDOCUMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmldoc</td>
<td>XMLType source for the DOMDOCUMENT</td>
</tr>
<tr>
<td>cl</td>
<td>CLOB source for the DOMDOCUMENT</td>
</tr>
</tbody>
</table>

200.7.90 NORMALIZE Procedure

This procedure normalizes the text children of the DOMELEMENT.

See Also:

DOMElement Subprograms

Syntax

```
DBMS_XMLDOM.NORMALIZE(
    elem       IN     DOMELEMENT);
```

Parameters

Table 200-111  NORMALIZE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
</tbody>
</table>

200.7.91 REMOVEATTRIBUTE Procedures

This procedure removes an attribute from the DOMELEMENT by name.

See Also:

DOMElement Subprograms
Syntax

Removes the value of a DOMELEMENT's attribute by name:

```
DBMS_XMLDOM.REMOVEATTRIBUTE(
    elem IN DOMELEMENT,
    name IN VARCHAR2);
```

Removes the value of a DOMELEMENT's attribute by name and namespace URI.

```
DBMS_XMLDOM.REMOVEATTRIBUTE(
    elem IN DOMELEMENT,
    name IN VARCHAR2,
    ns IN VARCHAR2);
```

Parameters

Table 200-112 REMOVEATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>name</td>
<td>Attribute name</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

200.7.92 REMOVEATTRIBUTENODE Function

This function removes the specified attribute node from the DOMELEMENT. The method returns the removed node.

See Also:

DOMElement Subprograms

Syntax

```
DBMS_XMLDOM.REMOVEATTRIBUTENODE(
    elem IN DOMELEMENT,
    oldAttr IN DOMATTR)
RETURN DOMAttr;
```

Parameters

Table 200-113 REMOVEATTRIBUTENODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>oldAttr</td>
<td>The old DOMATTR</td>
</tr>
</tbody>
</table>
200.7.93 REMOVECHILD Function

This function removes the child node indicated by oldchild from the list of children, and returns it.

See Also:
DOMNode Subprograms

Syntax

DBMS_XMLDOM.REMOVECHILD(
    n           IN     DOMNode,
    oldchild    IN     DOMNode)
RETURN DOMNODE;

Parameters

Table 200-114  REMOVECHILD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>oldChild</td>
<td>The child of the node n to be removed</td>
</tr>
</tbody>
</table>

200.7.94 REMOVENAMEDITEM Function

This function removes a node, specified by name, from the map and returns this node.

When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

See Also:
DOMNamedNodeMap Subprograms

Syntax

Removes a node specified by name:

DBMS_XMLDOM.REMOVENAMEDITEM(
    nnm        IN     DOMNamedNodeMap,
    name       IN     VARCHAR2)
RETURN DOMNode;

Removes a node specified by name and namespace URI:

DBMS_XMLDOM.REMOVENAMEDITEM(
    nnm        IN     DOMNamedNodeMap,
name IN VARCHAR2,
ns IN VARCHAR2)
RETURN DOMNode;

Parameters

Table 200-115 REMOVENAMEDITEM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmm</td>
<td>DOMNamedNodeMap</td>
</tr>
<tr>
<td>name</td>
<td>The name of the item to be removed from the map</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

200.7.95 REPLACECHILD Function

This function replaces the child node oldchild with newchild in the list of children, and returns the oldchild node.

If newchild is a DocumentFragment object, oldchild is replaced by all of the DocumentFragment children, which are inserted in the same order. If the newchild is already in the tree, it is first removed.

See Also:

DOMNode Subprograms

Syntax

DBMS_XMLDOM.REPLACECHILD(
  n IN DOMNode,
  newchild IN DOMNode,
  oldchild IN DOMNode)
RETURN DOMNode;

Parameters

Table 200-116 REPLACECHILD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNode</td>
</tr>
<tr>
<td>newchild</td>
<td>The new child which is to replace the old child</td>
</tr>
<tr>
<td>oldchild</td>
<td>The child of the node n which is to be replaced</td>
</tr>
</tbody>
</table>
200.7.96 REPLACEDATA Procedure

This procedure changes a range of characters in the node. Upon success, data and length reflect the change.

See Also:
DOMCharacterData Subprograms

Syntax

```sql
DBMS_XMLDOM.REPLACEDATA(
    cd        IN     DOMCHARACTERDATA,
    offset    IN     NUMBER,
    cnt       IN     NUMBER,
    arg       IN     VARCHAR2);
```

Parameters

Table 200-117  REPLACEDATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>offset</td>
<td>The offset at which to replace</td>
</tr>
<tr>
<td>cnt</td>
<td>The number of characters to replace</td>
</tr>
<tr>
<td>arg</td>
<td>The value to replace with</td>
</tr>
</tbody>
</table>

200.7.97 RESOLVENAMESPACEPREFIX Function

This function resolves the specified namespace prefix, and returns the resolved namespace.

See Also:
DOMElement Subprograms

Syntax

```sql
DBMS_XMLDOM.RESOLVENAMESPACEPREFIX(
    elem       IN     DOMELEMENT,
    prefix     IN     VARCHAR2)
RETURN VARCHAR2;
```
Parameters

Table 200-118  RESOLVENAMESPACEPREFIX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>prefix</td>
<td>Namespace prefix</td>
</tr>
</tbody>
</table>

200.7.98 SETATTRIBUTE Procedures

This procedure sets the value of a DOMELEMENT's attribute by name.

See Also:
DOMElement Subprograms

Syntax

Sets the value of a DOMELEMENT's attribute by name:

```sql
DBMS_XMLDOM.SETATTRIBUTE(
    elem  IN  DOMELEMENT,
    name  IN  VARCHAR2,
    newvalue IN  VARCHAR2);
```

Sets the value of a DOMElement's attribute by name and namespace URI:

```sql
DBMS_XMLDOM.SETATTRIBUTE(
    elem  IN  DOMELEMENT,
    name  IN  VARCHAR2,
    newvalue IN  VARCHAR2,
    ns     IN  VARCHAR2);
```

Parameters

Table 200-119  SETATTRIBUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>name</td>
<td>Attribute name</td>
</tr>
<tr>
<td>newvalue</td>
<td>Attribute value</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>
200.7.99 SETATTRIBUTENODE Functions

This function adds a new attribute node to the DOMELEMENT.

See Also:
DOMElement Subprograms

Syntax

Adds a new attribute node to the DOMELEMENT:

```sql
DBMS_XMLDOM.SETATTRIBUTENODE(
    elem      IN  DOMELEMENT,
    newAttr   IN  DOMATTR)
RETURN DOMATTR;
```

Adds a new attribute node to the DOMElement; namespace enabled:

```sql
DBMS_XMLDOM.SETATTRIBUTENODE(
    elem      IN  DOMELEMENT,
    newAttr   IN  DOMATTR,
    ns        IN  VARCHAR2)
RETURN DOMATTR;
```

Parameters

Table 200-120  SETATTRIBUTENODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>The DOMELEMENT</td>
</tr>
<tr>
<td>newAttr</td>
<td>The new DOMATTR</td>
</tr>
<tr>
<td>ns</td>
<td>The namespace</td>
</tr>
</tbody>
</table>

200.7.100 SETCHARSET Procedure

This function sets the character set of the DOM document.

See Also:
DOMDocument Subprograms

Syntax

```sql
DBMS_XMLDOM.SETCHARSET(
    doc      IN    DOMDocument,
    charset  IN    VARCHAR2);
```
Parameters

Table 200-121 SETCHARSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOM document</td>
</tr>
<tr>
<td>charset</td>
<td>Characterset</td>
</tr>
</tbody>
</table>

Usage Notes

This is used for WRITETOFILE Procedures if not explicitly specified at that time.

200.7.101 SETDATA Procedures

This overloaded procedure sets character data or DOMPROCESSINGINSTRUCTION content data. The specific functionality is described in the syntax declarations.

Syntax

Sets the character data of the node that implements this interface (See Also: DOMCharacterData Subprograms):

```sql
DBMS_XMLDOM.SETDATA(
    cd IN DOMCHARACTERDATA,
    data IN VARCHAR2);
```

Sets the content data of the DOMPROCESSINGINSTRUCTION (See Also: DOMProcessingInstruction Subprograms):

```sql
DBMS_XMLDOM.SETDATA(
    pi IN DOMPROCESSINGINSTRUCTION,
    data IN VARCHAR2);
```

Parameters

Table 200-122 SETDATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>data</td>
<td>The data to which the node is set</td>
</tr>
<tr>
<td>pi</td>
<td>DOMPROCESSINGINSTRUCTION</td>
</tr>
<tr>
<td>data</td>
<td>New processing instruction content data</td>
</tr>
</tbody>
</table>

200.7.102 SETDOCTYPE Procedure

Given a DOM document, this procedure creates a new DTD with the specified name, system id and public id and sets it in the document.

This DTD can later be retrieved using the GETDOCTYPE Function.
Syntax

```sql
DBMS_XMLDOM.SETDOCTYPE(
    doc     IN   DOMDocument,
    name    IN   VARCHAR2,
    sysid   IN   VARCHAR2,
    pubid   IN   VARCHAR2);
```

Parameters

Table 200-123  SETDOCTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>The document whose DTD has to be set</td>
</tr>
<tr>
<td>name</td>
<td>The name that the doctype needs to be initialized with</td>
</tr>
<tr>
<td>sysid</td>
<td>The system ID that the doctype needs to be initialized with</td>
</tr>
<tr>
<td>pubid</td>
<td>The public ID that the doctype needs to be initialized with</td>
</tr>
</tbody>
</table>

200.7.103 SETNAMEDITEM Function

This function adds a node using its `nodeName` attribute.

If a node with that name is already present in this map, it is replaced by the new one. The old node is returned on replacement; if no replacement is made, `NULL` is returned.

As the `nodeName` attribute is used to derive the name under which the node must be stored, multiple nodes of certain types, those that have a "special" string value, cannot be stored because the names would clash. This is seen as preferable to allowing nodes to be aliased.

See Also:

DOMNamedNodeMap Subprograms

Syntax

Adds a node using its `nodeName` attribute:

```sql
DBMS_XMLDOM.SETNAMEDITEM(
    nnm     IN     DOMNAMEDNODEMAP,
    arg     IN     DOMNODE)
RETURN DOMNode;
```

Adds a node using its `nodeName` attribute and namespace URI:

```sql
DBMS_XMLDOM.SETNAMEDITEM(
    nnm     IN     DOMNAMEDNODEMAP,
    arg     IN     DOMNODE,
    ns      IN     VARCHAR2)
RETURN DOMNode;
```
Parameters

Table 200-124  SETNAMEDITEM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmm</td>
<td>DOMNAMEDNODEMAP</td>
</tr>
<tr>
<td>arg</td>
<td>The Node to be added using its NodeName attribute</td>
</tr>
<tr>
<td>ns</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

200.7.104 SETNODEVALUE Procedure

This procedure sets the value of this node, depending on its type. When it is defined to be NULL, setting it has no effect.

See Also:
DOMNode Subprograms

Syntax

```sql
DBMS_XMLDOM.SETNODEVALUE(
    n IN DOMNODE,
    nodeValue IN VARCHAR2);
```

Parameters

Table 200-125  SETNODEVALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNode</td>
</tr>
<tr>
<td>nodeValue</td>
<td>The value to which node is set</td>
</tr>
</tbody>
</table>

200.7.105 SETNODEVALUEASBINARYSTREAM Function & Procedure

The operation of these subprograms is described in the syntax section.

See Also:
DOMNode Subprograms
Syntax

This function returns an instance of the PL/SQL XMLBINARYOUTPUTSTREAM into which the caller can write the node value. The datatype of the node must be RAW or BLOB – if not, an exception is raised.

```
DBMS_XMLDOM.SETNODEVALUEASBINARYSTREAM (n IN DOMNODE)
RETURN SYS.UTL_BINARYOUTPUTSTREAM;
```

Using this procedure, the application passes in an implementation of sys.utl_BinaryInputStream from which XDB reads data to populate the node. The datatype of the node must be RAW or BLOB – if not an exception is raised.

```
DBMS_XMLDOM.SETNODEVALUEASBINARYSTREAM (n in DOMNODE,
value in SYS.UTL_BINARYINPUTSTREAM);
```

Parameters

Table 200-126  SETNODEVALUEASBINARYSTREAM Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>value</td>
<td>BINARYINPUTSTREAM</td>
</tr>
</tbody>
</table>

200.7.106 SETNODEVALUEASCHARACTERSTREAM Function & Procedure

The operation of these subprograms is described in the syntax section.

See Also:

- DOMNode Subprograms

Syntax

This function returns an instance of the PL/SQL XMLCHARACTEROUTPUTSTREAM type into which the caller can write the node value. The datatype of the node can be any valid XDB datatype. If the type is not character or CLOB, the character data written to the stream is converted to the node datatype. If the datatype of the node is character or CLOB, then the character data written to the stream is converted from PL/SQL session character set to the character set of the node.

```
DBMS_XMLDOM.SETNODEVALUEASCHARACTERSTREAM (n IN DOMNODE)
RETURN SYS.UTL_CHARACTEROUTPUTSTREAM;
```

Using this procedure, the application passes in an implementation of sys.utl_CharacterInputStream from which XDB reads to populate the node. The datatype of
the node may be any valid type supported by XDB. If a non-character datatype, the
class data read from the stream is converted to the datatype of the node. If the
datatype of the node is either character or CLOB, then no conversion occurs and the
class set of the node becomes the character set of the PL/SQL session.

```sql
DBMS_XMLDOM.SETNODEVALUEASCHARACTERSTREAM(
    n        IN   DOMNODE,
    value    IN   SYS.UTL_CHARACTERINPUTSTREAM);
```

**Parameters**

Table 200-127  SETNODEVALUEASCHARACTERSTREAM Function & Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>value</td>
<td>CHARACTERINPUTSTREAM</td>
</tr>
</tbody>
</table>

200.7.107 SETPREFIX Procedure

This procedure sets the namespace prefix for this node to the specified value.

> See Also:

DOMNode Subprograms

**Syntax**

```sql
DBMS_XMLDOM.SETPREFIX(
    n       IN     DOMNODE,
    prefix  IN     VARCHAR2);
```

**Parameters**

Table 200-128  SETPREFIX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>prefix</td>
<td>The value for the namespace prefix of the node</td>
</tr>
</tbody>
</table>

200.7.108 SETSTANDALONE Procedure

This procedure sets the standalone property of the DOMDOCUMENT.

> See Also:

DOMDocument Subprograms
Syntax

DBMS_XMLDOM.SETSTANDALONE(
    doc IN DOMDOCUMENT,
    newvalue IN VARCHAR2);

Parameters

Table 200-129  SETSTANDALONE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>newvalue</td>
<td>Value of the standalone property of the document</td>
</tr>
</tbody>
</table>

200.7.109 SETVALUE Procedure

This procedure sets the value of the attribute.

See Also:

DOMAttr Subprograms

Syntax

DBMS_XMLDOM.SETVALUE(
    a IN DOMATTR,
    value IN VARCHAR2);

Parameters

Table 200-130  SETVALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>DOMATTR</td>
</tr>
<tr>
<td>value</td>
<td>The value to which to set the attribute</td>
</tr>
</tbody>
</table>

200.7.110 SETVERSION Procedure

This procedure sets the version of the DOMDOCUMENT.

See Also:

DOMDocument Subprograms
Syntax

DBMS_XMLDOM.SETVERSION(
    doc        IN     DOMDOCUMENT,
    version    IN     VARCHAR2);

Parameters

Table 200-131  SETVERSION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>version</td>
<td>The version of the document</td>
</tr>
</tbody>
</table>

200.7.111 SPLITTEXT Function

This function breaks this DOMTEXT node into two DOMTEXT nodes at the specified offset.

See Also:
- DBMS_XMLDOM DOMText Subprograms

Syntax

DBMS_XMLDOM.SPLITTEXT(
    t        IN     DOMTEXT,
    offset   IN     NUMBER)
RETURN DOMText;

Parameters

Table 200-132  SPLITTEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>DOMTEXT</td>
</tr>
<tr>
<td>offset</td>
<td>Offset at which to split</td>
</tr>
</tbody>
</table>

200.7.112 SUBSTRINGDATA Function

This function extracts a range of data from the node.

See Also:
- DOMCharacterData Subprograms
Syntax

```sql
DBMS_XMLDOM.SUBSTRINGDATA(
    cd        IN     DOMCHARACTERDATA,
    offset    IN     NUMBER,
    cnt       IN     NUMBER)
RETURN VARCHAR2;
```

Parameters

Table 200-133 SUBSTRINGDATA Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>DOMCHARACTERDATA</td>
</tr>
<tr>
<td>offset</td>
<td>The starting offset of the data from which to get the data</td>
</tr>
<tr>
<td>cnt</td>
<td>The number of characters (from the offset) of the data to get</td>
</tr>
</tbody>
</table>

200.7.113 USEBINARYSTREAM Function

This function returns `TRUE` if the datatype of the node is `RAW` or `BLOB`, so that the node value may be read or written using an `UTL_BINARYINPUTSTREAM` or `UTL_BINARYOUTPUTSTREAM`.

If a value of `FALSE` is returned, the node value may only be accessed through an `UTL_CHARACTERINPUTSTREAM` or `UTL_CHARACTEROUTPUTSTREAM`.

See Also:

DOMNode Subprograms

Syntax

```sql
DBMS_XMLDOM.USEBINARYSTREAM   (
    n        IN     DOMNODE)
RETURN BOOLEAN;
```

Parameters

Table 200-134 USEBINARYSTREAM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>

200.7.114 WRITETOBUFFER Procedures

`WRITETOBUFFER` is an overloaded procedure that writes an XML node, XML document, or a document fragment to a specified buffer.

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.
Syntax

Writes XML node to specified buffer using the database character set (See Also: DOMNode Subprograms):

```sql
DBMS_XMLDOM.WRITETOBUFFER(
  n IN DOMNODE,
  buffer IN OUT VARCHAR2);
```

Writes XML document to a specified buffer using database character set (See Also: DOMDocument Subprograms):

```sql
DBMS_XMLDOM.WRITETOBUFFER(
  doc IN DOMDOCUMENT,
  buffer IN OUT VARCHAR2);
```

Writes the contents of the specified document fragment into a buffer using the database character set (See Also: DOMDocumentFragment Subprograms):

```sql
DBMS_XMLDOM.WRITETOBUFFER(
  df IN DOMDOCUMENTFRAGMENT,
  buffer IN OUT VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer to which to write</td>
</tr>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>df</td>
<td>DOM document fragment</td>
</tr>
</tbody>
</table>

200.7.115 WRITETOCLOB Procedures

WRITETOCLOB is an overloaded procedure that writes an XML node or document to a specified CLOB.

The specific forms of functionality are described along with the syntax declarations.

Syntax

Writes XML node to specified CLOB using the database character set (See Also: DOMNode Subprograms):

```sql
DBMS_XMLDOM.WRITETOCLOB(
  n IN DOMNODE,
  cl IN OUT CLOB);
```

Writes XML document to a specified CLOB using database character set (See Also: DOMDocument Subprograms):

```sql
DBMS_XMLDOM.WRITETOCLOB(
  doc IN DOMDOCUMENT,
  cl IN OUT CLOB);
```
Parameters

Table 200-136  WRITETOCLOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
<tr>
<td>cl</td>
<td>CLOB to which to write</td>
</tr>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
</tbody>
</table>

200.7.116 WRITETOFIILE Procedures

This overloaded procedure writes an XML node or XML document to a specified node. The specific forms of functionality are described along with the syntax declarations.

Syntax

Writes XML node to specified file using the database character set (See Also: DOMNode Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
    n           IN      DOMNODE,
    fileName    IN      VARCHAR2);
```

Writes XML node to specified file using the specified character set, which is passed in as a separate parameter (See Also: DOMNode Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
    n           IN      DOMNODE,
    fileName    IN      VARCHAR2,
    charset     IN      VARCHAR2);
```

Writes an XML document to a specified file using database character set (See Also: DOMDocument Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
    doc         IN      DOMDOCUMENT,
    filename    IN      VARCHAR2);
```

Writes an XML document to a specified file using specified character set (See Also: DOMDocument Subprograms):

```
DBMS_XMLDOM.WRITETOFILE(
    doc         IN      DOMDOCUMENT,
    fileName    IN      VARCHAR2,
    charset     IN      VARCHAR2);
```

Parameters

Table 200-137  WRITETOFIILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNODE</td>
</tr>
</tbody>
</table>
Table 200-137  (Cont.) WRITETOFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>File to which to write. The filename should be in the format of database_directory_object_name/filename, for example mydir/filename (on windows, use \ instead of /).</td>
</tr>
<tr>
<td>charset</td>
<td>specified character set</td>
</tr>
<tr>
<td>doc</td>
<td>DOMDOCUMENT</td>
</tr>
<tr>
<td>charset</td>
<td>Character set</td>
</tr>
</tbody>
</table>
The DBMS_XMLGEN package converts the results of a SQL query to a canonical XML format. The package takes an arbitrary SQL query as input, converts it to XML format, and returns the result as a CLOB. This package is similar to the DBMS_XMLQUERY package, except that it is written in C and compiled into the kernel. This package can only be run on the database.

This chapter contains the following topic:

- Security Model
- Summary of DBMS_XMLGEN Subprograms

See Also:
Oracle XML DB Developer's Guide, for more information on XML support and on examples of using DBMS_XMLGEN

### 201.1 DBMS_XMLGEN Security Model

Owned by XDB, the DBMS_XMLGEN package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

### 201.2 Summary of DBMS_XMLGEN Subprograms

This table lists the DBMS_XMLGEN subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSECONTEXT Procedure</td>
<td>Closes the context and releases all resources</td>
</tr>
<tr>
<td>CONVERT Functions</td>
<td>Converts the XML into the escaped or unescaped XML equivalent</td>
</tr>
<tr>
<td>GETNUMROWSPROCESSED Function</td>
<td>Gets the number of SQL rows that were processed in the last call to GETXML Functions</td>
</tr>
<tr>
<td>GETXML Functions</td>
<td>Gets the XML document</td>
</tr>
<tr>
<td>GETXMLTYPE Functions</td>
<td>Gets the XML document and returns it as XMLType</td>
</tr>
<tr>
<td>NEWCONTEXT Functions</td>
<td>Creates a new context handle</td>
</tr>
<tr>
<td>NEWCONTEXTFROMHIERARCHY Function</td>
<td>Obtains a handle to use in the GETXML Functions and other functions to get a hierarchical XML with recursive elements from the result</td>
</tr>
<tr>
<td>RESTARTQUERY Procedure</td>
<td>Restarts the query to start fetching from the beginning</td>
</tr>
</tbody>
</table>
Table 201-1 (Cont.) Summary of DBMS(XmlGEN) Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETCONVERTSPECIALCHARS Procedure</td>
<td>Sets whether special characters such as $, which are non-XML characters, should be converted or not to their escaped representation.</td>
</tr>
<tr>
<td>SETMAXROWS Procedure</td>
<td>Sets the maximum number of rows to be fetched each time.</td>
</tr>
<tr>
<td>SETNULLHANDLING Procedure</td>
<td>Sets NULL handling options.</td>
</tr>
<tr>
<td>SETROWSETTAG Procedure</td>
<td>Sets the name of the element enclosing the entire result.</td>
</tr>
<tr>
<td>SETROWTAG Procedure</td>
<td>Sets the name of the element enclosing each row of the result.</td>
</tr>
<tr>
<td>SETSKIPROWS Procedure</td>
<td>Sets the number of rows to skip every time before generating the XML.</td>
</tr>
<tr>
<td>USEITEMTAGSFORCOLL Procedure</td>
<td>Forces the use of the collection column name appended with the tag _ITEM for collection elements.</td>
</tr>
<tr>
<td>USENULLATTRIBUTEINDICATOR Procedure</td>
<td>Specifies whether to use an XML attribute to indicate NULLness, or to do it by omitting the inclusion of the particular entity in the XML document.</td>
</tr>
</tbody>
</table>

201.2.1 CLOSECONTEXT Procedure

This procedure closes a given context and releases all resources associated with it, including the SQL cursor and bind and define buffers. After this call, the handle cannot be used for a subsequent function call.

**Syntax**

```sql
DBMS_XMLGEN.CLOSECONTEXT (
    ctx  IN ctxHandle);
```

**Parameters**

Table 201-2 CLOSECONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle to close.</td>
</tr>
</tbody>
</table>

201.2.2 CONVERT Functions

This function converts the XML data into the escaped or unescapes XML equivalent, and returns XML CLOB data in encoded or decoded format. There are several version of the function.

**Syntax**

Uses XMLDATA in string form (VARCHAR2):

```sql
DBMS_XMLGEN.CONVERT (  
    xmlData IN VARCHAR2,  
    flag    IN NUMBER := ENTITY_ENCODE)  
RETURN VARCHAR2;
```
Uses XMLDATA in CLOB form:

```sql
DBMS_XMLGEN.CONVERT (
    xmlData IN CLOB,
    flag    IN NUMBER := ENTITY_ENCODE)
RETURN CLOB;
```

### Parameters

**Table 201-3 CONVERT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlData</td>
<td>The XML CLOB data to be encoded or decoded.</td>
</tr>
<tr>
<td>flag</td>
<td>The flag setting; ENTITY_ENCODE (default) for encode, and ENTITY_DECODE for decode.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This function escapes the XML data if the ENTITY_ENCODE is specified. For example, the escaped form of the character `<` is `&lt;`. Unescaping is the reverse transformation.

### 201.2.3 GETNUMROWSPROCESSED Function

This function retrieves the number of SQL rows processed when generating the XML using the GETXML Functions call. This count does not include the number of rows skipped before generating the XML.

Note that GETXML Functions always generates an XML document, even if there are no rows present.

**Syntax**

```sql
DBMS_XMLGEN.GETNUMROWSPROCESSED (
    ctx     IN    ctxHandle)
RETURN NUMBER;
```

**Parameters**

**Table 201-4 GETNUMROWSPROCESSED Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle obtained from the NEWCONTEXT Functions call.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This function is used to determine the terminating condition if calling GETXML Functions in a loop.

**Related Topics**

- GETXML Functions
  This function gets the XML document. The function is overloaded.
201.2.4 GETXML Functions

This function gets the XML document. The function is overloaded.

Syntax

Gets the XML document by fetching the maximum number of rows specified. It appends the XML document to the CLOB passed in. Use this version of GETXML Functions to avoid any extra CLOB copies and to reuse the same CLOB for subsequent calls. Because of the CLOB reuse, this GETXML Functions call is potentially more efficient:

```
DBMS_XMLGEN.GETXML (  
    ctx          IN ctxHandle,  
    tmpclob      IN OUT NCOPY CLOB,  
    dtdOrSchema  IN number := NONE)  
RETURN BOOLEAN;
```

Generates the XML document and returns it as a temporary CLOB. The temporary CLOB obtained from this function must be freed using the DBMS_LOB.FREETEMPORARY call:

```
DBMS_XMLGEN.GETXML (  
    ctx          IN ctxHandle,  
    dtdOrSchema  IN number := NONE)  
RETURN CLOB;
```

Converts the results from the SQL query string to XML format, and returns the XML as a temporary CLOB, which must be subsequently freed using the DBMS_LOB.FREETEMPORARY call:

```
DBMS_XMLGEN.GETXML (  
    sqlQuery     IN VARCHAR2,  
    dtdOrSchema  IN number := NONE)  
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Table 201-5</th>
<th>GETXML Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>ctx</td>
<td>The context handle obtained from the newContext call.</td>
</tr>
<tr>
<td>tmpclob</td>
<td>The CLOB to which the XML document is appended.</td>
</tr>
<tr>
<td>sqlQuery</td>
<td>The SQL query string.</td>
</tr>
<tr>
<td>dtdOrSchema</td>
<td>Generate a DTD or a schema? Only NONE is supported.</td>
</tr>
</tbody>
</table>

Usage Notes

When the rows indicated by the SETSKIPROWS Procedure call are skipped, the maximum number of rows as specified by the SETMAXROWS Procedure call (or the entire result if not specified) is fetched and converted to XML. Use the GETNUMROWSPROCESSED Function to check if any rows were retrieved.
201.2.5 GETXMLTYPE Functions

This function gets the XML document and returns it as an XMLTYPE. XMLTYPE operations can be performed on the results. This function is overloaded.

Syntax

Generates the XML document and returns it as a sys.XMLType:

```sql
DBMS_XMLGEN.GETXMLTYPE (
    ctx           IN ctxhandle,
    dtdOrSchema   IN number := NONE)
RETURN sys.XMLType;
```

Converts the results from the SQL query string to XML format, and returns the XML as a sys.XMLType:

```sql
DBMS_XMLGEN.GETXMLTYPE (
    sqlQuery     IN VARCHAR2,
    dtdOrSchema  IN number := NONE)
RETURN sys.XMLType
```

Parameters

Table 201-6  GETXMLTYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle obtained from the newContext call.</td>
</tr>
<tr>
<td>sqlQuery</td>
<td>The SQL query string.</td>
</tr>
<tr>
<td>dtdOrSchema</td>
<td>Generate a DTD or a schema? Only NONE is supported.</td>
</tr>
</tbody>
</table>

201.2.6 NEWCONTEXT Functions

This function generates and returns a new context handle.

This context handle is used in GETXML Functions and other functions to get XML back from the result. There are several version of the function.

Syntax

Generates a new context handle from a query:

```sql
DBMS_XMLGEN.NEWCONTEXT (      
    query     IN VARCHAR2)   
RETURN ctxHandle;
```

Generates a new context handle from a query string in the form of a PL/SQL ref cursor:

```sql
DBMS_XMLGEN.NEWCONTEXT (      
    queryString  IN SYS_REFCURSOR) 
RETURN ctxHandle;
```
Parameters

Table 201-7  NEWCONTEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>The query, in the form of a VARCHAR, the result of which must be converted to XML.</td>
</tr>
<tr>
<td>queryString</td>
<td>The query string in the form of a PL/SQL ref cursor, the result of which must be converted to XML.</td>
</tr>
</tbody>
</table>

201.2.7 NEWCONTEXTFROMHIERARCHY Function

This function obtains a handle to use in the GETXML Functions and other functions to get a hierarchical XML with recursive elements from the result.

Syntax

```sql
DBMS_XMLGEN.NEWCONTEXTFROMHIERARCHY (queryString IN VARCHAR2) RETURN ctxHandle;
```

Parameters

Table 201-8  NEWCONTEXTFROMHIERARCHY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryString</td>
<td>The query string, the result of which must be converted to XML. The query is a hierarchical query typically formed using a CONNECT BY clause, and the result must have the same property as the result set generated by a CONNECT BY query. The result set must have only two columns, the level number and an XML value. The level number is used to determine the hierarchical position of the XML value within the result XML document.</td>
</tr>
</tbody>
</table>

Related Topics

- **GETXML Functions**
  This function gets the XML document. The function is overloaded.

201.2.8 RESTARTQUERY Procedure

This procedure restarts the query and generates the XML from the first row.

It can be used to start executing the query again, without having to create a new context.

Syntax

```sql
DBMS_XMLGEN.RESTARTQUERY (ctx IN ctxHandle);
```
Parameters

Table 201-9  RESTARTQUERY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle corresponding to the current query.</td>
</tr>
</tbody>
</table>

201.2.9 SETCONVERTSPECIALCHARS Procedure

This procedure sets whether or not special characters in the XML data must be converted into their escaped XML equivalent. For example, the `<` sign is converted to `&lt;`.

The default is to perform conversions.

This function improves performance of XML processing when the input data cannot contain any special characters such as `<`, `>`, `"`, `'`, which must be escaped. It is expensive to scan the character data to replace the special characters, particularly if it involves a lot of data.

Syntax

```sql
DBMS_XMLGEN.SETCONVERTSPECIALCHARS (  
  ctx   IN ctxHandle,  
  conv  IN BOOLEAN);
```

Parameters

Table 201-10  SETCONVERTSPECIALCHARS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle obtained from one of the NEWCONTEXT Functions call.</td>
</tr>
<tr>
<td>conv</td>
<td>TRUE indicates that conversion is needed.</td>
</tr>
</tbody>
</table>

201.2.10 SETMAXROWS Procedure

This procedure sets the maximum number of rows to fetch from the SQL query result for every invocation of the GETXML Functions call.

It is used when generating paginated results. For example, when generating a page of XML or HTML data, restrict the number of rows converted to XML or HTML by setting the `maxRows` parameter.

Syntax

```sql
DBMS_XMLGEN.SETMAXROWS (  
  ctx   IN ctxHandle,  
  maxRows  IN NUMBER);
```
Parameters

**Table 201-11  SETMAXROWS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle corresponding to the query executed.</td>
</tr>
<tr>
<td>maxRows</td>
<td>The maximum number of rows to get for each call to GETXML Functions</td>
</tr>
</tbody>
</table>

Related Topics

- GETXML Functions
  This function gets the XML document. The function is overloaded.

### 201.2.11 SETNULLHANDLING Procedure

This procedure sets NULL handling options, handled through the flag parameter setting.

**Syntax**

```sql
DBMS_XMLGEN.SETNULLHANDLING(
  ctx  IN ctx,
  flag IN NUMBER);
```

**Parameters**

**Table 201-12  SETNULLHANDLING Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle corresponding to the query executed.</td>
</tr>
<tr>
<td>flag</td>
<td>The NULL handling option set.</td>
</tr>
</tbody>
</table>
  - DROP_NULLS CONSTANT NUMBER:= 0; (Default) Leaves out the tag for NULL elements.
  - NULL_ATTR CONSTANT NUMBER:= 1; Sets xsi:nil="true".
  - EMPTY_TAG CONSTANT NUMBER:= 2; Sets, for example, <foo/>.

### 201.2.12 SETROWSETTAG Procedure

This procedure sets the name of the root element of the document. The default name is ROWSET.

**Syntax**

```sql
DBMS_XMLGEN.SETROWSETTAG ( 
  ctx IN ctxHandle,
  rowSetTagName IN VARCHAR2);
```
Parameters

Table 201-13  SETROWSETTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle obtained from the NEWCONTEXT Functions call.</td>
</tr>
<tr>
<td>rowSetTagName</td>
<td>The name of the document element. Passing NULL indicates that you do not want the ROWSET element present.</td>
</tr>
</tbody>
</table>

Usage Notes

The user can set the rowSetTag to NULL to suppress the printing of this element. However, an error is produced if both the row and the rowset are NULL and there is more than one column or row in the output. This is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

201.2.13 SETROWTAG Procedure

This procedure sets the name of the element separating all the rows. The default name is ROW.

Syntax

```sql
DBMS_XMLGEN.SETROWTAG (  
    ctx IN ctxHandle,  
    rowTagName IN VARCHAR2);  
```

Parameters

Table 201-14  SETROWTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle obtained from the NEWCONTEXT Functions call.</td>
</tr>
<tr>
<td>rowTagName</td>
<td>The name of the ROW element. Passing NULL indicates that you do not want the ROW element present.</td>
</tr>
</tbody>
</table>

Usage Notes

The user can set the name of the element to NULL to suppress the ROW element itself. However, an error is produced if both the row and the rowset are NULL and there is more than one column or row in the output. This is because the generated XML would not have a top-level enclosing tag, and so would be invalid.
201.2.14 SETSKIPROWS Procedure

This procedure skips a given number of rows before generating the XML output for every call to the GETXML Functions. It is used when generating paginated results for stateless Web pages using this utility.

For example, when generating the first page of XML or HTML data, set skiprows to zero. For the next set, set the skiprows to the number of rows obtained in the first case. See GETNUMROWSPROCESSED Function.

Syntax

```
DBMS_XMLGEN.SETSKIPROWS (  
  ctx       IN ctxHandle,  
  skipRows  IN NUMBER);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle corresponding to the query executed.</td>
</tr>
<tr>
<td>skipRows</td>
<td>The number of rows to skip for each call to getXML.</td>
</tr>
</tbody>
</table>

Related Topics

- GETXML Functions
  This function gets the XML document. The function is overloaded.

201.2.15 USEITEMTAGSFORCOLL Procedure

This procedure overrides the default name of the collection elements. The default name for collection elements is the type name itself.

Syntax

```
DBMS_XMLGEN.USEITEMTAGSFORCOLL (  
  ctx      IN ctxHandle);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>The context handle.</td>
</tr>
</tbody>
</table>

Usage Notes

Using this procedure, you can override the default to use the name of the column with the _ITEM tag appended to it. If there is a collection of NUMBER, the default tag name for the collection elements is NUMBER.
201.2.16 USENULLATTRIBUTEINDICATOR Procedure

This procedure specifies whether to use an XML attribute to indicate NULL, or to do it by omitting the inclusion of the particular entity in the XML document.

It is used as a shortcut for the SETNULLHANDLING Procedure.

Syntax

```sql
DBMS_XMLGEN.USENULLATTRIBUTEINDICATOR(
    ctx       IN   ctxType,
    attrind   IN   BOOLEAN := TRUE);
```

Parameters

Table 201-17 USENULLATTRIBUTEINDICATOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>Context handle.</td>
</tr>
<tr>
<td>attrind</td>
<td>Use attribute to indicate NULL?</td>
</tr>
</tbody>
</table>

The **DBMS_XMLINDEX** package provides an interface to implement asynchronous indexing.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_XMLINDEX Subprograms

### See Also:

*Oracle XML DB Developer's Guide* for more information about “XMLIndex”

## 202.1 DBMS_XMLINDEX Overview

DBMS_XMLINDEX provides a mechanism for asynchronous index maintenance.

### Asynchronous Index Maintenance

The basic XMLIndex is maintained on every DML operation. However, given the computing costs, in many cases the availability of stale result is adequate. In such situations, it is desirable to defer index updates to a more convenient time, for example when the load on the database is low. **DBMS_XMLINDEX** provides this mechanism.

## 202.2 DBMS_XMLINDEX Security Model

Owned by XDB, the **DBMS_XMLINDEX** package must be created by SYS or XDB. The **EXECUTE** privilege is granted to **PUBLIC**.

Subprograms in this package are executed using the privileges of the current user.

## 202.3 Summary of DBMS_XMLINDEX Subprograms

This table lists the **DBMS_XMLINDEX** subprograms and briefly describes them.

### Table 202-1  DBMS_XMLINDEX Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CREATEDATEINDEX Procedure</strong></td>
<td>Creates a secondary index for date values in the <strong>VALUE</strong> column of a <strong>PATH TABLE</strong> which is the storage table of an XMLIndex</td>
</tr>
</tbody>
</table>
Table 202-1  (Cont.) DBMS_XMLINDEX Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATENUMBERINDEX Procedure</td>
<td>Creates a secondary index for number values in the VALUE column of a PATH TABLE which is the storage table of an XMLIndex.</td>
</tr>
<tr>
<td>DROPPARAMETER Procedure</td>
<td>Drops the XMLIndex parameter string that is associated with a given parameter identifier.</td>
</tr>
<tr>
<td>MODIFYPARAMETER Procedure</td>
<td>Modifies the XMLIndex parameter string that is associated with a given parameter name.</td>
</tr>
<tr>
<td>PROCESS_PENDING Procedure</td>
<td>Processes pending rows for a NONBLOCKING ALTER INDEX OPERATION on an XMLIndex.</td>
</tr>
<tr>
<td>REGISTERPARAMETER Procedure</td>
<td>Registers a parameter string and XMLIndex parameter string pair in XDB.</td>
</tr>
<tr>
<td>SYNCINDEX Procedure</td>
<td>Synchronizes the index manually.</td>
</tr>
</tbody>
</table>

202.3.1 CREATEDATEINDEX Procedure

This procedure creates a secondary index for date values in the VALUE column of a PATH TABLE which is the storage table of an XMLIndex. The second form of the procedure allows for the date_index_clause to be set to an empty string.

Syntax

```sql
DBMS_XMLINDEX.CREATEDATEINDEX  (  
  xml_index_schema   IN   VARCHAR2,  
  xml_index_name     IN   VARCHAR2,  
  date_index_name    IN   VARCHAR2,  
  xmltypename        IN   VARCHAR2,  
  date_index_clause  IN   VARCHAR2);  
```

```sql
DBMS_XMLINDEX.CREATEDATEINDEX  (  
  xml_index_schema   IN   VARCHAR2,  
  xml_index_name     IN   VARCHAR2,  
  date_index_name    IN   VARCHAR2,  
  xmltypename        IN   VARCHAR2);  
```

Parameters

Table 202-2  CREATEDATEINDEX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml_index_schema</td>
<td>Name of the owner of the XMLIndex</td>
</tr>
<tr>
<td>xml_index_name</td>
<td>Name of the XMLIndex</td>
</tr>
<tr>
<td>date_index_name</td>
<td>Name of the secondary index to be created for date values in the VALUE column of the PATH TABLE of XMLIndex named xml_index_name and owned by xml_index_schema</td>
</tr>
</tbody>
</table>
Table 202-2  (Cont.) CREATEDATEINDEX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmltypename</td>
<td>The type to which values in the VALUE column of the path table are to be cast. Acceptable values are the following strings: DATETIME, TIME, DATE, GDAY, GMONTH, GYEAR, GYEARMONTH, GMONTHDAY.</td>
</tr>
<tr>
<td>date_index_clause</td>
<td>Storage clause to be applied to the date index during its creation. This is a string argument appended to the CREATE INDEX statement for creating the date index</td>
</tr>
</tbody>
</table>

202.3.2 CREATENUMBERINDEX Procedure

This procedure creates a secondary index for number values in the VALUE column of a PATH TABLE which is the storage table of an XMLIndex.

Syntax

```sql
DBMS_XMLINDEX.CREATENUMBERINDEX (  
    xml_index_schema   IN   VARCHAR2,  
    xml_index_name     IN   VARCHAR2,  
    num_index_name     IN   VARCHAR2,  
    num_index_clause   IN   VARCHAR2,  
    xmltypename        IN   VARCHAR2);  
```

Parameters

Table 202-3  CREATENUMBERINDEX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml_index_schema</td>
<td>Name of the owner of the XMLIndex</td>
</tr>
<tr>
<td>xml_index_name</td>
<td>Name of the XMLIndex</td>
</tr>
<tr>
<td>num_index_name</td>
<td>Name of the secondary index to be created for number values in the VALUE column of the PATH TABLE of XMLIndex named xml_index_name and owned by xml_index_schema</td>
</tr>
<tr>
<td>num_index_clause</td>
<td>Storage clause to be applied to the number index during its creation. This is a string argument appended to the CREATE INDEX statement for creating the number index.</td>
</tr>
<tr>
<td>xmltypename</td>
<td>The type to which values in the VALUE column of the path table are to be cast. Acceptable values are the following strings: FLOAT, DOUBLE, DECIMAL, INTEGER, NONPOSITIVEINTEGER, NEGATIVEINTEGER, LONG, INT, SHORT, BYTE, NONNEGATIVEINTEGER, UNSIGNEDLONG, UNSIGNEDINT, UNSIGNEDSHORT, UNSIGNEDBYTE, POSITIVEINTEGER.</td>
</tr>
</tbody>
</table>
202.3.3 DROPPARAMETER Procedure

This procedure drops the XMLIndex parameter string that is associated with a given parameter identifier.

Syntax

```
DBMS_XMLINDEX.DROPPARAMETER (
    name        IN      VARCHAR2);
```

Parameters

Table 202-4  DROPPARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Identifier for parameter string</td>
</tr>
</tbody>
</table>

Examples

```
DBMS_XMLINDEX.DROPPARAMETER (
    'myIndexParam');
```

202.3.4 MODIFYPARAMETER Procedure

This procedure modifies the XMLIndex parameter string that is associated with a given parameter identifier.

Syntax

```
DBMS_XMLINDEX.MODIFYPARAMETER (
    name        IN      VARCHAR2,
    parameter   IN      CLOB);
```

Parameters

Table 202-5  MODIFYPARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Identifier for parameter string</td>
</tr>
<tr>
<td>parameter</td>
<td>XMLIndex parameter clause that can appear in a CREATE INDEX or an ALTER INDEX statement</td>
</tr>
</tbody>
</table>

Examples

```
DBMS_XMLINDEX.MODIFYPARAMETER (
    'myIndexParam',
    'PATH TABLE po_ptab
     PATH ID INDEX po_pidx
     ORDER KEY INDEX po_oidx
     VALUE INDEX po_vidx');
```
202.3.5 PROCESS_PENDING Procedure

This procedure processes executes DMLs required to complete a **NONBLOCKING ALTER INDEX ADD_GROUP/ADD_COLUMN** operation on an **XMLIndex**.

Syntax

```sql
DBMS_XMLINDEX.PROCESS_PENDING (    xml_index_schema    IN     VARCHAR2,
                                  xml_index_name      IN     VARCHAR2,
                                  pending_row_count   OUT    BINARY_INTEGER,
                                  error_row_count     OUT    BINARY_INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml_index_schema</td>
<td>Name of the owner of the <strong>XMLIndex</strong></td>
</tr>
<tr>
<td>xml_index_name</td>
<td>Name of the <strong>XMLIndex</strong> to be altered using <strong>NONBLOCKING ALTER INDEX</strong></td>
</tr>
<tr>
<td>pending_row_count</td>
<td>Number of pending rows to be processed</td>
</tr>
<tr>
<td>error_row_count</td>
<td>Number of rows for which indexing may have failed because of an error</td>
</tr>
</tbody>
</table>

Usage Notes

- This procedure will iteratively attempt to index all necessary rows in small batches while skipping rows that are locked and rows for which index maintenance fails with an error. Therefore, it may have to be executed multiple times for an **XMLIndex** until all pending rows are processed. Once all pending rows are processed, user can complete the **NONBLOCKING ALTER INDEX** operation.

- If it is not possible process all the pending rows after multiple trials, the user will have to manually triage the locking or error issues by examining unprocessed rows in **SYS_AIXSXI_######_PENDINGTAB** and **errors** in **SYS_AIXSXI_#####_ERRORTAB**. Keeping track of rows and the errors is useful in triaging issues.

Examples

```sql
EXEC DBMS_XMLINDEX.PROCESS_PENDING(    'SCOTT', 'PO_XMLINDEX_IX', out_param1, out_param2);
```

202.3.6 REGISTERPARAMETER Procedure

This procedure registers a parameter identifier and **XMLIndex** parameter string pair in XDB.

Syntax

```sql
DBMS_XMLINDEX.REGISTERPARAMETER (    name     IN     VARCHAR2,
                                  parameter IN     CLOB);
```
Parameters

Table 202-7  REGISTERPARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Identifier for parameter string</td>
</tr>
<tr>
<td>parameter</td>
<td>XMLIndex parameter clause that can appear in a CREATE INDEX or an ALTER INDEX statement</td>
</tr>
</tbody>
</table>

Examples

DBMS_XMLINDEX.REGISTERPARAMETER (  
  'myIndexParam',  
  'PATH TABLE po_ptab  
  PATH ID INDEX po_pidx  
  ORDER KEY INDEX po_oidx  
  VALUE INDEX po_vidx  
  PATHS(NAMESPACE MAPPING(xmlns:p="http://www.example.com/IPO"))  
  GROUP MASTERGROUP XMLTABLE PO_TAB  
  (''/p:PurchaseOrder'')  
  COLUMNS  
    REFERENCE VARCHAR2(30) PATH ''p:Reference'',  
    REQUESTOR VARCHAR2(30) PATH ''p:Requestor''  
  GROUP ITEMGROUP XMLTABLE ITEMGROUP_TAB  
  (''/p:PurchaseOrder/p:LineItems/p:LineItem''  
  COLUMNS  
    LINENUMBER NUMBER(38) PATH ''@p:ItemNumber'',  
    QUANTITY NUMBER(38) PATH ''@p:Quantity'',  
    DESCRIPTION VARCHAR2(256) PATH ''p:Description''));

202.3.7 SYNCINDEX Procedure

This function synchronizes an asynchronously maintained XMLIndex.

It applies to the XMLIndex changes that are logged in the pending table, and brings the path table up-to-date with the base XMLTYPE column.

Syntax

DBMS_XMLINDEX.SYNCINDEX (  
  xml_index_schema      IN VARCHAR2,  
  xml_index_name        IN VARCHAR2,  
  partition_name        IN VARCHAR2 DEFAULT NULL,  
  reindex               IN BOOLEAN DEFAULT FALSE);

Parameters

Table 202-8  SYNCINDEX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml_index_schema</td>
<td>Name of the owner of the XMLIndex</td>
</tr>
<tr>
<td>xml_schema_name</td>
<td>Name of the XMLIndex</td>
</tr>
<tr>
<td>partition_name</td>
<td>[Currently not supported]</td>
</tr>
</tbody>
</table>
Table 202-8  (Cont.) SYNCINDEX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reindex</td>
<td>Default is FALSE. If set to TRUE, this drops the secondary indexes and recreates them later so that they can be bulk-loaded.</td>
</tr>
</tbody>
</table>

Examples

EXEC DBMS_XMLINDEX.SYNCINDEX('USER1', 'SS_TAB_XMLI', REINDEX=>TRUE);
Using **DBMS_XMLPARSER**, you can access the contents and structure of XML documents. XML describes a class of data XML document objects. It partially describes the behavior of computer programs which process them. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application. This PL/SQL implementation of the XML processor (or parser) follows the W3C XML specification REC-xml-19980210 and includes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The default behavior for this PL/SQL XML parser is to build a parse tree that can be accessed by DOM APIs, validate it if a DTD is found (otherwise, it is non-validating), and record errors if an error log is specified. If parsing fails, an application error is raised.

This chapter contains the following topics:

- **Security Model**
- **Summary of DBMS_XMLPARSER Subprograms**

---

**203.1 DBMS_XMLPARSER Security Model**

Owned by `XDB`, the **DBMS_XMLPARSER** package must be created by `SYS` or `XDB`. The **EXECUTE** privilege is granted to `PUBLIC`.

Subprograms in this package are executed using the privileges of the current user.

**203.2 Summary of DBMS_XMLPARSER Subprograms**

This table lists the **DBMS_XMLPARSER** subprograms and briefly describes them.

---

*Oracle XML DB Developer's Guide*
Table 203-1  DBMS_XMLPARSER Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREEPARSER</td>
<td>Frees a parser object.</td>
</tr>
<tr>
<td>GETDOCTYPE</td>
<td>Gets parsed DTD.</td>
</tr>
<tr>
<td>GETDOCUMENT</td>
<td>Gets DOM document.</td>
</tr>
<tr>
<td>GETRELEASEVERSION</td>
<td>Returns the release version of Oracle XML Parser for PL/SQL.</td>
</tr>
<tr>
<td>GETVALIDATIONMODE</td>
<td>Returns validation mode.</td>
</tr>
<tr>
<td>NEWSPARSER</td>
<td>Returns a new parser instance</td>
</tr>
<tr>
<td>PARSE</td>
<td>Parses XML stored in the given url/file.</td>
</tr>
<tr>
<td>PARSEBUFFER</td>
<td>Parses XML stored in the given buffer.</td>
</tr>
<tr>
<td>PARSECLOB</td>
<td>Parses XML stored in the given clob.</td>
</tr>
<tr>
<td>PARSEDTD</td>
<td>Parses DTD stored in the given url/file.</td>
</tr>
<tr>
<td>PARSEDTDBUFFER</td>
<td>Parses DTD stored in the given buffer.</td>
</tr>
<tr>
<td>PARSEDTDCLOB</td>
<td>Parses DTD stored in the given clob.</td>
</tr>
<tr>
<td>SETBASEDIR</td>
<td>Sets base directory used to resolve relative URLs.</td>
</tr>
<tr>
<td>SETDOCTYPE</td>
<td>Sets DTD.</td>
</tr>
<tr>
<td>SETERRORLOG</td>
<td>Sets errors to be sent to the specified file</td>
</tr>
<tr>
<td>SETPRESERVEWHITESPACE</td>
<td>Sets white space preserve mode</td>
</tr>
<tr>
<td>SETVALIDATIONMODE</td>
<td>Sets validation mode.</td>
</tr>
<tr>
<td>SHOWWARNINGS</td>
<td>Turns warnings on or off.</td>
</tr>
</tbody>
</table>

203.2.1 FREEPARSER

This procedure frees a parser object.

Syntax

PROCEDURE freeParser(  
    p Parser);  

Parameters

Table 203-2  FREEPARSER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
</tbody>
</table>
203.2.2 GETDOCTYPE

The GETDOCTYPE function returns the parsed DTD. This function must be called only after a DTD is parsed.

Syntax

FUNCTION getDoctype(  
p Parser)  
RETURN DOMDocumentType;

Parameters

Table 203-3  GETDOCTYPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
</tbody>
</table>

203.2.3 GETDOCUMENT

GETDOCUMENT returns the document node of a DOM tree document built by the parser. This function must be called only after a document is parsed.

Syntax

FUNCTION GETDOCUMENT(  
p Parser)  
RETURN DOMDocument;

Parameters

Table 203-4  GETDOCUMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
</tbody>
</table>

203.2.4 GETRELEASEVERSION

GETRELEASEVERSION returns the release version of the Oracle XML parser for PL/SQL.

Syntax

FUNCTION getReleaseVersion  
RETURN VARCHAR2;
203.2.5 GETVALIDATIONMODE

The GETVALIDATIONMODE function retrieves the validation mode: TRUE for validating, FALSE otherwise.

Syntax

FUNCTION GETVALIDATIONMODE(
    p Parser)
RETURN BOOLEAN;

Parameters

Table 203-5  GETVALIDATIONMODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
</tbody>
</table>

203.2.6 NEWPARSER

This function returns a new parser instance.

This function must be called before the default behavior of Parser can be changed and if other parse methods need to be used.

Syntax

FUNCTION newParser
RETURN Parser;

203.2.7 PARSE

PARSE parses XML stored in the given URL or file. An application error is raised if parsing fails.

There are several versions of this method.

Syntax

Function. Use this when the default parser behavior is acceptable, and only a URL or file needs to be parsed. Returns the built DOM document.

FUNCTION parse(url VARCHAR2)
RETURN DOMDocument;
Procedure. Any changes to the default parser behavior should be effected before calling this procedure.

PROCEDURE parse(
  p Parser,
  url VARCHAR2);

Parameters

Table 203-6  PARSE Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>(IN)</td>
<td>Complete path of the url/file to be parsed.</td>
</tr>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
</tbody>
</table>

203.2.8 PARSEBUFFER

PARSEBUFFER parses XML stored in the given buffer.

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

PROCEDURE PARSEBUFFER(
  p   Parser,
  doc VARCHAR2);

Parameters

Table 203-7  PARSEBUFFER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>doc</td>
<td>(IN)</td>
<td>XML document buffer to parse.</td>
</tr>
</tbody>
</table>

203.2.9 PARSECLOB

PARSECLOB parses XML stored in the given clob.

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

PROCEDURE PARSECLOB(
  p   Parser,
  doc CLOB);
203.2.10 PARSEDDTD

PARSEDDTD parses the DTD stored in the given URL or file.

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

PROCEDURE PARSEDDTD(
  p    Parser,
  url  VARCHAR2,
  root VARCHAR2);

Parameters

Table 203-9  PARSEDDTD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>url</td>
<td>(IN)</td>
<td>Complete path of the URL or file to be parsed.</td>
</tr>
<tr>
<td>root</td>
<td>(IN)</td>
<td>Name of the root element.</td>
</tr>
</tbody>
</table>

203.2.11 PARSEDTDBUFFER

PARSEDTDBUFFER parses the DTD stored in the given buffer.

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

PROCEDURE PARSEDTDBUFFER(
  p    Parser,
  dtd  VARCHAR2,
  root VARCHAR2);

Parameters

Table 203-9  PARSEDTDBUFFER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>url</td>
<td>(IN)</td>
<td>Complete path of the URL or file to be parsed.</td>
</tr>
<tr>
<td>root</td>
<td>(IN)</td>
<td>Name of the root element.</td>
</tr>
</tbody>
</table>
Parameters

### Table 203-10  PARSEDTDBUFFER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>dtd</td>
<td>(IN)</td>
<td>DTD buffer to parse.</td>
</tr>
<tr>
<td>root</td>
<td>(IN)</td>
<td>Name of the root element.</td>
</tr>
</tbody>
</table>

#### 203.2.12 PARSEDTDCLOB

PARSEDTDCLOB parses the DTD stored in the given clob.

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

**Syntax**

```sql
PROCEDURE PARSEDTDCLOB(
    p Parser,
    dtd CLOB,
    root VARCHAR2);
```

### Parameters

#### Table 203-11  PARSEDTDCLOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>dtd</td>
<td>(IN)</td>
<td>DTD Clob to parse.</td>
</tr>
<tr>
<td>root</td>
<td>(IN)</td>
<td>Name of the root element.</td>
</tr>
</tbody>
</table>

#### 203.2.13 SETBASEDIR

This procedure sets the base directory used to resolve relative URLs. An application error is raised if parsing fails.

**Syntax**

```sql
PROCEDURE setBaseDir(
    p Parser,
    dir VARCHAR2);
```
Parameters

Table 203-12  SETBASEDIR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>dir</td>
<td>(IN)</td>
<td>Directory used as a base directory.</td>
</tr>
</tbody>
</table>

203.2.14 SETDOCTYPE Procedure

This procedure sets a DTD to be used by the parser for validation. This call should be made before the document is parsed.

Syntax

PROCEDURE setDoctype(
  p   Parser,
  dtd DOMDocumentType);

Parameters

Table 203-13  SETDOCTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>dtd</td>
<td>(IN)</td>
<td>DTD to set.</td>
</tr>
</tbody>
</table>

203.2.15 SETERRORLOG Procedure

This procedure sets errors to be sent to the specified file.

Syntax

PROCEDURE setErrorLog(
  p        Parser,
  fileName VARCHAR2);

Parameters

Table 203-14  SETERRORLOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>fileName</td>
<td>(IN)</td>
<td>Complete path of the file to use as the error log.</td>
</tr>
</tbody>
</table>
203.2.16 SETPRESERVEWHITESPACE

This procedure sets whitespace preserving mode.

Syntax

PROCEDURE setPreserveWhitespace(
    p  Parser,
    yes BOOLEAN);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>yes</td>
<td>(IN)</td>
<td>Mode to set: TRUE - preserve, FALSE - don't preserve.</td>
</tr>
</tbody>
</table>

203.2.17 SETVALIDATIONMODE

This procedure sets the validation mode.

Syntax

PROCEDURE setValidationMode(
    p  Parser,
    yes BOOLEAN);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>yes</td>
<td>(IN)</td>
<td>Mode to set: TRUE - validate, FALSE - don't validate.</td>
</tr>
</tbody>
</table>

203.2.18 SHOWWARNINGS

This procedure turns warnings on or off.

Syntax

PROCEDURE showWarnings(
    p  Parser,
    yes BOOLEAN);
### Parameters

#### Table 203-17  SHOWWARNINGS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>(IN)</td>
<td>Parser instance.</td>
</tr>
<tr>
<td>yes</td>
<td>(IN)</td>
<td>Mode to set: TRUE - show warnings, FALSE - don't show warnings.</td>
</tr>
</tbody>
</table>
DBMS_XMLQUERY provides database-to-XMLType functionality.

**Note:**

With Oracle Database 18.1 Release, the DBMS_XMLQUERY package is deprecated. Use DBMS_XMLGEN instead.

The DBMS_XMLQUERY package has been replaced with improved technology. While Oracle recommends you not to begin development using DBMS_XMLQUERY, Oracle continues to support DBMS_XMLQUERY for reasons of backward compatibility. Your existing applications using DBMS_XMLQUERY will continue to work.

**See Also:**

For more information, see DBMS_XMLGEN.

This chapter contains the following topics:

- DBMS_XMLQUERY Security Model
- DBMS_XMLQUERY Constants
- Types
- Summary of DBMS_XMLQUERY Subprograms

### 204.1 DBMS_XMLQUERY Security Model

Owned by XDB, the DBMS_XMLQUERY package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC.

Subprograms in this package are executed using the privileges of the current user.

### 204.2 DBMS_XMLQUERY Constants

The DBMS_XMLQUERY package includes several constants to use when specifying parameter values.

These constants are defined in the following table.
Table 204-1  Constants of DBMS_XMLQUERY

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_ENCODING</td>
<td>Used to signal that the DB character encoding is to be used.</td>
</tr>
<tr>
<td>DEFAULT_ROWSETTAG</td>
<td>The tag name for the element enclosing the XML generated from the result set (that is, for most cases the root node tag name) -- ROWSET.</td>
</tr>
<tr>
<td>DEFAULT_ERRORTAG</td>
<td>The default tag to enclose raised errors -- ERROR.</td>
</tr>
<tr>
<td>DEFAULT_ROWIDATTR</td>
<td>The default name for the cardinality attribute of XML elements corresponding to db.records -- NUM</td>
</tr>
<tr>
<td>DEFAULT_ROWTAG</td>
<td>The default tag name for the element corresponding to db.records -- ROW</td>
</tr>
<tr>
<td>DEFAULT_DATE_FORMAT</td>
<td>Default date mask -- 'MM/dd/yyyy HH:mm:ss'</td>
</tr>
<tr>
<td>ALL_ROWS</td>
<td>Indicates that all rows are needed in the output.</td>
</tr>
<tr>
<td>NONE</td>
<td>Used to specifies that the output should not contain any XML metadata (for example, no DTD).</td>
</tr>
<tr>
<td>DTD</td>
<td>Used to specify that the generation of the DTD is desired.</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>Used to specify that the generation of the XML Schema is desired.</td>
</tr>
<tr>
<td>LOWER_CASE</td>
<td>Use lower case tag names.</td>
</tr>
<tr>
<td>UPPER_CASE</td>
<td>Use upper case tag names.</td>
</tr>
</tbody>
</table>

204.3 Types

The DBMS_XMLQUERY subprograms use the ctxType type.

Table 204-2  Types of DBMS_XMLQUERY

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxType</td>
<td>The type of the query context handle. This is the return type of NEWCONTEXT</td>
</tr>
</tbody>
</table>

204.4 Summary of DBMS_XMLQUERY Subprograms

This table lists the DBMS_XMLQUERY subprograms and briefly describes them.

Table 204-3  DBMS_XMLQUERY Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSECONTEXT</td>
<td>Closes or deallocates a particular query context.</td>
</tr>
<tr>
<td>GETDTD</td>
<td>Generates the DTD.</td>
</tr>
<tr>
<td>GETEXCEPTIONCONTENT</td>
<td>Returns the thrown exception's error code and error message.</td>
</tr>
<tr>
<td>GETNUMROWSPROCESSED</td>
<td>Returns the number of rows processed for the query.</td>
</tr>
<tr>
<td>GETVERSION</td>
<td>Prints the version of the XSU in use.</td>
</tr>
</tbody>
</table>
### Table 204-3  (Cont.) DBMS_XMLQUERY Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETXML</td>
<td>Generates the XML document.</td>
</tr>
<tr>
<td>NEWCONTEXT</td>
<td>Creates a query context and it returns the context handle.</td>
</tr>
<tr>
<td>PROPAGATEORIGINALEXCEPTION</td>
<td>Tells the XSU that if an exception is raised, and is being thrown, the XSU</td>
</tr>
<tr>
<td></td>
<td>should throw the very exception raised; rather then, wrapping it with an</td>
</tr>
<tr>
<td></td>
<td>OracleXMLSQLException.</td>
</tr>
<tr>
<td>REMOVEXSLTPARAM</td>
<td>Removes a particular top-level stylesheet parameter.</td>
</tr>
<tr>
<td>SETBINDVALUE</td>
<td>Sets a value for a particular bind name.</td>
</tr>
<tr>
<td>SETCOLLIDATTRNAME</td>
<td>Sets the name of the id attribute of the collection element's separator tag.</td>
</tr>
<tr>
<td>SETDATAHEADER</td>
<td>Sets the XML data header.</td>
</tr>
<tr>
<td>SETDATEFORMAT</td>
<td>Sets the format of the generated dates in the XML document.</td>
</tr>
<tr>
<td>SETENCODINGTAG</td>
<td>Sets the encoding processing instruction in the XML document.</td>
</tr>
<tr>
<td>SETERRORORTAG</td>
<td>Sets the tag to be used to enclose the XML error documents.</td>
</tr>
<tr>
<td>SETMAXROWS</td>
<td>Sets the maximum number of rows to be converted to XML.</td>
</tr>
<tr>
<td>SETMETAHEADER</td>
<td>Sets the XML meta header.</td>
</tr>
<tr>
<td>SETRAISEEXCEPTION</td>
<td>Tells the XSU to throw the raised exceptions.</td>
</tr>
<tr>
<td>SETRAISENOROWSEXCEPTION</td>
<td>Tells the XSU to throw or not to throw an OracleXMLNoRowsException in the</td>
</tr>
<tr>
<td></td>
<td>case when for one reason or another, the XML document generated is empty.</td>
</tr>
<tr>
<td>SETROWIDATTRNAME</td>
<td>Sets the name of the id attribute of the row enclosing tag.</td>
</tr>
<tr>
<td>SETROWIDATTRVALUE</td>
<td>Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.</td>
</tr>
<tr>
<td>SETROWSETTAG</td>
<td>Sets the tag to be used to enclose the XML dataset.</td>
</tr>
<tr>
<td>SETROWTAG</td>
<td>Sets the tag to be used to enclose the XML element.</td>
</tr>
<tr>
<td>SETSKIPROWS</td>
<td>Sets the number of rows to skip.</td>
</tr>
<tr>
<td>SETSQLTOXMLNAMEESCAPING</td>
<td>This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.</td>
</tr>
<tr>
<td>SETYLESHEETHEADER</td>
<td>Sets the stylesheet header.</td>
</tr>
<tr>
<td>SETTAGCASE</td>
<td>Specified the case of the generated XML tags.</td>
</tr>
<tr>
<td>SETXSLT</td>
<td>Registers a stylesheet to be applied to generated XML.</td>
</tr>
<tr>
<td>SETXSLTPARAM</td>
<td>Sets the value of a top-level stylesheet parameter.</td>
</tr>
<tr>
<td>USENULLATTRIBUTEINDICATOR</td>
<td>Specifies weather to use an XML attribute to indicate NULLness.</td>
</tr>
</tbody>
</table>
Table 204-3  (Cont.) DBMS_XMLQUERY Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USETYPEFORCOLLELEMTAG</td>
<td>Tells the XSU to use the collection element's type name as the collection element tag name.</td>
</tr>
</tbody>
</table>

204.4.1 CLOSECONTEXT

This procedure closes or deallocates a particular query context.

Syntax

PROCEDURE CLOSECONTEXT(
  ctxHdl IN ctxType);

Table 204-4  CLOSECONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>

204.4.2 GETDTD

Generates and returns the DTD based on the SQL query used to initialize the context. The options are described in the following table.

Syntax

Function that generates the DTD based on the SQL query that is used to initialize the context.

FUNCTION GETDTD(
  ctxHdl IN ctxType,
  withVer IN BOOLEAN := false)
RETURN CLOB;

Procedure that generates the DTD based on the SQL query that is used to initialize the context. Specifies the output CLOB for XML document result.

PROCEDURE GETDTD(
  ctxHdl IN ctxType,
  xDoc IN CLOB,
  withVer IN BOOLEAN := false);
204.4.3 GETEXCEPTIONCONTENT

GETEXCEPTIONCONTENT returns the thrown exception's SQL error code and error message through the procedure's OUT parameters.

This procedure is a work around the JVM functionality that obscures the original exception by its own exception, rendering PL/SQL unable to access the original exception content.

Syntax

PROCEDURE GETEXCEPTIONCONTENT(
    ctxHdl IN ctxType,
    errNo OUT NUMBER,
    errMsg OUT VARCHAR2);

Parameters

Table 204-6    GETEXCEPTIONCONTENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>errNo</td>
<td>(OUT)</td>
<td>Error number.</td>
</tr>
<tr>
<td>errMsg</td>
<td>(OUT)</td>
<td>Error message.</td>
</tr>
</tbody>
</table>

204.4.4 GETNUMROWSPROCESSED

Return the number of rows processed for the query.

Syntax

FUNCTION GETNUMROWSPROCESSED(
    ctxHdl IN ctxType)
RETURN NUMBER;

Table 204-7    GETNUMROWSPROCESSED Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>
204.4.5 GETVERSION

The GETVERSION procedure prints the version of the XSU in use.

Syntax

PROCEDURE GETVERSION();

204.4.6 GETXML

GETXML creates the new context, executes the query, gets the XML back and closes the context. This is a convenience function. The context does not need to be explicitly opened or closed.

Syntax

This function uses a SQL query in string form.

FUNCTION GETXML(
   sqlQuery IN VARCHAR2,
   metaType IN NUMBER := NONE)
RETURN CLOB;

This function uses a SQL query in CLOB form.

FUNCTION GETXML(
   sqlQuery IN CLOB,
   metaType IN NUMBER := NONE)
RETURN CLOB;

This function generates the XML document based on a SQL query used to initialize the context.

FUNCTION GETXML(
   ctxHdl IN ctxType,
   metaType IN NUMBER := NONE)
RETURN CLOB;

This procedure generates the XML document based on the SQL query used to initialize the context.

PROCEDURE GETXML(
   ctxHdl IN ctxType,
   xDoc IN CLOB,
   metaType IN NUMBER := NONE);
Parameters

Table 204-8  GETXML Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>metaType</td>
<td>(IN)</td>
<td>XML metadatatype (NONE, DTD, or SCHEMA).</td>
</tr>
<tr>
<td>sqlQuery</td>
<td>(IN)</td>
<td>SQL query.</td>
</tr>
<tr>
<td>xDoc</td>
<td>(IN)</td>
<td>CLOB into which to write the generated XML document.</td>
</tr>
</tbody>
</table>

204.4.7 NEWCONTEXT

NEWCONTEXT creates a save context and returns the context handle.

Syntax

FUNCTION NEWCONTEXT(
    sqlQuery IN VARCHAR2)
RETURN ctxType;

FUNCTION NEWCONTEXT(
    sqlQuery IN CLOB)
RETURN ctxType;

Parameters

Table 204-9  NEWCONTEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlQuery</td>
<td>(IN)</td>
<td>SQL query, the results of which to convert to XML.</td>
</tr>
</tbody>
</table>

204.4.8 PROPAGATEORIGINALEXCEPTION

The PROPAGATEORIGINALEXCEPTION procedure specifies whether to throw every original exception raised or to wrap it in an OracleXMLSQLException.

Syntax

PROCEDURE PROPAGATEORIGINALEXCEPTION(
    ctxHdl IN ctxType,
    flag IN BOOLEAN);
Parameters

Table 204-10  PROPAGATEORIGINALEXCEPTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>TRUE if want to propagate original exception, FALSE to wrap in OracleXMLException.</td>
</tr>
</tbody>
</table>

204.4.9 REMOVEXSLTPARAM

This procedure removes the value of a top-level stylesheet parameter. If no stylesheet is registered, this method is not operational.

Syntax

PROCEDURE REMOVEXSLTPARAM(
    ctxHdl IN ctxType,
    name IN VARCHAR2);

Table 204-11  REMOVEXSLTPARAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>name</td>
<td>(IN)</td>
<td>Name of the top level stylesheet parameter.</td>
</tr>
</tbody>
</table>

204.4.10 SETBINDVALUE

This procedure sets a value for a particular bind name.

Syntax

PROCEDURE SETBINDVALUE(
    ctxHdl IN ctxType,
    bindName IN VARCHAR2,
    bindValue IN VARCHAR2);

Table 204-12  SETBINDVALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>bindName</td>
<td>(IN)</td>
<td>Bind name.</td>
</tr>
<tr>
<td>bindValue</td>
<td>(IN)</td>
<td>Bind value.</td>
</tr>
</tbody>
</table>
204.4.11 SETCOLLIDATTRNAME

This procedure sets the name of the id attribute of the collection element's separator tag. Passing NULL or an empty string for the tag causes the row id attribute to be omitted.

Syntax

PROCEDURE SETCOLLIDATTRNAME(
  ctxHdl IN ctxType,
  attrName IN VARCHAR2);

Table 204-13  SETCOLLIDATTRNAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>attrName</td>
<td>(IN)</td>
<td>Attribute name.</td>
</tr>
</tbody>
</table>

204.4.12 SETDATAHEADER

This procedure sets the XML data header.

The data header is an XML entity that is appended at the beginning of the query-generated XML entity, the rowset. The two entities are enclosed by the docTag argument. The last data header specified is used. Passing in NULL for the header parameter unsets the data header.

Syntax

PROCEDURE SETDATAHEADER(
  ctxHdl IN ctxType,
  header IN CLOB := null,
  tag IN VARCHAR2 := null);

Table 204-14  SETDATAHEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>header</td>
<td>(IN)</td>
<td>Header.</td>
</tr>
<tr>
<td>tag</td>
<td>(IN)</td>
<td>Tag used to enclose the data header and the rowset.</td>
</tr>
</tbody>
</table>

204.4.13 SETDATEFORMAT

This procedure sets the format of the generated dates in the XML document.

The syntax of the date format pattern, the date mask, should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to NULL or an empty string sets the default mask -- DEFAULT_DATE_FORMAT.
Syntax
PROCEDURE SETDATEFORMAT(
ctxHdl IN ctxType,
mask IN VARCHAR2);

Table 204-15  SETDATEFORMAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>mask</td>
<td>(IN)</td>
<td>The date mask.</td>
</tr>
</tbody>
</table>

204.4.14 SETENCODINGTAG Procedure

This procedure sets the encoding processing instruction in the XML document.

Syntax
PROCEDURE SETENCODINGTAG(
ctxHdl IN ctxType,
enc IN VARCHAR2 := DB_ENCODING);

Table 204-16  SETENCODINGTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>enc</td>
<td>(IN)</td>
<td>The encoding to use.</td>
</tr>
</tbody>
</table>

204.4.15 SETERRORTAG Procedure

This procedure sets the tag to be used to enclose the XML error documents.

Syntax
PROCEDURE SETERRORTAG(
ctxHdl IN ctxType,
tag IN VARCHAR2);

Table 204-17  SETERRORTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>tag</td>
<td>(IN)</td>
<td>Tag name.</td>
</tr>
</tbody>
</table>
204.4.16 SETMAXROWS Procedure

This procedure sets the maximum number of rows to be converted to XML. By default, there is no set maximum.

Syntax

PROCEDURE SETMAXROWS (  
ctxHdl IN ctxType,  
rows IN NUMBER);  

Table 204-18  SETMAXROWS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>rows</td>
<td>(IN)</td>
<td>Maximum number of rows to generate.</td>
</tr>
</tbody>
</table>

204.4.17 SETMETAHEADER Procedure

This procedure sets the XML meta header.

When set, the header is inserted at the beginning of the metadata part (DTD or XMLSchema) of each XML document generated by this object. The last meta header specified is used. Passing in NULL for the header parameter unsets the meta header.

Syntax

PROCEDURE SETMETAHEADER(  
ctxHdl IN ctxType,  
header IN CLOB := null);  

Table 204-19  SETMETAHEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>Header</td>
<td>(IN)</td>
<td>Header.</td>
</tr>
</tbody>
</table>

204.4.18 SETRAISEEXCEPTION

This procedure specifies whether to throw raised exceptions.

If this call isn't made or if FALSE is passed to the flag argument, the XSU catches the SQL exceptions and generates an XML document from the exception message.

Syntax

PROCEDURE SETRAISEEXCEPTION(  
ctxHdl IN ctxType,  
flag IN BOOLEAN:=true);
Table 204-20  SETRAISEEXCEPTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Throw raised exceptions? TRUE for yes, otherwise FALSE.</td>
</tr>
</tbody>
</table>

204.4.19 SETRAISENOROWSEXCEPTION

This procedure specifies whether to throw an OracleXMLNoRowsException when the generated XML document is empty. By default, the exception is not thrown.

Syntax

PROCEDURE SETRAISENOROWSEXCEPTION(
  ctxHdl IN ctxType,
  flag IN BOOLEAN:=false);

Table 204-21  SETRAISENOROWSEXCEPTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Throws an OracleXMLNoRowsException if set to TRUE.</td>
</tr>
</tbody>
</table>

204.4.20 SETROWIDATTRNAME

This procedure sets the name of the id attribute of the row enclosing tag. Passing NULL or an empty string for the tag causes the row id attribute to be omitted.

Syntax

PROCEDURE SETROWIDATTRNAME(
  ctxHdl IN ctxType,
  attrName IN VARCHAR2);

Table 204-22  SETROWIDATTRNAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>attrName</td>
<td>(IN)</td>
<td>Attribute name.</td>
</tr>
</tbody>
</table>

204.4.21 SETROWIDATTRVALUE

This procedure specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.

Passing NULL or an empty string for the colName assigns the row count value (0, 1, 2 and so on) to the row id attribute.
Syntax
PROCEDURE SETROWIDATTRVALUE(
ctxHdl IN ctxType,
colName IN VARCHAR2);

Table 204-23  SETROWIDATTRVALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>colName</td>
<td>(IN)</td>
<td>Column whose value is to be assigned to the row id attribute.</td>
</tr>
</tbody>
</table>

204.4.22 SETROWSETTAG

This procedure sets the tag to be used to enclose the XML dataset.

Syntax
PROCEDURE SETROWSETTAG(
ctxHdl IN ctxType,
tag IN VARCHAR2);

Table 204-24  SETROWSETTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>tag</td>
<td>(IN)</td>
<td>Tag name.</td>
</tr>
</tbody>
</table>

204.4.23 SETROWTAG

This procedure sets the tag to be used to enclose the XML element corresponding to a db.record.

Syntax
PROCEDURE SETROWTAG(
ctxHdl IN ctxType,
tag IN VARCHAR2);

Table 204-25  SETROWTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>tag</td>
<td>(IN)</td>
<td>Tag name.</td>
</tr>
</tbody>
</table>
### 204.4.24 SETSKIPROWS

`SETSKIPROWS` sets the number of rows to skip. By default, 0 rows are skipped.

**Syntax**

```sql
PROCEDURE SETSKIPROWS(
  ctxHdl IN ctxType,
  rows IN NUMBER);
```

**Table 204-26  SETSKIPROWS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>rows</td>
<td>(IN)</td>
<td>Maximum number of rows to skip.</td>
</tr>
</tbody>
</table>

### 204.4.25 SETSQLTOXMLNAMEESCAPING

This procedure turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

**Syntax**

```sql
PROCEDURE SETSQLTOXMLNAMEESCAPING(
  ctxHdl IN ctxType,
  flag IN BOOLEAN := true);
```

**Table 204-27  SETSQLTOXMLNAMEESCAPING Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Turn on escaping? TRUE for yes, otherwise FALSE.</td>
</tr>
</tbody>
</table>

### 204.4.26 SETSTYLESHEETHEADER

`SETSTYLESHEETHEADER` sets the stylesheet header (the stylesheet processing instructions) in the generated XML document.

Passing NULL for the `uri` argument will unset the stylesheet header and the stylesheet type.

**Syntax**

```sql
PROCEDURE SETSTYLESHEETHEADER(
  ctxHdl IN ctxType,
  uri IN VARCHAR2,
  type IN VARCHAR2 := 'text/xsl');
```
Table 204-28  SETSTYLESHEETHEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>uri</td>
<td>(IN)</td>
<td>Stylesheet URI.</td>
</tr>
<tr>
<td>type</td>
<td>(IN)</td>
<td>Stylesheet type; defaults to &quot;text/xsl&quot;.</td>
</tr>
</tbody>
</table>

204.4.27 SETTAGCASE

SETTAGCASE specifies the case of the generated XML tags.

Syntax

PROCEDURE SETTAGCASE(
    ctxHdl IN ctxType,
    tCase IN NUMBER);

Table 204-29  SETTAGCASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>tCase</td>
<td>(IN)</td>
<td>The tag’s case:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 0 for as are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 1 for lower case</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 2 for upper case</td>
</tr>
</tbody>
</table>

204.4.28 SETXSLT

SETXSLT registers a stylesheet to be applied to generated XML. If a stylesheet was already registered, it is replaced by the new one. Passing NULL for the uri argument, or NULL or an empty string for the stylesheet argument, unsets the stylesheet header and type.

Syntax

To unregister the stylesheet, pass in NULL for the URI.

PROCEDURE SETXSLT(
    ctxHdl IN ctxType,
    uri IN VARCHAR2,
    ref IN VARCHAR2 := null);

To unregister the stylesheet pass in NULL or an empty string for the stylesheet.

PROCEDURE SETXSLT(
    ctxHdl IN ctxType,
    stylesheet CLOB,
    ref IN VARCHAR2 := null);
Parameters

Table 204-30  SETXSLT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>uri</td>
<td>(IN)</td>
<td>Stylesheet URL.</td>
</tr>
<tr>
<td>stylesheet</td>
<td>(IN)</td>
<td>Stylesheet.</td>
</tr>
<tr>
<td>ref</td>
<td>(IN)</td>
<td>URL to include, imported and external entities.</td>
</tr>
</tbody>
</table>

204.4.29 SETXSLTPARAM

SETXSLTPARAM sets the value of a top-level stylesheet parameter.

The parameter value is expected to be a valid XPath expression; the string literal values would therefore have to be quoted explicitly. If no stylesheet is registered, this method is not operational.

Syntax

PROCEDURE SETXSLTPARAM(
    ctxHdl IN ctxType,
    name IN VARCHAR2,
    value IN VARCHAR2);

Table 204-31  SETXSLTPARAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>name</td>
<td>(IN)</td>
<td>Name of the top level stylesheet parameter.</td>
</tr>
<tr>
<td>value</td>
<td>(IN)</td>
<td>Value to be assigned to the stylesheet parameter.</td>
</tr>
</tbody>
</table>

204.4.30 USENULLATTRIBUTEINDICATOR

This procedure specifies whether to use an XML attribute to indicate NULLness, or to do this by omitting the particular entity in the XML document.

Syntax

PROCEDURE USENULLATTRIBUTEINDICATOR(
    ctxHdl IN ctxType,
    flag IN BOOLEAN);

Table 204-32  USENULLATTRIBUTEINDICATOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>
Table 204-32  USENULLATTRIBUTEINDICATOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Sets attribute to NULL if TRUE, omits from XML document if FALSE.</td>
</tr>
</tbody>
</table>

204.4.31 USETYPEFORCOLLELEMTAG

This procedure specifies whether to use the collection element's type name as its element tag name.

By default, the tag name for elements of a collection is the collection's tag name followed by _item.

Syntax

PROCEDURE USETYPEFORCOLLELEMTAG(
ctxHdl IN ctxType,
flag IN BOOLEAN := true);

Table 204-33  USETYPEFORCOLLELEMTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Turn on use of the type name?</td>
</tr>
</tbody>
</table>
DBMS_XMLSAVE provides XML to database-type functionality.

**Note:**
With Oracle Database 18.1 Release, the DBMS_XMLSAVE package is deprecated. Use DBMS_XMLSTORE instead.

The DBMS_XMLSAVE package has been replaced with improved technology. While Oracle recommends you not to begin development using DBMS_XMLSAVE, Oracle continues to support this package for reasons of backward compatibility. Your existing applications using DBMS_XMLSAVE will continue to work.

This chapter contains the following topics:

- **DBMS_XMLSAVE Security Model**
- **DBMS_XMLSAVE Constants**
- **Types**
- **Summary of DBMS_XMLSAVE Subprograms**

**See Also:**
- For more information on DBMS_XMLSTORE, see DBMS_XMLSTORE
- Oracle XML DB Developer’s Guide

### 205.1 DBMS_XMLSAVE Security Model

Owned by XDB, the DBMS_XMLSAVE package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC.

Subprograms in this package are executed using the privileges of the current user.

### 205.2 DBMS_XMLSAVE Constants

The DBMS_XMLSAVE includes constants to use when specifying parameter values. These constants are defined in the following table.
Table 205-1  Constants of DBMS_XMLSAVE

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_ROWTAG</td>
<td>The default tag name for the element corresponding to database records -- ROW</td>
</tr>
<tr>
<td>DEFAULT_DATE_FORMAT</td>
<td>Default date mask: 'MM/dd/yyyy HH:mm:ss'</td>
</tr>
<tr>
<td>MATCH_CASE</td>
<td>Used to specify that when mapping XML elements to database entities; the XSU should be case sensitive.</td>
</tr>
<tr>
<td>IGNORE_CASE</td>
<td>Used to specify that when mapping XML elements to database entities the XSU should be case insensitive.</td>
</tr>
</tbody>
</table>

205.3 Types

The DBMS_XMLSAVE subprograms use the ctxType Type.

Table 205-2  Types of DBMS_XMLSAVE

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxType</td>
<td>The type of the query context handle. The type of the query context handle. The return type of NEWCONTEXT.</td>
</tr>
</tbody>
</table>

205.4 Summary of DBMS_XMLSAVE Subprograms

This table lists the DBMS_XMLSAVE subprograms and briefly describes them.

Table 205-3  DBMS_XMLSAVE Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEARKEYCOLUMNLIST</td>
<td>Clears the key column list.</td>
</tr>
<tr>
<td>CLEARUPDATECOLUMNLIST</td>
<td>Clears the update column list.</td>
</tr>
<tr>
<td>CLOSECONTEXT</td>
<td>It closes/deallocates a particular save context.</td>
</tr>
<tr>
<td>DELETEXML</td>
<td>Deletes records specified by data from the XML document, from the table specified at the context creation time.</td>
</tr>
<tr>
<td>GETEXCEPTIONCONTENT</td>
<td>Returns the thrown exception's error code and error message.</td>
</tr>
<tr>
<td>INSERTXML</td>
<td>Inserts the XML document into the table specified at the context creation time.</td>
</tr>
<tr>
<td>NEWCONTEXT</td>
<td>Creates a save context, and returns the context handle.</td>
</tr>
<tr>
<td>PROPAGATEORIGINALEXCEPTION</td>
<td>Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.</td>
</tr>
<tr>
<td>REMOVEXSLTPARAM</td>
<td>Removes the value of a top-level stylesheet parameter.</td>
</tr>
<tr>
<td>SETBATCHSIZE</td>
<td>Changes the batch size used during DML operations.</td>
</tr>
</tbody>
</table>
Table 205-3  (Cont.) DBMS_XMLSAVE Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETCOMMITBATCH</td>
<td>Sets the commit batch size.</td>
</tr>
<tr>
<td>SETDATEFORMAT</td>
<td>Sets the format of the generated dates in the XML document.</td>
</tr>
<tr>
<td>SETIGNORECASE</td>
<td>The XSU does mapping of XML elements to database.</td>
</tr>
<tr>
<td>SETKEYCOLUMN</td>
<td>This method adds a column to the key column list.</td>
</tr>
<tr>
<td>SETPRESERVEWHITESPACE</td>
<td>Tells the XSU whether to preserve whitespace or not.</td>
</tr>
<tr>
<td>SETROWTAG</td>
<td>Names the tag used in the XML document to enclose the XML elements corresponding to database.</td>
</tr>
<tr>
<td>SETSQLTOXMLNAMEESCAPING</td>
<td>This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.</td>
</tr>
<tr>
<td>SETUPDATECOLUMN</td>
<td>Adds a column to the update column list.</td>
</tr>
<tr>
<td>SETXSLT</td>
<td>Registers a XSL transform to be applied to the XML to be saved.</td>
</tr>
<tr>
<td>SETXSLTPARAM</td>
<td>Sets the value of a top-level stylesheet parameter.</td>
</tr>
<tr>
<td>UPDATEXML</td>
<td>Updates the table given the XML document.</td>
</tr>
</tbody>
</table>

205.4.1 CLEARKEYCOLUMNLIST

This procedure clears the key column list.

Syntax

```
PROCEDURE clearKeyColumnList(
   ctxHdl IN ctxType);
```

Parameters

Table 205-4  CLEARKEYCOLUMNLIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>

205.4.2 CLEARUPDATECOLUMNLIST

This procedure clears the update column list.

Syntax

```
PROCEDURE clearUpdateColumnList(
   ctxHdl IN ctxType);
```
205.4.3 CLOSECONTEXT

This procedure closes/deallocates a particular save context.

Syntax

```sql
PROCEDURE closeContext(
    ctxHdl IN ctxType);
```

Parameters

### Table 205-6 CLOSECONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>

205.4.4 DELETEXML

The `DELETEXML` function deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted.

The options are described in the following table.

Syntax

```sql
FUNCTION deleteXML(
    ctxHdl IN ctxPType,
    xDoc IN VARCHAR2)
RETURN NUMBER;
```

```sql
FUNCTION deleteXML(
    ctxHdl IN ctxType,
    xDoc IN CLOB)
RETURN NUMBER;
```
205.4.5 GETEXCEPTIONCONTENT

Through its arguments, this method returns the thrown exception's error code and error message, SQL error code.

This is to get around the fact that the JVM throws an exception on top of whatever exception was raised, rendering PL/SQL unable to access the original exception.

Syntax

```
PROCEDURE getExceptionContent(
    ctxHdl IN ctxType,
    errNo OUT NUMBER,
    errMsg OUT VARCHAR2);
```

Parameters

Table 205-8  GETEXCEPTIONCONTENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>errNo</td>
<td>(IN)</td>
<td>Error number.</td>
</tr>
<tr>
<td>errMsg</td>
<td>(IN)</td>
<td>Error message.</td>
</tr>
</tbody>
</table>
205.4.6 INSERTXML

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted. The options are described in the following table.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| FUNCTION insertXML(  
  ctxHd1 IN ctxType,  
  xDoc IN VARCHAR2)  
RETURN NUMBER; | Passes in the xDoc parameter as a VARCHAR2. |

FUNCTION insertXML(  
  ctxHd1 IN ctxType,  
  xDoc IN CLOB)  
RETURN NUMBER; | Passes in the xDoc parameter as a CLOB. |

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHd1</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>xDoc</td>
<td>(IN)</td>
<td>String containing the XML document.</td>
</tr>
</tbody>
</table>

205.4.7 NEWCONTEXT

NEWCONTEXT creates a save context and returns the context handle.

Syntax

FUNCTION newContext(  
  targetTable IN VARCHAR2)  
RETURN ctxType; |

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetTable</td>
<td>(IN)</td>
<td>The target table into which to load the XML document.</td>
</tr>
</tbody>
</table>
205.4.8 PROPAGATEORIGINALEXCEPTION

The `PROPAGATEORIGINALEXCEPTION` procedure tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an `OracleXMLSQLException`.

**Syntax**

```sql
PROCEDURE propagateOriginalException(
    ctxHdl IN ctxType,
    flag IN BOOLEAN);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Propagate the original exception? 0=FALSE, 1=TRUE.</td>
</tr>
</tbody>
</table>

205.4.9 REMOVEXSLTPARAM

This procedure removes the value of a top-level stylesheet parameter.

**Syntax**

```sql
PROCEDURE removeXSLTParam(
    ctxHdl IN ctxType,
    name IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>name</td>
<td>(IN)</td>
<td>Parameter name.</td>
</tr>
</tbody>
</table>

205.4.10 SETBATCHSIZE

This procedure changes the batch size used during DML operations.

When performing inserts, updates or deletes, it is better to batch the operations so that they get executed in one shot rather than as separate statements. The flip side is that more memory is needed to buffer all the bind values. Note that when batching is used, a commit occurs only after a batch is executed. So if one of the statement inside a batch fails, the whole batch is rolled back. This is a small price to pay considering the performance gain; nevertheless, if this behavior is unacceptable, then set the batch size to 1.
Syntax

PROCEDURE setBatchSize(
    ctxHdl IN ctxType,
    batchSize IN NUMBER);

Parameters

Table 205-14 SETBATCHSIZE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>batchSize</td>
<td>(IN)</td>
<td>Batch size.</td>
</tr>
</tbody>
</table>

205.4.11 SETCOMMITBATCH

This procedure sets the commit batch size.

The commit batch size refers to the number or records inserted after which a commit should follow. If batchSize is less than 1 or the session is in "auto-commit" mode, using the XSU does not make any explicit commits. By default, commitBatch is 0.

Syntax

PROCEDURE setCommitBatch(
    ctxHdl IN ctxType,
    batchSize IN NUMBER);

Parameters

Table 205-15 SETCOMMITBATCH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>batchSize</td>
<td>(IN)</td>
<td>Commit batch size.</td>
</tr>
</tbody>
</table>

205.4.12 SETDATEFORMAT

This procedure sets the format of the generated dates in the XML document.

The syntax of the date format pattern, the date mask, should conform to the requirements of the class `java.text.SimpleDateFormat`. Setting the mask to `<code>null</code>` or an empty string unsets the date mask.
Syntax

PROCEDURE setDateFormat(
    ctxHdl IN ctxType,
    mask IN VARCHAR2);

Parameters

Table 205-16  SETDATEFORMAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>mask</td>
<td>(IN)</td>
<td>Syntax of the date format pattern.</td>
</tr>
</tbody>
</table>

205.4.13 SETIGNORECASE

This function tells the XSU whether to ignore case when the XSU maps XML elements to database columns/attributes. This matching is based on the element names (XML tags).

Syntax

PROCEDURE setIgnoreCase(
    ctxHdl IN ctxType,
    flag IN NUMBER);

Parameters

Table 205-17  SETIGNORECASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Ignore tag case in the XML doc? 0=FALSE, 1=TRUE.</td>
</tr>
</tbody>
</table>

205.4.14 SETKEYCOLUMN

This method adds a column to the "key column list".

The value for the column cannot be NULL. In case of update or delete, the columns in the key column list make up the WHERE clause of the statement. The key columns list must be specified before updates can complete; this is optional for delete operations.

Syntax

PROCEDURE setKeyColumn(
    ctxHdl IN ctxType,
    colName IN VARCHAR2);
205.4.15 SETPRESERVEWHITESPACE

This procedure tells the XSU whether or not to preserve whitespace.

**Syntax**

```
PROCEDURE setPreserveWhitespace(
    ctxHdl IN ctxType,
    flag IN BOOLEAN := true);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Should XSU preserve whitespace?</td>
</tr>
</tbody>
</table>

205.4.16 SETROWTAG

This procedure names the tag used in the XML document to enclose the XML elements corresponding to db. records.

**Syntax**

```
PROCEDURE setRowTag(
    ctxHdl IN ctxType,
    tag IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>tag</td>
<td>(IN)</td>
<td>Tag name.</td>
</tr>
</tbody>
</table>
205.4.17 SETSQLTOXMNAMEESCAPING

**SETSQLTOXMNAMEESCAPING** turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

**Syntax**

```sql
PROCEDURE setSQLToXMLNameEscaping(  
  ctxHdl IN ctxType,  
  flag IN BOOLEAN := true);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>Turn on escaping?</td>
</tr>
</tbody>
</table>

205.4.18 SETUPDATECOLUMN

**SETUPDATECOLUMN** adds a column to the update column list.

In case of insert, the default is to insert values to all the columns in the table. In case of updates, the default is to only update the columns corresponding to the tags present in the **ROW** element of the XML document. When the update column list is specified, the columns making up this list alone will get updated or inserted into.

**Syntax**

```sql
PROCEDURE setUpdateColumn(  
  ctxHdl IN ctxType,  
  colName IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>colName</td>
<td>(IN)</td>
<td>Column to be added to the update column list.</td>
</tr>
</tbody>
</table>

205.4.19 SETXSLT

**SETXSLT** registers an XSL transform to be applied to the XML to be saved.

If a stylesheet was already registered, it gets replaced by the new one. To un-register the stylesheet, pass in null for the URI. The options are described in the following table.
Syntax

Table 205-23  SETXSLT Procedure Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| PROCEDURE setXSLT(  
  ctxHdl IN ctxType,  
  uri IN VARCHAR2,  
  ref IN VARCHAR2 := null); | Passes in the stylesheet through a URI. |
| PROCEDURE setXSLT(  
  ctxHdl IN ctxType,  
  stylesheet IN CLOB,  
  ref IN VARCHAR2 := null); | Passes in the stylesheet through a CLOB. |

Parameters

Table 205-24  SETXSLT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>uri</td>
<td>(IN)</td>
<td>URI to the stylesheet to register.</td>
</tr>
<tr>
<td>ref</td>
<td>(IN)</td>
<td>URL for include, import, and external entities.</td>
</tr>
<tr>
<td>stylesheet</td>
<td>(IN)</td>
<td>CLOB containing the stylesheet to register.</td>
</tr>
</tbody>
</table>

205.4.20 SETXSLTPARAM

SETXSLTPARAM sets the value of a top-level stylesheet parameter.

The parameter is expected to be a valid XPath expression; literal values would therefore have to be explicitly quoted.

Syntax

PROCEDURE setXSLTParam(  
  ctxHdl IN ctxType,  
  name IN VARCHAR2,  
  value IN VARCHAR2);
Parameters

Table 205-25  SETXSLTPARAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>name</td>
<td>(IN)</td>
<td>Parameter name.</td>
</tr>
<tr>
<td>value</td>
<td>(IN)</td>
<td>Parameter value as an XPath expression</td>
</tr>
</tbody>
</table>

205.4.21 UPDATEXML

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated.

The options are described in the following table.

Syntax

FUNCTION updateXML(
  ctxHdl IN ctxType,
  xDoc IN VARCHAR2)
RETURN NUMBER;

FUNCTION updateXML(
  ctxHdl IN ctxType,
  xDoc IN CLOB)
RETURN NUMBER;

Parameters

Table 205-26  UPDATEXML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>xDoc</td>
<td>(IN)</td>
<td>String containing the XML document.</td>
</tr>
</tbody>
</table>
DBMS_XMLSCHEMA package provides procedures to manage XML schemas. It is created by script dbmsxsch.sql during Oracle database installation.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Views
- Operational Notes
- Summary of DBMS_XMLSCHEMA Subprograms

See Also:

Oracle XML DB Developer's Guide

206.1 DBMS_XMLSCHEMA Overview

The DBMS_XMLSCHEMA package uses subprograms to manage XML schemas. These subprograms provide the following XML schema management:

- Register an XML schema
- Delete a previously registered XML schema
- Re-compile a previously registered XML schema
- Generate an XML schema
- Evolves an XML schema

206.2 DBMS_XMLSCHEMA Security Model

Owned by XDB, the DBMS_XMLSCHEMA package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.
206.3 DBMS_XMLSCHEMA Constants

The DBMS_XMLSCHEMA package defines several constants to use when specifying parameter values.

These constants are shown in following tables.

- Table 206-1
- Table 206-2
- Table 206-3

Table 206-1  DBMS_XMLSCHEMA Constants - Delete Option

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE_RESTRICT</td>
<td>NUMBER</td>
<td>1</td>
<td>Deletion of an XML schema fails if there are any tables or XML schemas that depend on it</td>
</tr>
<tr>
<td>DELETE_INVALIDATE</td>
<td>NUMBER</td>
<td>2</td>
<td>Deletion of an XML schema does not fail if there are tables or XML schemas that depend on it. All dependent tables and schemas are invalidated.</td>
</tr>
<tr>
<td>DELETE_CASCADE</td>
<td>NUMBER</td>
<td>3</td>
<td>Deletion of an XML schema also drops all SQL types and default tables associated with it. SQL types are dropped only if gentypes argument was set to TRUE during registration of the XML schema. However, deletion of the XML schema fails if there are any instance documents conforming to the schema or any dependent XML schemas.</td>
</tr>
<tr>
<td>DELETE_CASCADE_FORCE</td>
<td>NUMBER</td>
<td>4</td>
<td>This option is similar to DELETE_CASCADE except that it does not check for any stored instance documents conforming to the schema or any dependent XML schemas. Also, it ignores any errors.</td>
</tr>
</tbody>
</table>

Table 206-2  DBMS_XMLSCHEMA Constants - Enable Hierarchy

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_HIERARCHY_NONE</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>The ENABLE_HIERARCHY procedure of the DBMS_XDBZ package will not be called on any tables created while registering that schema</td>
</tr>
</tbody>
</table>
Table 206-2  (Cont.) DBMS_XMLSCHEMA Constants - Enable Hierarchy

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_HIERARCHY_CONTENTS</td>
<td>PLS_INTEGER</td>
<td>2</td>
<td>The ENABLE_HIERARCHY procedure of the DBMS_XDBZ package will be called for all tables created during schema registration with hierarchy_type as DBMS_XDBZ.ENABLE_CONTENTS</td>
</tr>
<tr>
<td>ENABLE_HIERARCHY_RESMETADATA</td>
<td>PLS_INTEGER</td>
<td>3</td>
<td>The ENABLE_HIERARCHY procedure of the DBMS_XDBZ package will be called on all tables created during schema registration with hierarchy_type as DBMS_XDBZ.ENABLE_RESMETADATA. Users should pass in DBMS_XMLSCHEMA.ENABLE_RESMETADATA for schemas they intend to use as resource metadata tables.</td>
</tr>
</tbody>
</table>

Table 206-3  DBMS_XMLSCHEMA Constants - Register CSID

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER_NODOCID</td>
<td>NUMBER</td>
<td>1</td>
<td>If a schema is registered for metadata use (using the value ENABLE_HIER_RESMETADATA for parameter enablehierarchy during registration), a column named DOCID is added to all tables created during schema registration. This constant can be used in the options argument of REGISTERSCHEMA to prevent the creation of this column if the user wishes to optimize on storage.</td>
</tr>
<tr>
<td>REGISTER_CSID_NULL</td>
<td>NUMBER</td>
<td>-1</td>
<td>If user wishes to not specify the character set of the input schema document when invoking REGISTERSCHEMA, this value can be used for the csid parameter.</td>
</tr>
</tbody>
</table>

206.4 Views

This table lists the views used by the DBMS_XMLSCHEMA package.

The columns of these views are described in detail in the Oracle Database Reference.
Table 206-4  Summary of Views used by DBMS_XMLSCHEMA

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_XML_SCHEMAS</td>
<td>All registered XML Schemas owned by the user</td>
</tr>
<tr>
<td>ALL_XML_SCHEMAS</td>
<td>All registered XML Schemas usable by the current user</td>
</tr>
<tr>
<td>DBA_XML_SCHEMAS</td>
<td>All registered XML Schemas in the database</td>
</tr>
<tr>
<td>DBA_XML_TABLES</td>
<td>All XMLType tables in the system</td>
</tr>
<tr>
<td>USER_XML_TABLES</td>
<td>All XMLType tables owned by the current user</td>
</tr>
<tr>
<td>ALL_XML_TABLES</td>
<td>All XMLType tables usable by the current user</td>
</tr>
<tr>
<td>DBA_XML_TAB_COLS</td>
<td>All XMLType table columns in the system</td>
</tr>
<tr>
<td>USER_XML_TAB_COLS</td>
<td>All XMLType table columns in tables owned by the current user</td>
</tr>
<tr>
<td>ALL_XML_TAB_COLS</td>
<td>All XMLType table columns in tables usable by the current user</td>
</tr>
<tr>
<td>DBA_XML_VIEWS</td>
<td>All XMLType views in the system</td>
</tr>
<tr>
<td>USER_XML_VIEWS</td>
<td>All XMLType views owned by the current user</td>
</tr>
<tr>
<td>ALL_XML_VIEWS</td>
<td>All XMLType views usable by the current user</td>
</tr>
<tr>
<td>DBA_XML_VIEW_COLS</td>
<td>All XMLType view columns in the system</td>
</tr>
<tr>
<td>USER_XML_VIEW_COLS</td>
<td>All XMLType view columns in views owned by the current user</td>
</tr>
<tr>
<td>ALL_XML_VIEW_COLS</td>
<td>All XMLType view columns in views usable by the current user</td>
</tr>
</tbody>
</table>

206.5 DBMS_XMLSCHEMA Operational Notes

There are guidelines for using in-place XML schema evolution.

Before you perform an in-place XML-schema evolution, you should follow these preparatory steps:

1. Back up all existing data (instance documents) for the XML schema that will be evolved.

2. Perform a dry run using trace only, that is, without actually evolving the XML schema or updating any instance documents, to produce a trace of the update operations that would be performed during evolution. To do this, set the flag parameter value to only INPLACE_TRACE. Do not also use INPLACE_EVOLVE. After performing the dry run, examine the trace file, verifying that the listed DDL operations are in fact those that you intend.

206.6 Summary of DBMS_XMLSCHEMA Subprograms

This table lists the DBMS_XMLSCHEMA subprograms and briefly describes them.
Table 206-5  DBMS_XMLSCHEMA Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPILESCHEMA</td>
<td>Used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state. Can result in a ORA-31001 exception: invalid resource handle or path name.</td>
</tr>
<tr>
<td>COPYEVOLVE Procedure</td>
<td>Evolves registered schemas so that existing XML instances remain valid.</td>
</tr>
<tr>
<td>DELETESchema</td>
<td>Removes the schema from the database</td>
</tr>
<tr>
<td>INPLACEEVOLVE</td>
<td>Evolves registered schemas by propagating schema changes to object types and tables</td>
</tr>
<tr>
<td>PURGESchema</td>
<td>Removes the XML schema</td>
</tr>
<tr>
<td>REGISTERSchema</td>
<td>Registers the specified schema for use by Oracle. This schema can then be used to store documents conforming to this.</td>
</tr>
<tr>
<td>REGISTERURI</td>
<td>Registers an XML schema specified by a URI name</td>
</tr>
</tbody>
</table>

206.6.1 COMPILESCHEMA Procedure

This procedure can be used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state. Can result in a ORA-31001 exception: invalid resource handle or path name.

Syntax

DBMS_XMLSCHEMA.COMPILESCHEMA(
  schemaurl IN VARCHAR2);

Parameters

Table 206-6  COMPILESCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaurl</td>
<td>URL identifying the schema</td>
</tr>
</tbody>
</table>

206.6.2 COPYEVOLVE Procedure

This procedure evolves registered schemas so that existing XML instances remain valid.

This procedure is accomplished in according to the following basic scenario (alternative actions are controlled by the procedure's parameters):

- copies data in schema based XMLType tables to temporary table storage
- drops old tables
- deletes old schemas
- registers new schemas
- creates new XMLType tables
- Populates new tables with data in temporary storage; auxiliary structures (constraints, triggers, indexes, and others) are not preserved
• drops temporary tables

See Also:

– "Schema Evolution" chapter of the Oracle XML DB Developer's Guide for examples on how to evolve existing schemas
– Oracle Database Error Messages for information on exceptions specific to schema evolution, ORA-30142 through ORA-30946.

Syntax

```
DBMS_XMLSCHEMA.COPYEVOLVE(
    schemaurls       IN  XDB$STRUBG_LIST_T,
    newschemas       IN  XMLSequenceType,
    transforms       IN  XMLSequenceType :=NULL,
    preserveolddocs  IN  BOOLEAN :=FALSE,
    maptablename     IN  VARCHAR2 :=NULL,
    generatetables   IN  BOOLEAN :=TRUE,
    force            IN  BOOLEAN :=FALSE,
    schemaowners     IN  XDB$STRING_LIST_T :=NULL
    parallelDegree   IN  PLS_INTEGER := 0,
    options          IN  PLS_INTEGER := 0);
```

Parameters

**Table 206-7  COPYEVOLVE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaurls</td>
<td>VARRAY of URLs of all schemas to be evolved. Should include the dependent schemas. Unless the FORCE parameter is TRUE, URLs should be in the order of dependency.</td>
</tr>
<tr>
<td>newschemas</td>
<td>VARRAY of new schema documents. Should be specified in same order as the corresponding URLs.</td>
</tr>
<tr>
<td>transforms</td>
<td>VARRAY of transforming XSL documents to be applied to schema-based documents. Should be specified in same order as the corresponding URLs. Optional if no transformations are required.</td>
</tr>
<tr>
<td>preserveolddocs</td>
<td>Default is FALSE, and temporary tables with old data are dropped. If TRUE, these table are still available after schema evolution is complete.</td>
</tr>
</tbody>
</table>
Table 206-7  (Cont.) COPYEVOLVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maptabname</td>
<td>Specifies the name of the table mapping permanent to temporary tables during the evolution process. Valid columns are:</td>
</tr>
<tr>
<td></td>
<td>·  SCHEMA_URL - VARCHAR2(700) - URL of schema to which this table conforms</td>
</tr>
<tr>
<td></td>
<td>·  SCHEMA_OWNER - VARCHAR2(30) - Owner of the schema</td>
</tr>
<tr>
<td></td>
<td>·  ELEMENT_NAME - VARCHAR2(256) - Element to which this table conforms</td>
</tr>
<tr>
<td></td>
<td>·  TAB_NAME - VARCHAR2(65) - Qualified table name: &lt;owner_name&gt;.&lt;table_name&gt;</td>
</tr>
<tr>
<td></td>
<td>·  COL_NAME - VARCHAR2(4000) - Name of the column (NULL for XMLType tables)</td>
</tr>
<tr>
<td></td>
<td>·  TEMP_TABNAME - VARCHAR2(30) - Name of temporary tables which holds data for this table.</td>
</tr>
<tr>
<td>generatetables</td>
<td>Default is TRUE, and new tables will be generated.</td>
</tr>
<tr>
<td></td>
<td># FALSE:</td>
</tr>
<tr>
<td></td>
<td>·  new tables will not be generated after registration of new schemas</td>
</tr>
<tr>
<td></td>
<td>·  preserveolddocs must be TRUE</td>
</tr>
<tr>
<td></td>
<td>·  maptablename must be non-NULL</td>
</tr>
<tr>
<td>force</td>
<td>Default is FALSE.</td>
</tr>
<tr>
<td></td>
<td># TRUE, ignores errors generated during schema evolution. Used when there are circular dependencies among schemas to ensure that all schemas are stored despite possible errors in registration.</td>
</tr>
<tr>
<td>schemaowners</td>
<td>VARRAY of names of schema owners. Should be specified in same order as the corresponding URLs. Default is NULL,</td>
</tr>
<tr>
<td></td>
<td>assuming that all schemas are owned by the current user.</td>
</tr>
<tr>
<td>paralleldegree</td>
<td>Specifies the degree of parallelism to be used in a PARALLEL hint during the data copy stage of the evolution. If this is 0 (default), the PARALLEL hint will not be given in the data copy statements.</td>
</tr>
<tr>
<td>options</td>
<td>Currently, the only supported option is COPYEVOLVE_BINARY_XML which lets you register the new schemas for binary XML and create the new tables/columns with binary XML as the storage type.</td>
</tr>
</tbody>
</table>

Usage Notes

You should back up all schemas and documents prior to invocation because COPYEVOLVE Procedure deletes all conforming documents prior to implementing the schema evolution.
206.6.3 DELETESCHEMA Procedure

This procedure deletes the XML Schema specified by the URL.

Syntax

```
DBMS_XMLSCHEMA.DELETESCHEMA(
    schema_url IN VARCHAR2,
    delete_option IN PLS_INTEGER := DELETE_RESTRICT);
```

See Also:

"XMLSCHEMA Storage and Query: Basic" chapter of the Oracle XML DB Developer's Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_url</td>
<td>URL identifying the schema to be deleted</td>
</tr>
<tr>
<td>delete_option</td>
<td>Delete options:</td>
</tr>
<tr>
<td></td>
<td>• DELETE_RESTRICT - Schema deletion fails if there are any tables or schemas that depend on this schema</td>
</tr>
<tr>
<td></td>
<td>• DELETE_INVALIDATE - Schema deletion does not fail if there are any dependencies. Instead, it simply invalidates all dependent objects.</td>
</tr>
<tr>
<td></td>
<td>• DELETE_CASCADE - Schema deletion will also drop all default SQL types and default tables. However the deletion fails if there are any stored instances conforming to this schema.</td>
</tr>
<tr>
<td></td>
<td>• DELETE_CASCADE_FORCE - Similar to DELETE_CASCADE except that it does not check for any stored instances conforming to this schema. Also, it ignores any errors.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-31001</td>
<td>Invalid resource handle or path name</td>
</tr>
</tbody>
</table>
206.6.4 INPLACEEVOLVE Procedure

This procedure evolves registered schemas by propagating schema changes to object types and tables.

Syntax

```sql
DBMS_XMLSCHEMA.INPLACEEVOLVE(
    schemaURL    IN   VARCHAR2,
    diffXML      IN   XMLType,
    flags        IN   NUMBER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaURL</td>
<td>URL of the schema to evolve</td>
</tr>
<tr>
<td>diffXML</td>
<td>Changes to be applied to the schema. This is an XML document conforming to the XDIFF schema and specifies what changes need to be applied and the locations in the schema document where the changes are to be applied.</td>
</tr>
<tr>
<td>flags</td>
<td>The following bits may be set in this parameter to control the behavior of this procedure:</td>
</tr>
<tr>
<td></td>
<td>- INPLACE_EVOLVE (value 1, meaning that bit 1 is on) – Perform in-place XML schema evolution: construct a new XML schema and validate it (against the XML schema for XML schemas); construct the DDL statements needed to evolve the instance-document disk structures, execute the DDL statements, and replace the old XML schema with the new.</td>
</tr>
<tr>
<td></td>
<td>- INPLACE_TRACE (value 2, meaning that bit 2 is on) – Perform all steps necessary for in-place evolution, except executing the DDL statements and overwriting the old XML schema with the new, then write both the DDL statements and the new XML schema to a trace file.</td>
</tr>
</tbody>
</table>

That is, each of the bits constructs the new XML schema, validates it, and determines the steps needed to evolve the disk structures underlying the instance documents. In addition:

- Bit INPLACE_EVOLVE carries out those evolution steps and replaces the old XML schema with the new.
- Bit INPLACE_TRACE saves the evolution steps and the new XML schema in a trace file (it does not carry out the evolution steps)

Exceptions

The procedure raises exceptions in the following cases:

- An error will be raised for invalid XPATH expressions and for XDIFF documents that do not conform to the xdiff schema.
- Path expressions that are syntactically correct but result in an invalid node in the schema document will result in an error.
- If the schema change makes the schema an ill-formed XML document or an invalid XML schema, this will raise an error.
• Any errors resulting from CREATE TYPE, ALTER TYPE and like commands will generate error messages.

**Usage Notes**

• Users are required to backup all their data before attempting in-place evolution, as there is no rollback with this operation.

• A user must register their new XML schema with the database using the REGISTERSCHEMA Procedures and the REGISTERURI Procedure at a schema URL that is different from that of the one to be evolved. If the new schema registers successfully and is usable, only then should the user attempt to evolve the existing schema to the new schema by means of this subprogram. If the registration of the new schema is successful, then the user must delete this schema (and all its dependent objects) before attempting to evolve the schema at the old schema URL.

### 206.6.5 PURGESCHEMA Procedure

This procedure removes the XML schema.

#### Syntax

```
DBMS_XMLSCHEMA.PURGESCHEMA(
    schemaid IN RAW);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaid</td>
<td>ID of the schema to be purged</td>
</tr>
</tbody>
</table>

#### Usage Notes

• The schema should have been originally registered for binary encoding and should have been deleted in the HIDE mode.

• Once a schema has been deleted in HIDE mode, it continues to exist in the XML DB dictionary and is used for decoding already encoded documents. The user invokes this interface when there are no stored instances encoded with this schema.

• Once the schema is purged, any space used by that schema will be reclaimed and documents encoded using the schema will raise an error if an attempt is made to decode them.

• The Schema ID can be obtained from the catalog views.
206.6.6 REGISTERSCHEMA Procedures

This procedure registers the specified schema for use by the database.

The procedure is overloaded. The different functionality of each form of syntax is presented along with the definition.

Note:

As of Oracle Database 11g Release 2 (11.2) the `genbean` parameter is deprecated. Oracle recommends that you do not use this parameter in new applications. Support for this feature is for backward compatibility only.

See Also:

"XMLSCHEMA Storage and Query: Basic" chapter of the Oracle XML DB Developer's Guide

Syntax

Registers a schema specified as a `VARCHAR2`:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    schemaurl        IN  VARCHAR2,
    schemadoc        IN  VARCHAR2,
    local            IN  BOOLEAN := TRUE,
    gentypes         IN  BOOLEAN := TRUE,
    genbean          IN  BOOLEAN := FALSE,
    gentables        IN  BOOLEAN := TRUE,
    force            IN  BOOLEAN := FALSE,
    owner            IN  VARCHAR2 := NULL,
    enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
    options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a `BFILE`. The contents of the schema document must be in the database character set:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    schemaurl        IN  VARCHAR2,
    schemadoc        IN  BFILE,
    local            IN  BOOLEAN := TRUE,
    gentypes         IN  BOOLEAN := TRUE,
    genbean          IN  BOOLEAN := FALSE,
    gentables        IN  BOOLEAN := TRUE,
    force            IN  BOOLEAN := FALSE,
    owner            IN  VARCHAR2 := NULL,
    enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
    options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a `BFILE` and identifies the character set id of the schema document:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    schemaurl        IN  VARCHAR2,
    schemadoc        IN  BFILE,
    local            IN  BOOLEAN := TRUE,
    gentypes         IN  BOOLEAN := TRUE,
    genbean          IN  BOOLEAN := FALSE,
    gentables        IN  BOOLEAN := TRUE,
    force            IN  BOOLEAN := FALSE,
    owner            IN  VARCHAR2 := NULL,
    enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
    options          IN  PLS_INTEGER := 0);
```
Registers the schema specified as a BLOB. The contents of the schema document must be in the database character set:

```sql
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    schemaurl        IN  VARCHAR2,
    schemadoc        IN  CLOB,
    local            IN  BOOLEAN := TRUE,
    gentypes         IN  BOOLEAN := TRUE,
    genbean          IN  BOOLEAN := TRUE,
    gentables        IN  BOOLEAN := TRUE,
    force            IN  BOOLEAN := TRUE,
    owner            IN  VARCHAR2 := '',
    csid             IN  NUMBER,
    enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
    options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a BLOB and identifies the character set id of the schema document:

```sql
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    schemaurl        IN  VARCHAR2,
    schemadoc        IN  CLOB,
    local            IN  BOOLEAN := TRUE,
    gentypes         IN  BOOLEAN := TRUE,
    genbean          IN  BOOLEAN := FALSE,
    gentables        IN  BOOLEAN := TRUE,
    force            IN  BOOLEAN := FALSE,
    owner            IN  VARCHAR2 := NULL,
    csid             IN  NUMBER,
    enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
    options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a CLOB:

```sql
DBMS_XMLSCHEMA.REGISTERSCHEMA(
    schemaurl        IN  VARCHAR2,
    schemadoc        IN  CLOB,
    local            IN  BOOLEAN := TRUE,
    gentypes         IN  BOOLEAN := TRUE,
    genbean          IN  BOOLEAN := FALSE,
    gentables        IN  BOOLEAN := FALSE,
    force            IN  BOOLEAN := FALSE,
    owner            IN  VARCHAR2 := NULL,
    options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as an XMLTYPE.
DBMS_XMLSCHEMA.REGISTERSCHEMA(
  schemaurl        IN  VARCHAR2,
  schemadoc        IN  SYS.XMLTYPE,
  local            IN  BOOLEAN := TRUE,
  gentypes         IN  BOOLEAN := TRUE,
  genbean          IN  BOOLEAN := FALSE,
  force            IN  BOOLEAN := FALSE,
  owner            IN  VARCHAR2 := NULL,
  enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
  options          IN  PLS_INTEGER := 0);

Registers the schema specified as a BLOB. The contents of the schema document must be in the database character set:

DBMS_XMLSCHEMA.REGISTERSCHEMA(
  schemaurl        IN  VARCHAR2,
  schemadoc        IN  SYS.URIType,
  local            IN  BOOLEAN := TRUE,
  gentypes         IN  BOOLEAN := TRUE,
  genbean          IN  BOOLEAN := FALSE,
  force            IN  BOOLEAN := FALSE,
  owner            IN  VARCHAR2 := NULL,
  enablehierarchy  IN  PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
  options          IN  PLS_INTEGER := 0);

Parameters

Table 206-12  REGISTERSCHEMA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaurl</td>
<td>URL that uniquely identifies the schema document. This value is used to derive the path name of the schema document within the database hierarchy. Can be used inside schemalocation attribute of XML Schema import element.</td>
</tr>
<tr>
<td>schemadoc</td>
<td>A valid XML schema document</td>
</tr>
<tr>
<td>local</td>
<td>Is this a local or global schema?</td>
</tr>
<tr>
<td></td>
<td>• By default, all schemas are registered as local schemas, under /sys/schemas/&lt;username&gt;...</td>
</tr>
<tr>
<td></td>
<td>• If a schema is registered as global, it is added under /sys/schemas/PUBLIC/...</td>
</tr>
<tr>
<td></td>
<td>You need write privileges on the directory to be able to register a schema as global.</td>
</tr>
<tr>
<td>gentypes</td>
<td>Determines whether the schema compiler generates object types. By default, TRUE. If you use binary XML, you must be set gentypes to FALSE.</td>
</tr>
<tr>
<td>genbean</td>
<td>Determines whether the schema compiler generates Java beans. By default, FALSE. Oracle recommends that this parameter always be set to FALSE.</td>
</tr>
<tr>
<td>gentables</td>
<td>Determines whether the schema compiler generates default tables. By default, TRUE</td>
</tr>
<tr>
<td>force</td>
<td>If this parameter is set to TRUE, the schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is FALSE.</td>
</tr>
</tbody>
</table>
Table 206-12  (Cont.) REGSITERSCHEM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.</td>
</tr>
<tr>
<td>csid</td>
<td>Identifies the character set of the input schema document. If this value is 0, the schema document's encoding is determined by the current rule for &quot;text/xml&quot; MIME type.</td>
</tr>
</tbody>
</table>
| enablehierarchy | · ENABLE_HIERARCHY_NONE - enable hierarchy will not be called on any tables created while registering that schema  
· ENABLE_HIERARCHY_CONTENTS - enable hierarchy will be called for all tables created during schema registration with hierarchy_type as DBMS_XDBZ.ENABLE_CONTENTS. This is the default.  
· ENABLE_HIERARCHY_RESMETADATA - enable hierarchy will be called on all tables created during schema registration with hierarchy_type as DBMS_XDBZ.ENABLE_RESMETADATA. Users should pass in DBMS_XMLSHEMA.ENABLE_RESMETADATA for schemas they intend to use as resource metadata tables. |
| options     | Additional options to specify how the schema should be registered. The various options are represented as bits of an integer and the options parameter should be constructed by doing a BITOR of the desired bits. Possible bits:  
· REGISTER_NODOCID - this will suppress the creation of the DOCID column for out of line tables. This is a storage optimization which might be desirable when we do not need to join back to the document table (for example if we do not care about rewriting certain queries that could be rewritten by making use of the DOCID column)  
· REGISTER_BINARYXML - Register the schema for Binary XML  
· REGISTER_NT_AS_IOT - Store nested tables created during schema registration as index organized tables. The default is to store nested tables as heap tables |

206.6.7 REGISTERURI Procedure

This procedure registers an XML Schema specified by a URI name.

```
DBMS_XMLSHEMA.REGISTERURI(
    schemaurl IN VARCHAR2,
    schemadocuri IN VARCHAR2,
)
```

Note: As of Oracle Database 11g Release 2 (11.2) the genbean parameter is deprecated. Oracle recommends that you do not use this parameter in new applications. Support for this feature is for backward compatibility only.
Parameters

Table 206-13 REGISTERURI Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaurl</td>
<td>Uniquely identifies the schema document. Can be used inside schemaLocation attribute of XML Schema import element.</td>
</tr>
<tr>
<td>schemadocuri</td>
<td>Pathname (URI) corresponding to the physical location of the schema document. The URI path could be based on HTTP, FTP, DB or Oracle XML DB protocols. This function constructs a URIType instance using the urifactory, and invokes the REGISTERSCHEMA Procedures.</td>
</tr>
<tr>
<td>local</td>
<td>Determines whether this is a local or global schema. By default, all schemas are registered as local schemas, under /sys/schemas/ &lt;username&gt;/... If a schema is registered as global, it is added under /sys/schemas/PUBLIC/... The user needs write privileges on the directory to register a global schema.</td>
</tr>
<tr>
<td>gentypes</td>
<td>Determines whether the compiler generate object types. By default, TRUE.</td>
</tr>
<tr>
<td>genbean</td>
<td>Determines whether the compiler generate Java beans. By default, FALSE.</td>
</tr>
<tr>
<td>gentables</td>
<td>Determines whether the compiler generate default tables. TRUE by default.</td>
</tr>
<tr>
<td>force</td>
<td>TRUE: schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is FALSE.</td>
</tr>
<tr>
<td>owner</td>
<td>This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.</td>
</tr>
<tr>
<td>options</td>
<td>Additional options to specify how the schema should be registered. The various options are represented as bits of an integer and the options parameter should be constructed by doing a BITOR of the desired bits. Possible bits:</td>
</tr>
<tr>
<td></td>
<td>- REGISTER_NODOCID - this will suppress the creation of the DOCID column for out of line tables. This is a storage optimization which might be desirable when we do not need to join back to the document table (for example if we do not care about rewriting certain queries that could be rewritten by making use of the DOCID column)</td>
</tr>
</tbody>
</table>
The DBMS_XMLSCHEMA_ANNOTATE package provides an interface to manage and configure the structured storage model, mainly through the use of pre-registration schema annotations.

This chapter contains the following topics:

• Overview
• Security Model
• Summary of DBMS_XMLSCHEMA_ANNOTATE Subprograms

See Also:
Oracle XML DB Developer's Guide

207.1 DBMS_XMLSCHEMA_ANNOTATE Overview

The DBMS_XMLSCHEMA_ANNOTATE package contains procedures to manage and configure the structured storage model, mainly through the use of pre-registration schema annotations.

Schema annotations influence the way the XML data is stored. For example, the default table annotation assigns a user-provided name to an XML element instead of allowing the database to generate a system name. Consequently, query plans are more readable and it is easier to create constraints on that table.

207.2 DBMS_XMLSCHEMA_ANNOTATE Security Model

Owned by XDB, the DBMS_XMLSCHEMA_ANNOTATE package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

207.3 Summary of DBMS_XMLSCHEMA_ANNOTATE Subprograms

This table lists and describes the DBMS_XMLSCHEMA_ANNOTATE package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDXDBNAMESPACE Procedure</td>
<td>Adds the XDB namespace required for XDB annotation</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DISABLEDEFAULTTABLECREATION Procedure</td>
<td>Prevents the creation of a table for the top-level element by adding a default table attribute with an empty value to the element</td>
</tr>
<tr>
<td>DISABLEMAINTAINDOM Procedure</td>
<td>Sets the DOM fidelity attribute to FALSE</td>
</tr>
<tr>
<td>ENABLEDEFAULTTABLECREATION Procedure</td>
<td>Enables the creation of ALL top level tables by removing the empty default table name annotation</td>
</tr>
<tr>
<td>ENABLEMAINTAINDOM Procedure</td>
<td>Sets the DOM fidelity attribute to TRUE</td>
</tr>
<tr>
<td>GETSCHEMAANNOTATIONS Function</td>
<td>Creates a document containing the differences between the annotated XML schema and the original XML schema</td>
</tr>
<tr>
<td>GETSIDXDEFFROMVIEW Function</td>
<td>Takes a XMLTABLE view definition on a xmltype column or table and it returns a CLOB which can be used as parameter to create a structured xmlindex that backs up the XMLTABLE view as relational table</td>
</tr>
<tr>
<td>PRINTWARNINGS Procedure</td>
<td>Lets a user raise or suppress a warning if an annotation maps to zero nodes in the XML schema</td>
</tr>
<tr>
<td>REMOVEANYSTORAGE Procedure</td>
<td>Removes the setting of the SQL type from the ANY child of the complex type with the given name</td>
</tr>
<tr>
<td>REMOVEDEFAULTTABLE Procedure</td>
<td>Removes any default table attribute given for the element. After calling this procedure, the system generates tables names</td>
</tr>
<tr>
<td>REMOVEMAINHAINDOM Procedure</td>
<td>Removes all annotations used to maintain DOM from the given schema</td>
</tr>
<tr>
<td>REMOVEOUTOFLINE Procedure</td>
<td>Removes any existing SQLInline attributes to prevent out-of-line storage</td>
</tr>
<tr>
<td>REMOVESQLCOLLTYPE Procedure</td>
<td>Removes a SQL collection type.</td>
</tr>
<tr>
<td>REMOVESQLNAME Procedure</td>
<td>Removes a SQLNAME from a global element</td>
</tr>
<tr>
<td>REMOVESQLTYPE Procedure</td>
<td>Removes a SQL type</td>
</tr>
<tr>
<td>REMOVESQLTYPEPMAPPING Procedure</td>
<td>Removes the SQL type mapping for the given schema type.</td>
</tr>
<tr>
<td>REMOVETABLEPROPS Procedure</td>
<td>Removes the table storage properties from the CREATE TABLE statement</td>
</tr>
<tr>
<td>REMOVETIMESTAMPWITHTIMEZONE Procedure</td>
<td>Removes the setting of the TimeStampWithTimeZone datatype from all dateTime typed elements in the XML schema</td>
</tr>
<tr>
<td>SETANYSTORAGE Procedure</td>
<td>Assigns a SQL datatype to the ANY child of the complex type with the given name</td>
</tr>
<tr>
<td>SETDEFAULTTABLE Procedure</td>
<td>Sets the name of the table for the specified global element</td>
</tr>
<tr>
<td>SETOUTOFLINE Procedure</td>
<td>Sets the SQLInline attribute to FALSE</td>
</tr>
<tr>
<td>SETSCHEMAANNOTATIONS Procedure</td>
<td>Takes the annotated differences resulting from a call to DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS and patches them into the provided XML schema</td>
</tr>
<tr>
<td>SETSQLCOLLTYPE Procedure</td>
<td>Assigns a SQL type name for a collection</td>
</tr>
<tr>
<td>SETSQLNAME Procedure</td>
<td>Assigns a name to the SQL attribute that corresponds to an element defined in the XML schema</td>
</tr>
<tr>
<td>SETSQLTYPE Procedure</td>
<td>Assigns a SQL type to a global object</td>
</tr>
</tbody>
</table>
Table 207-1  (Cont.) DBMS_XMLSCHEMA_ANNOTATE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETSQLTYPEMAPPING Procedure</td>
<td>Defines a mapping of schema type and SQL type</td>
</tr>
<tr>
<td>SETTABLEPROPS Procedure</td>
<td>Specifies properties in the TABLE storage clause that is appended to the default CREATE TABLE statement</td>
</tr>
<tr>
<td>SETTIMESTAMPWITHTIMEZONE Procedure</td>
<td>Sets the TIMESTAMPWITHTIMEZONE datatype to all dateTime typed elements in the XML schema</td>
</tr>
</tbody>
</table>

### 207.3.1 ADDXDBNAMESPACE Procedure

This procedure adds the XDB namespace required for XDB annotation.

**Syntax**

```
DBMS_XMLSCHEMA_ANNOTATE.ADDXDBNAMESPACE (xmlschema IN OUT XMLTYPE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmschema</td>
<td>Gets an XML Schema as XMLTYPE, performs the annotation and returns it</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure is called implicitly by any other procedure that adds a schema annotation. Since there is no reason to add an XDB namespace without other annotations, this procedure is most likely called by other annotations procedures and not by the user directly.

### 207.3.2 DISABLEDEFAULTTABLECREATION Procedure

This procedure prevents the creation of a table for the top-level element by adding a default table attribute with an empty value to the element. The first overload applies to a specified top-level element and the second applies to all top-level elements. The procedure always overwrites. This is equivalent to using the schema annotation `xdb:defaultTable=""` for the top-level element or elements.

**Syntax**

```
DBMS_XMLSCHEMA_ANNOTATE.DISABLEDEFAULTTABLECREATION (xmlschema IN OUT XMLType,
                                                      globalElementName IN VARCHAR2);
DBMS_XMLSCHEMA_ANNOTATE.DISABLEDEFAULTTABLECREATION (xmlschema IN OUT XMLType);
```
Parameters

Table 207-3 DISABLEDEFAULTTABLECREATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element in the schema</td>
</tr>
</tbody>
</table>

Example

The *purchaseOrder* element will have an annotation similar to `xdb:defaultTable=""`.  

```sql
DECLARE
    xml_schema   XMLTYPE;
BEGIN
    SELECT out INTO xml_schema FROM annotation_tab;
    DBMS_XMLSCHEMA_ANNOTATE.DISABLEDEFAULTTABLECREATION(xml_schema,
        'purchaseOrder');
    UPDATE annotation_tab SET out = xml_schema;
END;
/
```

207.3.3 DISABLEMAINTAINDOM Procedure

This procedure sets the DOM fidelity attribute to `FALSE`.  

There are two overloads. The first sets DOM fidelity attribute to `FALSE` for all complex types, and the second sets it to `FALSE` for the named complex type. This is equivalent to adding `xdb:maintainDOM="false"` on all or specified complex types respectively.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.DISABLEMAINTAINDOM (  
    xmlschema       IN OUT XMLType,  
    overwrite       IN BOOLEAN default TRUE);  

DBMS_XMLSCHEMA_ANNOTATE.DISABLEMAINTAINDOM (  
    xmlschema       IN OUT XMLType,  
    complexTypeName IN VARCHAR2,  
    overwrite       IN BOOLEAN default TRUE);  
```

Parameters

Table 207-4 DISABLEMAINTAINDOM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated</td>
</tr>
<tr>
<td>complexTypeName</td>
<td>The name of the complex type</td>
</tr>
<tr>
<td>overwrite</td>
<td>A boolean that indicates whether or not the procedure overwrites element attributes. The default is <code>TRUE</code></td>
</tr>
</tbody>
</table>
207.3.4 ENABLEDEFAULTTABLECREATION Procedure

This procedure enables the creation of ALL top level tables by removing the empty default table name annotation.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.ENABLEDEFAULTTABLECREATION (    
    xmlschema            IN OUT  XMLTYPE);
```

```sql
DBMS_XMLSCHEMA_ANNOTATE.ENABLEDEFAULTTABLECREATION (    
    xmlschema            IN OUT  XMLTYPE,    
    globalElementName    IN      VARCHAR2);)
```

Parameters

Table 207-5 ENABLEDEFAULTTABLECREATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element in the schema</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure does not affect elements that have a default table name.

207.3.5 ENABLEMAINTAINDOM Procedure

This overloaded procedure sets the DOM fidelity attribute to TRUE.

There are two overloads. The first sets DOM fidelity attribute to TRUE for all complex types, and the second sets it to TRUE for the named complex type.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.ENABLEMAINTAINDOM (    
    xmlschema       IN OUT XMLType,    
    overwrite       IN BOOLEAN default TRUE);
```

```sql
DBMS_XMLSCHEMA_ANNOTATE.ENABLEMAINTAINDOM (    
    xmlschema        IN OUT XMLType,    
    complexTypeName  IN VARCHAR2,    
    overwrite        IN BOOLEAN default TRUE);
```

Parameters

Table 207-6 ENABLEMAINTAINDOM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated</td>
</tr>
<tr>
<td>complexTypeName</td>
<td>The name of the complex type</td>
</tr>
</tbody>
</table>
Table 207-6   (Cont.) ENABLEMAINTAINDOM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overwrite</td>
<td>A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE</td>
</tr>
</tbody>
</table>

207.3.6 GETSCHEMAANNOTATIONS Function

This function creates a document containing the differences between the annotated XML schema and the original XML schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS (
  xmlschema IN xmlType)
RETURN XMLType;
```

Parameters

Table 207-7   GETSCHEMAANNOTATIONS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The original XML schema</td>
</tr>
</tbody>
</table>

Return Values

This function returns the document `annotations.xml` as an `XMLType`.

Usage Notes

This function saves all annotations in one document, named `annotations`, and returns it. With this document, you can apply all annotations to a non-annotated schema, using `DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS`.

`DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS` is not available on Oracle Database release 10.2 (only Oracle Database release 11.x).

See Also:

- SETSCHEMAANNOTATIONS Procedure

Example

For an example of `DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS`, see the example in `SETSCHEMAANNOTATIONS Procedure`. 
207.3.7 GETSIDXDEFFROMVIEW Function

This function takes a XMLTABLE view definition on a xmltype column or table and it returns a CLOB which can be used as parameter to create a structured xmlindex that backs up the XMLTABLE view as relational table.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.GETSIDXDEFFROMVIEW (  
    viewName   IN   xmlType)  
RETURN CLOB;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewName</td>
<td>The original XML schema</td>
</tr>
</tbody>
</table>

Return Values

This function returns a CLOB which can be used as parameter to create a structured xmlindex that backs up the XMLTABLE view as relational table.

207.3.8 PRINTWARNINGS Procedure

This procedure lets a user raise or suppress a warning if an annotation maps to zero nodes in the XML schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.PRINTWARNINGS (  
    value       IN   BOOLEAN DEFAULT TRUE);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>For the NO MATCHING ELEMENTS FOUND error message to be raised, val must be set to TRUE. In cases in which user wishes to suppress this warning, set to FALSE.</td>
</tr>
</tbody>
</table>

Usage Notes

If an annotation maps to more than one node in the XML schema, this raise the error ANNOTATION MAPS TO MULTIPLE ELEMENTS. In this case no annotation is performed, and the user must correct the parameters to the procedure call to refer to a unique node in the XML schema.
207.3.9 REMOVEANYSTORAGE Procedure

This procedure removes the setting of the SQL type from the ANY child of the complex type with the given name.

Syntax

DBMS_XMLSCHEMA_ANNOTATE.REMOVEANYSTORAGE (xmlschema IN OUT XMLType, complexTypeName IN VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated.</td>
</tr>
<tr>
<td>complexTypeName</td>
<td>The name of the complex type.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure reverses the SETANYSTORAGE Procedure.

207.3.10 REMOVEDEFAULTTABLE Procedure

This procedure removes any default table attribute given for the element.

After calling this procedure, the system generates table names. This procedure always overwrites.

Syntax

DBMS_XMLSCHEMA_ANNOTATE.REMOVEDEFAULTTABLE (xmlschema IN OUT XMLTYPE, globalElementName IN VARCHAR2);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element in the schema</td>
</tr>
</tbody>
</table>

Example

Annotations can be verified anytime using "select out from annotation_tab".

```
--The purchaseOrder element will have no annotation for defaultTable.
DECLARE
xml_schema XMLTYPE;
```
BEGIN
  SELECT out INTO xml_schema FROM annotation_tab;
  DBMS_XMLSCHEMA_ANNOTATE.REMOVEDEFAULTTABLE(xml_schema,
    'purchaseOrder');
  UPDATE annotation_tab SET out = xml_schema;
END;
/

207.3.11 REMOVEMAIN TAINDOM Procedure

This procedure removes all annotations used to maintain DOM from the given schema.

Syntax

DBMS_XMLSCHEMA_ANNOTATE.REMOVEMAIN TAINDOM (  
xlschema IN OUT XMLType);

Parameters

Table 207-12 REMOVEMAIN TAINDOM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated</td>
</tr>
</tbody>
</table>

207.3.12 REMOVEOUTOFLINE Procedure

This procedure removes any existing SQLInline attributes to prevent out-of-line storage.

There are three overloads.

Syntax

Removes the SQLInline attribute for the named element.

DBMS_XMLSCHEMA_ANNOTATE.REMOVEOUTOFLINE (  
xlschema IN OUT XMLType,
  elementName IN VARCHAR2,
  elementType IN VARCHAR2,
  overwrite IN BOOLEAN default TRUE);

Removes the SQLInline attribute for the object specified by its global object and local element names.

DBMS_XMLSCHEMA_ANNOTATE.REMOVEOUTOFLINE (  
xlschema IN OUT XMLType,
  globalObject IN VARCHAR2,
  globalObjectName IN VARCHAR2,
  localElementName IN VARCHAR2);

Removes the SQLInline attribute for the referenced global element.

DBMS_XMLSCHEMA_ANNOTATE.REMOVEOUTOFLINE (  
xlschema IN OUT XMLType,
  reference IN VARCHAR2);
Parameters

Table 207-13 REMOVOUTOFLINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated</td>
</tr>
<tr>
<td>elementName</td>
<td>The element name</td>
</tr>
<tr>
<td>elementType</td>
<td>The element type</td>
</tr>
<tr>
<td>globalObject</td>
<td>The global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>The name of the global object</td>
</tr>
<tr>
<td>localElementName</td>
<td>The name of a local element that descends from the global element</td>
</tr>
<tr>
<td>reference</td>
<td>A reference to a global element</td>
</tr>
<tr>
<td>overwrite</td>
<td>A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure reverses SETOUTOFLINE Procedure.

207.3.13 REMOVESQLCOLLTYPE Procedure

This procedure removes a SQL collection type.

The first overload removes the SQL collection type corresponding to the named element and the second overload removes the type from the XML element inside the complex type.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLCOLLTYPE (  
    xmlschema  IN OUT XMLType,  
    elementName IN VARCHAR2);
```  
```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLCOLLTYPE (  
    xmlschema IN OUT XMLType,  
    globalObject IN VARCHAR2,  
    globalName  IN VARCHAR2,  
    localElementName IN VARCHAR2);
```  

Parameters

Table 207-14 REMOVESQLCOLLTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated</td>
</tr>
<tr>
<td>elementName</td>
<td>The element name</td>
</tr>
<tr>
<td>globalObject</td>
<td>The global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalName</td>
<td>The name of the global object</td>
</tr>
</tbody>
</table>
Table 207-14  (Cont.) REMOVESQLCOLLTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>localElementName</td>
<td>The name of a local element that descends from the global element</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure reverses the SETSQLCOLLTYPE Procedure.

### 207.3.14 REMOVESQLNAME Procedure

This procedure removes a SQLNAME from a global element.

**Syntax**

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLNAME (xmlschema          IN OUT  XMLType,
globalObject       IN      VARCHAR2,
globalObjectName   IN      VARCHAR2,
localObject        IN      VARCHAR2,
localObjectName    IN      VARCHAR2,
sqlName            IN      VARCHAR2,
overwrite          IN      BOOLEAN DEFAULT TRUE);
```

**Parameters**

Table 207-15  REMOVESQLNAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalObject</td>
<td>Global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>Name of the global object</td>
</tr>
<tr>
<td>localObject</td>
<td>Object descended from the global object</td>
</tr>
<tr>
<td>localObjectName</td>
<td>Name of the local object</td>
</tr>
<tr>
<td>sqlName</td>
<td>Name of the SQL attribute that corresponds to the element defined in the XML schema</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

**Example**

The shipTo element will have an annotation similar to `xdb:SQLName="SHIPTO_SQLNAME"`.  

```sql
DECLARE
    xml_schema XMLTYPE;
BEGIN
    SELECT out INTO xml_schema FROM annotation_tab;
    DBMS_XMLSCHEMA_ANNOTATE.SETSQLNAME (xml_schema,
        'element', 'purchaseOrder',
        'element', 'shipTo',
        'SHIPTO_SQLNAME');
```

---

**Chapter 207**

Summary of DBMS_XMLSCHEMA_ANNOTATE Subprograms
207.3.15 REMOVESQLTYPE Procedure

This procedure removes a SQL type.

The first overload removes a SQL type from a global element and the second overload removes the type from a global element inside the complex type.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLTYPE (  
    xmlschema in out XMLType,  
    globalElementName   IN      VARCHAR2);
```

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLTYPE (  
    xmlschema           IN OUT  XMLTYPE,  
    globalObject        IN      VARCHAR2,  
    globalObjectName    IN      VARCHAR2,  
    localObject         IN      VARCHAR2,  
    localObjectName     IN      VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated.</td>
</tr>
<tr>
<td>globalObject</td>
<td>Global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element.</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>Name of the global object</td>
</tr>
<tr>
<td>localObject</td>
<td>Object descended from the global object</td>
</tr>
<tr>
<td>localObjectName</td>
<td>Name of the local object</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure reverses the SETSQLTYPE Procedure.

207.3.16 REMOVESQLTYPEMAPPING Procedure

This procedure removes the SQL type mapping for the given schema type.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLTYPEMAPPING (  
    xmlschema        IN OUT  XMLTYPE,  
    schemaTypeName   IN      VARCHAR2);
```
Parameters

Table 207-17  REMOVESQLTYPEMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>schemaTypeName</td>
<td>Name of the schema type</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure reverses the SETSQLTYPEMAPPING Procedure.

207.3.17 REMOVETABLEPROPS Procedure

This procedure removes the table storage properties from the `CREATE TABLE` statement.

This procedure is overloaded. Each overload has different parameter requirements as indicated.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVETABLEPROPS ( 
    xmlschema            IN OUT  XMLTYPE, 
    globalElementName    IN      VARCHAR2);
```

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVETABLEPROPS ( 
    xmlschema            IN OUT  XMLTYPE, 
    globalObject         IN      VARCHAR2, 
    globalObjectName     IN      VARCHAR2, 
    localElementName     IN      VARCHAR2);
```

Parameters

Table 207-18  REMOVETABLEPROPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element in the schema</td>
</tr>
<tr>
<td>globalObject</td>
<td>Global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>Name of the global object</td>
</tr>
<tr>
<td>localElementName</td>
<td>Name of a local element that descends from the global element</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure reverses the SETTABLEPROPS Procedure.
207.3.18 REMOVETIMESTAMPWITHTIMEZONE Procedure

This procedure removes the setting of the `TimeStampWithTimeZone` datatype from all `dateTime` typed elements in the XML schema.

**Syntax**

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVETIMESTAMPWITHTIMEZONE (  
    xmlschema          IN OUT  XMLTYPE);
```

```sql
DBMS_XMLSCHEMA_ANNOTATE.REMOVETIMESTAMPWITHTIMEZONE (  
    xmlschema          IN OUT  XMLTYPE,  
    schemaTypeName    IN      VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>schemaTypeName</td>
<td>Name of the schema type</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure reverses the `SETTIMESTAMPWITHTIMEZONE Procedure`.

207.3.19 SETANYSTORAGE Procedure

This procedure assigns a SQL datatype to the `ANY` child of the complex type with the given name.

**Syntax**

```sql
DBMS_XMLSCHEMA_ANNOTATE.SETANYSTORAGE (  
    xmlschema       IN OUT  XMLType,  
    complexTypeName IN      VARCHAR2,  
    sqlTypeName     IN      VARCHAR2,  
    overwrite       IN      BOOLEAN DEFAULT TRUE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>complexTypeName</td>
<td>Name of the complex type</td>
</tr>
<tr>
<td>sqlTypeName</td>
<td>Name of the SQL type</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>
Example

The xsd:any child of complex type Items is assigned an annotation similar to xdb:SQLType="VARCHAR".

DECLARE xml_schema XMLTYPE;BEGIN SELECT out INTO xml_schema FROM annotation_tab; DBMS_XMLSCHEMA_ANNOTATE.setAnyStorage(xml_schema, 'Items', 'VARCHAR'); UPDATE annotation_tab SET out = xml_schema;END;

207.3.20 SETDEFAULTTABLE Procedure

This procedure sets the name of the table for the specified global element. This is equivalent to using the schema annotation xdb:defaultTable="<default_table_name>" for the top-level element.

Syntax

DBMS_XMLSCHEMA_ANNOTATE.SETDEFAULTTABLE {
  xmlschema IN OUT XMLTYPE,
globalElementName IN VARCHAR2,
tableName IN VARCHAR2,
overwrite IN BOOLEAN DEFAULT TRUE};

Parameters

Table 207-21 SETDEFAULTTABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element in the schema</td>
</tr>
<tr>
<td>tableName</td>
<td>Name being assigned to the table</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

207.3.21 SETOUTOFLINE Procedure

This procedure sets the SQLInline attribute to FALSE, that is, it sets xdb:SQLInLine=FALSE.

This forces XDB to store the corresponding elements in the XML document out-of-line as rows in a separate XMLType table. XDB stores references to each row of the XMLType table in a link table that is maintained by the main table.

This procedure can improve performance in some situations if the out-of-line table acts as the driver for the query. Storing elements in an out-of-line table also reduces the numbers of columns in the base table, thus avoiding '1000 column limit' errors during XML schema registration, when some elements have complex types with many elements.
Also See:

Oracle XML DB Developer’s Guide

There are three overloads.

Syntax

Sets the SQLInline attribute to FALSE, forcing out-of-line storage for the named element.

```sql
DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE (  
  xmlschema          IN OUT  XMLType,  
  elementName        IN      VARCHAR2,  
  elementType        IN      VARCHAR2,  
  defaultTableName   IN      VARCHAR2,  
  overwrite          IN      BOOLEAN DEFAULT TRUE);
```

Sets the SQLInline attribute to FALSE, forcing out-of-line storage for the element specified by its local and global name.

```sql
DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE (  
  xmlschema          IN OUT  XMLType,  
  globalObject       IN      VARCHAR2,  
  globalObjectName   IN      VARCHAR2,  
  localElementName   IN      VARCHAR2,  
  defaultTableName   IN      VARCHAR2,  
  overwrite          IN      BOOLEAN DEFAULT TRUE);
```

Sets the SQLInline attribute to FALSE to force out-of-line storage and sets the default table name for all references to a particular global element.

```sql
DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE (  
  xmlschema         IN OUT  XMLType,  
  reference         IN      VARCHAR2,  
  defaultTableName  IN      VARCHAR2,  
  overwrite         IN      BOOLEAN DEFAULT TRUE);
```

Parameters

Table 207-22  SETOUTOFLINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated.</td>
</tr>
<tr>
<td>elementName</td>
<td>The element name</td>
</tr>
<tr>
<td>elementType</td>
<td>The element type</td>
</tr>
<tr>
<td>defaultTableName</td>
<td>The name of the default table.</td>
</tr>
<tr>
<td>globalObject</td>
<td>The global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>The name of the global object.</td>
</tr>
<tr>
<td>localElementName</td>
<td>The name of a local element that descends from the global element.</td>
</tr>
<tr>
<td>reference</td>
<td>A reference to a global element.</td>
</tr>
</tbody>
</table>
Table 207-22  (Cont.) SETOUTOFLINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overwrite</td>
<td>A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

Usage Notes

After XML schema registration and before loading XML instance data, use DBMS_XMLSTORAGE_MANAGE.SCOPEXMLREFERENCES() to make these references scope to the out-of-line table only. This ensures better query performance later on.

Example

The following example illustrates the third overloaded method. The element comment will have an annotation similar to xdb:defaultTable="CMMNT_DEFAULT_TABLE"

```sql
DECLARE
    xml_schema xmltype;
BEGIN
    SELECT OUT INTO xml_schema FROM annotation_tab;
    DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE (xml_schema,
                                                       'ipo:comment',
                                                       'CMMNT_DEFAULT_TABLE');
    UPDATE annotation_tab SET OUT = xml_schema;
END;
/```

207.3.22 SETSCHEMAANNOTATATIONS Procedure

This procedure takes the annotated differences resulting from a call to DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS and patches them into the provided XML schema.

Syntax

```sql
DBMS_XMLSCHEMA_ANNOTATE.SETSCHEMAANNOTATIONS (  
    xmlschema     IN OUT xmlType,  
    annotations   IN VARCHAR2);
```

Parameters

Table 207-23  SETSCHEMAANNOTATIONS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>An XML schema to be patched.</td>
</tr>
<tr>
<td>annotations</td>
<td>The differences document produced by calling DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS on the original XML schema and an annotated XML schema.</td>
</tr>
</tbody>
</table>
Usage Notes

`DBMS_XMLSCHEMA_ANNOTATE.SETSCHEMAANNOTATIONS` is not available on Oracle Database release 10.2 (only Oracle Database release 11.x).

See Also:

- `GETSCHEMAANNOTATIONS` Function

Example

The following example illustrates `DBMS_XMLSCHEMA_ANNOTATE.SETSCHEMAANNOTATIONS` shown here and `GETSCHEMAANNOTATIONS` Function.

```
-- test getannotations and apply them
declare
    xml_schema xmltype;
    xml_schema2 xmltype;
    annotations xmltype;
begin
    select out into xml_schema from annotation_tab;
    -- get the annotations from the schema
    annotations := DBMS_XMLSCHEMA_ANNOTATE.getSchemaAnnotations (xml_schema);
    -- apply the annotations to the schema
    select inp into xml_schema2 from annotation_tab;
    DBMS_XMLSCHEMA_ANNOTATE.setSchemaAnnotations(xml_schema2, annotations);
    update annotation_tab t set t.out = xml_schema2;
end;
/```

207.3.23 SETSQLCOLLTYPE Procedure

This procedure assigns a SQL type name for a collection. A collection is a global or local element with `maxOccurs>1`.

Using this procedure, XDB creates SQLTypes with the user-defined names provided.

There are two overloads. The first sets the name of the SQL collection type corresponding to an XML element and the second to an XML element inside the specified complex type.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLCOLLTYPE (xmlschema IN OUT XMLTYPE,
    elementName IN VARCHAR2,
    sqlCollType IN VARCHAR2,
    overwrite IN BOOLEAN DEFAULT TRUE);
```

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLCOLLTYPE (xmlschema IN OUT XMLType,
    globalObject IN VARCHAR2,
    sqlCollType IN VARCHAR2,
    overwrite IN BOOLEAN DEFAULT TRUE);
```
globalObjectName IN VARCHAR2,
localElementName IN VARCHAR2,
sqlCollType IN VARCHAR2,
overwrite IN BOOLEAN default TRUE );

Parameters

Table 207-24 SETSQLCOLLTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>The XML schema to be annotated</td>
</tr>
<tr>
<td>elementName</td>
<td>The element name</td>
</tr>
<tr>
<td>sqlCollType</td>
<td>The SQL collection type</td>
</tr>
<tr>
<td>globalObject</td>
<td>The global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>The name of the global object</td>
</tr>
<tr>
<td>localElementName</td>
<td>The name of a local element that descends from the global element</td>
</tr>
<tr>
<td>overwrite</td>
<td>A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

Example

The *item* element will have an annotation similar to `xdb:SQLCollType="ITEM_SQL_COL_TYPE"`.

declare
   xml_schema xmltype;
begin
   SELECT out INTO xml_schema FROM annotation_tab;
   DBMS_XMLSCHEMA_ANNOTATE.setSQLCollType (xml_schema,
      'item',
      'ITEM_SQL_COL_TYPE',TRUE);

   UPDATE annotation_tab SET out = xml_schema;
end;

207.3.24 SETSQLNAME Procedure

This procedure assigns a name to the SQL attribute that corresponds to an element defined in the XML schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLNAME ( 
   xmlschema IN OUT XMLType,
   globalObject IN VARCHAR2,
   globalObjectName IN VARCHAR2,
   localObject IN VARCHAR2,
   localObjectName IN VARCHAR2,
   sqlName IN VARCHAR2,
   overwrite IN BOOLEAN DEFAULT TRUE);
```
**Parameters**

**Table 207-25  SETSQLNAME Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalObject</td>
<td>Global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>Name of the global object</td>
</tr>
<tr>
<td>localObject</td>
<td>Object descended from the global object</td>
</tr>
<tr>
<td>localObjectName</td>
<td>Name of the local object</td>
</tr>
<tr>
<td>sqlName</td>
<td>Name of the SQL attribute that corresponds to the element defined in the XML schema</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

**Example**

The `shipTo` element will have an annotation similar to `xdb:SQLName="SHIPTO_SQLNAME"`.

```sql
DECLARE
    xml_schema XMLTYPE;
BEGIN
    SELECT out INTO xml_schema FROM annotation_tab;
    DBMS_XMLSCHEMA_ANNOTATE.SETSQLNAME (xml_schema,
        'element', 'purchaseOrder',
        'element', 'shipTo',
        'SHIPTO_SQLNAME');
    UPDATE annotation_tab SET out = xml_schema;
END;
/
```

### 207.3.25 SETSQLTYPE Procedure

This procedure assigns a SQL type to a global object.

There are two overloads. The first overload assigns a SQL Type to a global object, such as a global element or global complex type and the second to a local object.

**Syntax**

```sql
DBMS_XMLSCHEMA_ANNOTATE.SETSQLTYPE (xmlschema IN OUT XMLTYPE,
    globalElementName IN VARCHAR2,
    sqlType IN VARCHAR2,
    overwrite IN BOOLEAN DEFAULT TRUE);
```

```sql
DBMS_XMLSCHEMA_ANNOTATE.SETSQLTYPE (xmlschema IN OUT XMLTYPE,
    globalObject IN VARCHAR2,
    globalObjectName IN VARCHAR2,
    localObject IN VARCHAR2,
    localObjectName IN VARCHAR2,
    sqlType IN VARCHAR2);```
sqlType             IN       VARCHAR2,
overwrite           IN       BOOLEAN DEFAULT TRUE);

Parameters

Table 207-26  SETSQLTYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalObject</td>
<td>Global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>Name of the global object</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element</td>
</tr>
<tr>
<td>localObject</td>
<td>Object descended from the global object</td>
</tr>
<tr>
<td>localObjectName</td>
<td>Name of the local object</td>
</tr>
<tr>
<td>sqlType</td>
<td>SQL type assigned to the named global element</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

Example

The `purchaseOrder` element will have an annotation similar to `xdb:SQLType="PO_SQLTYPE"` and the `shipTo` element has one similar to `xdb:SQLType="VARCHAR"`.

```
DECLARE
  xml_schema xmltype;
BEGIN
  SELECT out INTO xml_schema FROM annotation_tab;
  DBMS_XMLSCHEMA_ANNOTATE.setSQLType (xml_schema,
                                     'purchaseOrder',
                                     'PO_SQLTYPE');
  UPDATE annotation_tab SET out = xml_schema;
END;
/

DECLARE   xml_schema   xmltype;BEGIN  SELECT out INTO xml_schema FROM annotation_tab;
DBMS_XMLSCHEMA_ANNOTATE.setSQLType
(xml_schema,
  'element', 'purchaseOrder',
  'element', 'shipTo',
  'VARCHAR');  UPDATE
annotation_tab SET out = xml_schema;END;
/
```

207.3.26 SETSQLTYPEMAPPING Procedure

This procedure defines a mapping of schema type and SQL type.

If you use this procedure, you do not need to call the `SETSQLTYPE` procedure on all instances of the schema type; instead the procedure traverses the schema and assigns the SQL type automatically.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLTYPEMAPPING ( xmlschema IN OUT XMLType,
```

```
Parameters

Table 207-27 SETSQLTYPEMAPPING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>schemaTypeName</td>
<td>Schema type</td>
</tr>
<tr>
<td>sqlTypeName</td>
<td>Name of the SQL type</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE</td>
</tr>
</tbody>
</table>

Example

The attribute orderDate will have an annotation similar to xdb:SQLType="DATE".

declare xml_schema xmltype;beginSELECT out INTO xml_schema FROM annotation_tab;DBMS_XMLSCHEMA_ANNOTATE.setSQLTypeMapping (xml_schema, 'date', 'DATE');UPDATE annotation_tab SET out = xml_schema;end;
/

207.3.27 SETTABLEPROPS Procedure

This procedure specifies properties in the TABLE storage clause that is appended to the default CREATE TABLE statement.

There are two overloads with different parameter requirements, as indicated:

Syntax

DBMS_XMLSCHEMA_ANNOTATE.SETTABLEPROPS {  
xmlschema IN OUT XMLType,  
globalElementName IN VARCHAR2,  
tableProps IN VARCHAR2,  
overwrite IN BOOLEAN DEFAULT TRUE};

DBMS_XMLSCHEMA_ANNOTATE.SETTABLEPROPS {  
xmlelement IN OUT XMLTYPE,  
globalObject IN VARCHAR2,  
globalObjectName IN VARCHAR2,  
localElementName IN VARCHAR2,  
tableProps IN VARCHAR2,  
overwrite IN BOOLEAN DEFAULT TRUE};
Parameters

Table 207-28  SETTABLEPROPS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>globalElementName</td>
<td>Name of the global element in the schema</td>
</tr>
<tr>
<td>tableProps</td>
<td>Table properties</td>
</tr>
<tr>
<td>globalObject</td>
<td>Global object (global complex type or global element)</td>
</tr>
<tr>
<td>globalObjectName</td>
<td>Name of the global object</td>
</tr>
<tr>
<td>localElementName</td>
<td>Name of a local element that descends from the global element</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>

Example

The `purchaseOrder` element will have an annotation similar to `xdb:tableProps="CACHE"`.

```sql
DECLARE   xml_schema XMLTYPE;
BEGIN   SELECT out INTO xml_schema FROM annotation_tab;
DBMS_XMLSCHEMA_ANNOTATE.SETTABLEPROPS(xml_schema,
    'purchaseOrder','CACHE');   UPDATE annotation_tab SET out = xml_schema;END;
/
```

207.3.28 SETTIMESTAMPWITHTIMEZONE Procedure

This procedure sets the `TIMESTAMPWITHTIMEZONE` datatype to all `dateTime` typed elements in the XML schema.

This is equivalent to adding `xdb:SQLType="TIMESTAMP WITH TIME ZONE"` to all `dateTime` objects.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETTIMESTAMPWITHTIMEZONE (   xmlschema IN OUT XMLTYPE,   overwrite IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 207-29  SETTIMESTAMPWITHTIMEZONE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlschema</td>
<td>XML schema to be annotated</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.</td>
</tr>
</tbody>
</table>
The DBMS_XMLSTORAGE_MANAGE package provides an interface to manage and modify XML storage after schema registration has been completed.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_XMLSTORAGE_MANAGE Subprograms

**See Also:**

Oracle XML DB Developer's Guide

### 208.1 DBMS_XMLSTORAGE_MANAGE Overview

DBMS_XMLSTORAGE_MANAGE contains procedures to manage and modify XML storage after schema registration has been completed.

Use subprograms from this package to improve the performance of bulk load operations. You can disable indexes and constraints before doing a bulk load process and to enable them afterwards.

### 208.2 DBMS_XMLSTORAGE_MANAGE Security Model

Owned by XDB, the DBMS_XMLSTORAGE_MANAGE package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC.

Subprograms in this package are executed using the privileges of the current user.

### 208.3 Summary of DBMS_XMLSTORAGE_MANAGE Subprograms

This table lists and describes the DBMS_XMLSTORAGE_MANAGE package subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLEINDEXESANDCONSTRAINTS Procedure</td>
<td>Disables the indexes and constraints for XMLType tables and XMLType columns</td>
</tr>
</tbody>
</table>
Table 208-1  (Cont.) DBMS_XMLSTORAGE_MANAGE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLEINDEXESANDCONSTRAINTS Procedure</td>
<td>Rebuilds all indexes and enables the constraints on an XMLType table including its child tables and out-of-line tables</td>
</tr>
<tr>
<td>EXCHANGEPOSTPROC Procedure</td>
<td>Enable constraints after exchange partition</td>
</tr>
<tr>
<td>EXCHANGEPREPROC Procedure</td>
<td>Disable constraints before exchange partition</td>
</tr>
<tr>
<td>INDEXXMLREFERENCES Procedure</td>
<td>Creates unique indexes on the REF columns of the given XML type table or the XML type column of a given table</td>
</tr>
<tr>
<td>REFPARTITIONEXCHANGEIN Procedure</td>
<td>This is an auxiliary procedure to load data through exchange partition operation into a partitioned table and its reference-partitioned child table provided the child table has an xmltype column with a local xmlindex.</td>
</tr>
<tr>
<td>REFPARTITIONEXCHANGEOUT Procedure</td>
<td>Auxiliary procedure to load data through exchange partition operation out of a partitioned table and its reference-partitioned child table provided that the child table has an xmltype column with a local xmlindex.</td>
</tr>
<tr>
<td>RENAMECOLLECTIONTABLE Procedure</td>
<td>Renames a collection table to the given table name</td>
</tr>
<tr>
<td>SCOPEXMLREFERENCES Procedure</td>
<td>Scopes all XML references. Scoped REF types require less storage space and allow more efficient access than unscoped REF types</td>
</tr>
<tr>
<td>XPATH2TABCOLMAPPING Function</td>
<td>Maps a path expression (in XPath notation or DOT notations) to the corresponding table name and column name</td>
</tr>
</tbody>
</table>

208.3.1 DISABLEINDEXESANDCONSTRAINTS Procedure

This procedure disables the indexes and constraints for XMLType tables and XMLType columns.

Syntax

```sql
DBMS_XMLSTORAGE_MANAGE.DISABLEINDEXESANDCONSTRAINTS (    owner_name IN VARCHAR2 DEFAULT USER,    table_name IN VARCHAR2,    column_name IN VARCHAR2 DEFAULT NULL,    clear IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 208-2  DISABLEINDEXESANDCONSTRAINTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>Owner's name</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the XMLType table that the procedure is being performed on</td>
</tr>
<tr>
<td>column_name</td>
<td>XMLType column name</td>
</tr>
</tbody>
</table>
### Table 208-2  (Cont.) DISABLEINDEXESANDCONSTRAINTS Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Boolean that when set to TRUE clears all stored index and constraint data for the table before the procedure executes. The default is FALSE, which does not clear them.</td>
</tr>
</tbody>
</table>

### Usage Notes

#### Passing XMLTYPE tables

For XMLType tables, you must pass the XMLType table name on which the bulk load operation is to be performed. For XMLType columns, you must pass the relational table name and the corresponding XMLType column name.

#### Using clear to Enable and Disable Indexes and Constraints

**Note:**

If the DISABLEINDEXESANDCONSTRAINTS procedure is called with clear set to TRUE, it removes any index or constraint information about the XMLTYPE table or column memorized during earlier executions of the procedure.

Therefore, you must ensure that all disabled indexes and constraints are re-enabled on the table or column before you call the DISABLEINDEXESANDCONSTRAINTS procedure with clear set to TRUE.

Ideally, it is recommended that you set clear set to TRUE for the first execution. For any subsequent executions (due to errors while disabling or enabling indexes) clear should be set to FALSE, the default value. Once you have successfully re-enabled all the indexes and constraints following the bulk load operation, you can call this procedure again with clear set to TRUE for the next bulk load operation.

#### Example

The following example illustrates the use of clear in the DISABLEINDEXESANDCONSTRAINTS procedure and the ENABLEINDEXESANDCONSTRAINTS Procedure.

First, add a not-NULL constraint on comment element of the PURCHASEORDER_TAB table:

```sql
ALTER TABLE PURCHASEORDER_TAB ADD CONSTRAINT c1 check            ("XMLDATA"."comment" IS NOT NULL);
```

Then, disable all the indexes and constraints by passing the clear as TRUE, by calling the DISABLEINDEXESANDCONSTRAINTS procedure:

```sql
BEGIN
    XDB.DBMS_XMLSTORAGE_MANAGE.DISABLEINDEXESANDCONSTRAINTS ( USER,'PURCHASEORDER_TAB',NULL,TRUE );
END;
/
```
Next, perform a bulk load operation (such as datapump import) which violates constraint c1 in the ALTER table statement. This does not raise an error because the constraint is disabled:

```sql
host impdp orexample/orexample directory=dir dumpfile=dmp.txt
tables=OREXAMPLE.PURCHASEORDER_TAB content = DATA_ONLY;
```

**NOTE:** To view the disabled constraints and indexes use:

```sql
SELECT constraint_name,table_name,status FROM all_constraints
WHERE owner = user;
```

Finally, try to enable the constraint using the `ENABLEINDEXESANDCONSTRAINTS` procedure. It raises an error because c1, the not null constraint, is violated by the bulk load operation:

```sql
BEGIN
   xdb.DBMS_XMLSTORAGE_MANAGE.ENABLEINDEXESANDCONSTRAINTS
   ( USER,'PURCHASEORDER_TAB');
END;
/
```

To disable all the indexes and constraints, again use `DISABLEINDEXESANDCONSTRAINTS`, but set `clear= FALSE (because the ENABLEINDEXESANDCONSTRAINTS failed to complete successfully). Note: clear = FALSE by default, so we do not need to pass it explicitly in the next call.

```sql
BEGIN
   xdb.DBMS_XMLSTORAGE_MANAGE.DISABLEINDEXESANDCONSTRAINTS
   ( USER,'PURCHASEORDER_TAB');
END;
/
```

Then, delete the incorrect rows entered into the table

```sql
DELETE FROM purchaseorder_tab p
WHERE p.xmldata."comment" IS NULL;
```

Re-enable the indexes and constraints using `ENABLEINDEXESANDCONSTRAINTS`, which completes successfully.

```sql
BEGIN
   xdb.DBMS_XMLSTORAGE_MANAGE.ENABLEINDEXESANDCONSTRAINTS
   ( USER,'PURCHASEORDER_TAB');
END;
/
```

## 208.3.2 ENABLEINDEXESANDCONSTRAINTS Procedure

This procedure rebuilds all indexes and enables the constraints on an XMLType table including its child tables and out-of-line tables.

When `column_name` is passed, it does the same for this XMLType column.
Syntax

DBMS_XMLSTORAGE_MANAGE.ENABLEINDEXESANDCONSTRAINTS {
    owner_name   IN VARCHAR2 DEFAULT USER,
    table_name   IN VARCHAR2,
    column_name  IN VARCHAR2 DEFAULT NULL);

Parameters

Table 208-3  ENABLEINDEXESANDCONSTRAINTS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_user</td>
<td>Owner’s name</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table that the indexes and constraints are being removed from</td>
</tr>
<tr>
<td>column_name</td>
<td>Column name</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure reverses DISABLEINDEXESANDCONSTRAINTS Procedure.

Example

See DISABLEINDEXESANDCONSTRAINTS Procedure

208.3.3 EXCHANGEPOSTPROC Procedure

This procedure enable constraints after exchange partition.

Syntax

DBMS_XMLSTORAGE_MANAGE.EXCHANGEPOSTPROC {
    owner_name   IN VARCHAR2 DEFAULT USER,
    table_name   IN VARCHAR2);

Parameters

Table 208-4  EXCHANGEPOSTPROC Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_user</td>
<td>Owner’s name</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table that the indexes and constraints are being removed from</td>
</tr>
</tbody>
</table>
208.3.4 EXCHANGEPREPROC Procedure

This procedure disable constraints before exchange partition.

Syntax

```sql
DBMS_XMLSTORAGE_MANAGE.EXCHANGEPREPROC (  
    owner_name   IN VARCHAR2 DEFAULT USER,  
    table_name   IN VARCHAR2);
```

Parameters

Table 208-5  EXCHANGEPREPROC Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_user</td>
<td>Owner’s name</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the table that the indexes and constraints are being removed from</td>
</tr>
</tbody>
</table>

208.3.5 INDEXXMLREFERENCES Procedure

This procedure creates unique indexes on the REF columns of the given XML type table or the XML type column of a given table.

If the procedure creates multiple REF columns, it appends _1, _2, and so on to their names.

Syntax

```sql
DBMS_XMLSTORAGE_MANAGE.INDEXXMLREFERENCES (  
    owner_name    IN VARCHAR2 DEFAULT USER,  
    table_name    IN VARCHAR2,  
    column_name   IN VARCHAR2 DEFAULT NULL,  
    index_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 208-6  INDEXXMLREFERENCES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>The owner’s name</td>
</tr>
<tr>
<td>table_name</td>
<td>The table being indexed</td>
</tr>
<tr>
<td>column_name</td>
<td>A column name. Not needed for XML type tables.</td>
</tr>
<tr>
<td>index_name</td>
<td>The name of the newly created index</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure is only used if the REFS are scoped. See SCOPEXMLREFERENCES Procedure.

Indexed REFS lead to better performance when joins between the base table and a child table occur in the query plan.
• If the base table has a higher selectivity than the child table, there is no need to index the `REFs`.
• If the selectivity of the child table is higher than that of the base table and if no indexes are present, then the join of one row in the child table with the base table leads to a full table scan of the base table.

`INDEXXML` does not index `REFs` recursively in child tables of a table it is called on. To do this, Oracle recommends calling the procedure from within a loop over the `XML_OUT_OF_LINE_TABLES` or `XML_NESTED_TABLES` view. This creates the index names from the current value of a column in the view.

![Note:]
This procedure is limited to the structured storage model.

### 208.3.6 REFPARTITIONEXCHANGEIN Procedure

This is an auxiliary procedure to load data through exchange partition operation into a partitioned table and its reference-partitioned child table provided that the child table has an `xmltype` column with a local `xmlindex`.

**Syntax**

```
DBMS_XMLSTORAGE_MANAGE.REFPARTITIONEXCHANGEIN (  
    owner_name                 IN VARCHAR2,  
    parent_table_name          IN VARCHAR2,  
    child_table_name           IN VARCHAR2,  
    parent_exchange_table_name IN VARCHAR2,  
    child_exchange_table_name  IN VARCHAR2,  
    parent_exchange_stmt       IN CLOB,  
    child_exchange_stmt        IN CLOB);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>owner's name</td>
</tr>
<tr>
<td>parent_table_name</td>
<td>the partitioned base table</td>
</tr>
<tr>
<td>child_table_name</td>
<td>a partitioned table with reference partitioning based on the table named <code>parent_table_name</code></td>
</tr>
<tr>
<td>parent_exchange_table_name</td>
<td>an exchange table for the partitioned base table</td>
</tr>
<tr>
<td>child_exchange_table_name</td>
<td>an exchange table for the table named <code>child_table_name</code></td>
</tr>
<tr>
<td>parent_exchange_stmt</td>
<td>SQL statement to execute exchange partition operation between the table named <code>parent_table_name</code> and the table named <code>parent_exchange_table_name</code></td>
</tr>
</tbody>
</table>
Table 208-7 (Cont.) REFPARTITIONEXCHANGEIN Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>child_exchange_stmt</td>
<td>SQL statement to execute exchange partition operation between the table named child_table_name and the table named child_exchange_table_name</td>
</tr>
</tbody>
</table>

208.3.7 REFPARTITIONEXCHANGEOUT Procedure

This is an auxiliary procedure to load data through exchange partition operation out of a partitioned table and its reference-partitioned child table provided that the child table has an xmltype column with a local xmlindex.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.REFPARTITIONEXCHANGEOUT (  
    owner_name                IN VARCHAR2,  
    parent_table_name         IN VARCHAR2,  
    child_table_name          IN VARCHAR2,  
    parent_exchange_table_name IN VARCHAR2,  
    child_exchange_table_name  IN VARCHAR2,  
    parent_exchange_stmt      IN CLOB,  
    child_exchange_stmt       IN CLOB);
```

Parameters

Table 208-8 REFPARTITIONEXCHANGEOUT Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>owner’s name</td>
</tr>
<tr>
<td>parent_table_name</td>
<td>the partitioned base table</td>
</tr>
<tr>
<td>child_table_name</td>
<td>a partitioned table with reference partitioning based on the table named parent_table_name</td>
</tr>
<tr>
<td>parent_exchange_table_name</td>
<td>an exchange table for the partitioned base table</td>
</tr>
<tr>
<td>child_exchange_table_name</td>
<td>an exchange table for the table named child_table_name</td>
</tr>
<tr>
<td>parent_exchange_stmt</td>
<td>SQL statement to execute exchange partition operation between the table named parent_table_name and the table named parent_exchange_table_name</td>
</tr>
<tr>
<td>child_exchange_stmt</td>
<td>SQL statement to execute exchange partition operation between the table named child_table_name and the table named child_exchange_table_name</td>
</tr>
</tbody>
</table>
208.3.8 RENAMECOLLECTIONTABLE Procedure

This procedure renames a collection table to the given table name.

An XPath expression specifies the collection table, starting from the XMLtype base table or an XMLType column of the base table.

This procedure provides the only way to derive a collection table name from the corresponding collection type name because there is no direct schema annotation for the purpose.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.RENAMECOLLECTIONTABLE (
  owner_name             IN VARCHAR2 DEFAULT USER,
  table_name             IN VARCHAR2,
  column_name            IN VARCHAR2 DEFAULT NULL,
  xpath                  IN VARCHAR2,
  collection_table_name  IN VARCHAR2,
  namespaces             IN VARCHAR2 default NULL);  // For release 11.2 only
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_name</td>
<td>The name of the owner</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of a base table that can be used as the starting point for specifying the collection table</td>
</tr>
<tr>
<td>column_name</td>
<td>An XMLType column that can be the starting point for specifying the collection table</td>
</tr>
<tr>
<td>xpath</td>
<td>The XPath expression that specifies the collection table</td>
</tr>
<tr>
<td>collection_table_name</td>
<td>The name of the collection table</td>
</tr>
<tr>
<td>namespaces</td>
<td>For Oracle Database 11g Release 2 (11.2) and higher. The namespaces used in XPath.</td>
</tr>
</tbody>
</table>

Usage Notes

Call this procedure after registering the XML schema.

The table name serves as a prefix to the index names.

Oracle recommends using this function because it makes query execution plans more readable.

Report errors that occur while this procedure runs to the user that called the procedure.

`Note:`

This procedure is limited to the structured storage model.
For Oracle Database 11g Release 2 (11.2) and higher, only, this function accepts XPath notation as well as DOT notation. If XPath notation is used, a namespaces parameter may also be required.

Example

The collection table name will be EMP_TAB_NAMELIST. You can verify this using SELECT * FROM user_nested_tables.

Using DOT Notation:

call XDB.DBMS_XMLSTORAGE_MANAGE.RENAMECOLLECTIONTABLE (USER, 'EMP_TAB', NULL, '"XMLDATA"."EMPLOYEE"."NAME"', 'EMP_TAB_NAMELIST');

Using XPath Notation:

XPath notation is available with Oracle Database 11g Release 2 (11.2) and higher.

call XDB.DBMS_XMLSTORAGE_MANAGE.RENAMECOLLECTIONTABLE (USER, 'EMP_TAB', NULL, '/e:Employee/Name', 'EMP_TAB_NAMELIST', 'http://www.oracle.com/emp.xsd' as "e");

208.3.9 SCOPEXMLREFERENCES Procedure

This procedure scopes all XML references. Scoped REF types require less storage space and allow more efficient access than unscoped REF types.

Syntax

DBMS_XMLSTORAGE_MANAGE.SCOPESXMLREFERENCES;

Usage Notes

• If you have used SETOUTOFLINE Procedure in the DBMS_XMLSTORAGE_MANAGE package to avoid raising '1000 column limit' errors during XML schema registration, you should also use SCOPEXMLREFERENCES Procedure.

• Using SCOPEXMLREFERENCES after XML schema registration and before loading XML instance data, makes these reference scoped to the out-of-line table only.

Note:

This procedure is limited to the structured storage model.
208.3.10 XPATH2TABCOLMAPPING Function

This function maps a path expression (in XPath notation or DOT notations) to the corresponding table name and column name. This is necessary in cases in which the user wants to create an index on this table, or to add a constraint, or to rename a table to make query execution plans more readable.

Syntax

```sql
DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING (  
    owner_name   IN  VARCHAR2 DEFAULT USER,  
    table_name   IN  VARCHAR2,  
    column_name  IN  VARCHAR2 DEFAULT NULL,  
    xpath        IN  VARCHAR2,  
    namespaces   IN  VARCHAR2 DEFAULT NULL)  
RETURN XMLTYPE;
```

Parameters

**Table 208-10  XPATH2TABCOLMAPPING Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_user</td>
<td>Owner’s name</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of the base table</td>
</tr>
<tr>
<td>column_name</td>
<td>Optional name of the XML type column if table_name is not an XMLType table. If table_name refers to XML Type table then column_name should be NULL.</td>
</tr>
<tr>
<td>xpath</td>
<td>Path expression in DOT notation or XPath notation (see examples below)</td>
</tr>
<tr>
<td>namespaces</td>
<td>Optional namespace definitions for path expression</td>
</tr>
</tbody>
</table>

Examples

**XPath2TabColMapping evaluated on XMLType table with Xpath Notation, namespaces provided**

```sql
SELECT XDB.DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING (  
    USER,  'XML_TAB',  '',  '/n1:item/n1:location','''xdbXmark'' as "n1")  
FROM DUAL;
```

This produces a result, for example:

```xml
<Result>
  <Mapping TableName="SYS_NT12345" ColumnName="location"/>
</Result>
```

This allows us to define an index or constraint on table SYS_NT12345 and column location.

**XPath2TabColMapping evaluated on table not of XMLType but with XMLType column by means of DOT notation**

```sql
SELECT XDB.DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING (  
    USER,'PurchaseOrderTab','XMLCOL','xmldata.LineItems.LineItem', '')  
FROM DUAL;
```
DBMS_XMLSTORE provides the ability to store XML data in relational tables.

This chapter contains the following sections:

- Security Model
- Types
- Summary of DBMS_XMLSTORE Subprograms

See Also:
Oracle XML DB Developer's Guide

209.1 DBMS_XMLSTORE Security Model

Owned by XDB, the DBMS_XMLSTORE package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

209.2 Types

The DBMS_XMLSTORE subprograms use the ctxType Type.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxType</td>
<td>The type of the query context handle. This is the return type of NEWCONTEXT.</td>
</tr>
</tbody>
</table>

209.3 Summary of DBMS_XMLSTORE Subprograms

This table lists the DBMS_XMLSTORE subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEARKEYCOLUMNLIST</td>
<td>Clears the key column list.</td>
</tr>
<tr>
<td>CLEARUPDATECOLUMNLIST</td>
<td>Clears the update column list.</td>
</tr>
<tr>
<td>CLOSECONTEXT</td>
<td>It closes/deallocates a particular save context.</td>
</tr>
</tbody>
</table>
Table 209-2  (Cont.) DBMS_XMLSTORE Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETEXML</td>
<td>Deletes records specified by data from the XML document, from the table specified at the context creation time.</td>
</tr>
<tr>
<td>INSERTXML</td>
<td>Inserts the XML document into the table specified at the context creation time.</td>
</tr>
<tr>
<td>NEWCONTEXT</td>
<td>Creates a save context, and returns the context handle.</td>
</tr>
<tr>
<td>SETKEYCOLUMN</td>
<td>This method adds a column to the key column list.</td>
</tr>
<tr>
<td>SETROWTAG</td>
<td>Names the tag used in the XML document, to enclose the XML elements corresponding to the database.</td>
</tr>
<tr>
<td>SETUPDATECOLUMN</td>
<td>Adds a column to the &quot;update column list&quot;.</td>
</tr>
<tr>
<td>UPDATEXML</td>
<td>Updates the table given the XML document.</td>
</tr>
</tbody>
</table>

209.3.1 CLEARKEYCOLUMNLIST

This procedure clears the key column list.

Syntax

PROCEDURE clearKeyColumnList(
    ctxHdl IN ctxType);

Table 209-3  CLEARKEYCOLUMNLIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>

209.3.2 CLEARUPDATECOLUMNLIST

This procedure clears the update column list.

Syntax

PROCEDURE clearUpdateColumnList(
    ctxHdl IN ctxType);

Table 209-4  CLEARUPDATECOLUMNLIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>
209.3.3 CLOSECONTEXT

This procedure closes/deallocates a particular save context.

Syntax

PROCEDURE closeContext(ctxHdl IN ctxType);

Table 209-5  CLOSECONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>

209.3.4 DELETEXML

DELETEXML deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted.

Syntax

The following syntax uses a VARCHAR2 type for the xDoc parameter.

FUNCTION deleteXML(
  ctxHdl IN ctxPType,
  xDoc IN VARCHAR2)
RETURN NUMBER;

The following syntax uses a CLOB type for the xDoc parameter.

FUNCTION deleteXML(
  ctxHdl IN ctxType,
  xDoc IN CLOB)
RETURN NUMBER;

The following syntax uses an XMLType type for the xDoc parameter.

FUNCTION deleteXML(
  ctxHdl IN ctxType,
  xDoc IN XMLType)
RETURN NUMBER;

Parameters

Table 209-6  DELETEXML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>
Table 209-6  (Cont.) DELETEXML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xDoc</td>
<td>(IN)</td>
<td>String containing the XML document.</td>
</tr>
</tbody>
</table>

209.3.5 INSERTXML

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted.

Note that if a user passes an XML file for `INSERTXML` to `DBMS_XMLSTORE` that contains extra elements (elements that do not match any columns in the table), Oracle tries to insert into those columns unless `SEL_UPDATECOLUMN` is used. The use of `setUpdateColumn` is optional only if the elements in the XML file match up to the columns in the table.

Syntax

```sql
FUNCTION insertXML(
    ctxHdl IN ctxType,
    xDoc IN VARCHAR2)
RETURN NUMBER;
```

```sql
FUNCTION insertXML(
    ctxHdl IN ctxType,
    xDoc IN CLOB)
RETURN NUMBER;
```

```sql
FUNCTION insertXML(
    ctxHdl IN ctxType,
    xDoc IN XMLType)
RETURN NUMBER;
```

Parameters

Table 209-7  INSERTXML Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>xDoc</td>
<td>(IN)</td>
<td>String containing the XML document.</td>
</tr>
</tbody>
</table>
209.3.6 NEWCONTEXT

NEWCONTEXT creates a save context and returns the context handle.

Syntax

FUNCTION newContext(
    targetTable IN VARCHAR2)
RETURN ctxType;

Table 209-8   NEWCONTEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetTable</td>
<td>(IN)</td>
<td>The target table into which to load the XML document.</td>
</tr>
</tbody>
</table>

209.3.7 SETKEYCOLUMN

This method adds a column to the "key column list".

The value for the column cannot be NULL. In case of update or delete, the columns in the key column list make up the WHERE clause of the statement. The key columns list must be specified before updates can complete; this is optional for delete operations.

Syntax

PROCEDURE setKeyColumn(
    ctxHdl IN ctxType,
    colName IN VARCHAR2);

Table 209-9   SETKEYCOLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>colName</td>
<td>(IN)</td>
<td>Column to be added to the key column list; cannot be NULL.</td>
</tr>
</tbody>
</table>

209.3.8 SETROWTAG

This procedure names the tag used in the XML document, to enclose the XML elements corresponding to database records.

Syntax

PROCEDURE setRowTag(
    ctxHdl IN ctxType,
    tag IN VARCHAR2);

Table 209-10   SETROWTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
</tbody>
</table>
Table 209-10  (Cont.) SETROWTAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>(IN)</td>
<td>Tag name.</td>
</tr>
</tbody>
</table>

209.3.9 SETUPDATECOLUMN

SETUPDATECOLUMN adds a column to the update column list.

In case of insert, the default is to insert values to all the columns in the table. In case of updates, the default is to only update the columns corresponding to the tags present in the **ROW** element of the XML document. When the update column list is specified, the columns making up this list alone will get updated or inserted into.

Note that if a user passes an XML file for INSERTXML to DBMS_XMLSTORE which contains extra elements (ones that do not match up to any columns in the table), Oracle will try to insert into those columns unless setUpdateColumn is used. The use of setUpdateColumn is optional only if the elements in the XML file match up to the columns in the table.

Syntax

```sql
PROCEDURE setUpdateColumn(
    ctxHdl IN ctxType,
    colName IN VARCHAR2);
```

Table 209-11  SETUPDATECOLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>colName</td>
<td>(IN)</td>
<td>Column to be added to the update column list.</td>
</tr>
</tbody>
</table>

209.3.10 UPDATEXML

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated.

The options are described in the following table.

Syntax

The following syntax passes the `xDoc` parameter as a `VARCHAR2`.

```sql
FUNCTION updateXML(
    ctxHdl IN ctxType,
    xDoc IN VARCHAR2)
RETURN NUMBER;
```
The following syntax passes the `xDoc` parameter as a `CLOB`.

```sql
FUNCTION updateXML(
    ctxHdl IN ctxType,
    xDoc IN CLOB)
RETURN NUMBER;
```

The following syntax passes the `xDoc` parameter as a `XMLType`.

```sql
FUNCTION updateXML(
    ctxHdl IN ctxType,
    xDoc IN XMLType)
RETURN NUMBER;
```

Parameters

**Table 209-12  UPDATEXML Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctxHdl</td>
<td>(IN)</td>
<td>Context handle.</td>
</tr>
<tr>
<td>xDoc</td>
<td>(IN)</td>
<td>String containing the XML document.</td>
</tr>
</tbody>
</table>
The DBMS_XPLAN package provides an easy way to display the output of the EXPLAIN PLAN command in several, predefined formats.

You can also use the DBMS_XPLAN package to display the plan of a statement stored in the Automatic Workload Repository (AWR) or stored in a SQL tuning set. It further provides a way to display the SQL execution plan and SQL execution runtime statistics for cached SQL cursors based on the information stored in the V$SQL_PLAN and V$SQL_PLAN_STATISTICS_ALL fixed views. Finally, it displays plans from a SQL plan baseline.

**See Also:**
- For more information on the EXPLAIN PLAN command, the AWR, and SQL tuning set, see Oracle Database SQL Tuning Guide.
- For more information on the V$SQL_PLAN fixed view, see Oracle Database Reference
- For more information on the V$SQL_PLAN_STATISTICS fixed view, see Oracle Database Reference

This chapter contains the following topics:
- Overview
- Security Model
- Examples
- Summary of DBMS_XPLAN Subprograms

### 210.1 DBMS_XPLAN Overview

The DBMS_XPLAN package supplies five table functions.

These functions are listed below:
- DISPLAY - to format and display the contents of a plan table.
- DISPLAY_AWR - to format and display the contents of the execution plan of a stored SQL statement in the AWR.
- DISPLAY_CURSOR - to format and display the contents of the execution plan of any loaded cursor.
- DISPLAY_SQL_PLAN_BASELINE - to display one or more execution plans for the SQL statement identified by SQL handle.
- DISPLAY_SQLSET - to format and display the contents of the execution plan of statements stored in a SQL tuning set.
210.2 DBMS_XPLAN Security Model

This package runs with the privileges of the calling user, not the package owner (SYS). The table function DISPLAY_CURSOR requires SELECT or READ privileges on the following fixed views: V$SQL_PLAN, V$SESSION and V$SQL_PLAN_STATISTICS_ALL. This function also requires SELECT/READ permissions on V$SQL.

DISPLAY_AWR Function requires the user to have SELECT or READ privileges on DBA_HIST_SQL_PLAN, DBA_HIST_SQLTEXT, and V$DATABASE.

DISPLAY_SQLSET Function requires the user to have the SELECT or READ privilege on ALL_SQLSET_STATEMENTS and ALL_SQLSET_PLANS.

DISPLAY_SQL_PLAN_BASELINE Function requires the user to have the SELECT or READ privilege on DBA_SQL_PLAN_BASELINES as well as the privileges to execute the SQL statement for which the user is trying to get the plan.

The preceding privileges are granted automatically as part of SELECT_CATALOG_ROLE.

210.3 Examples

These examples show sample uses of DBMS_XPLAN.

Displaying a Plan Table Using DBMS_XPLAN.DISPLAY

Execute an explain plan command on a SELECT statement:

```
EXPLAIN PLAN FOR
SELECT * FROM emp e, dept d
  WHERE e.deptno = d.deptno
  AND e.ename='benoit';
```

Display the plan using the DBMS_XPLAN.DISPLAY table function

```
SET LINESIZE 130
SET PAGESIZE 0
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

This query produces the following output:

```
Plan hash value: 3693697075

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>57</td>
<td>6 (34)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td>57</td>
<td>6 (34)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>1</td>
<td>37</td>
<td>3 (34)</td>
<td>00:00:01</td>
</tr>
<tr>
<td></td>
<td>TABLE ACCESS FULL</td>
<td>DEPT</td>
<td>4</td>
<td>80</td>
<td>3 (34)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

```
1 - access("E"."DEPTNO"="D"."DEPTNO")
2 - filter("E"."ENAME"='benoit')
```

15 rows selected.
Displaying a Cursor Execution Plan Using DBMS_XPLAN.DISPLAY_CURSOR

By default, the table function DISPLAY_CURSOR formats the execution plan for the last SQL statement executed by the session. For example:

```sql
SELECT ename FROM emp e, dept d
WHERE e.deptno = d.deptno
AND e.empno=7369;

ENAME
----------
SMITH
```

To display the execution plan of the last executed statement for that session:

```sql
SET PAGESIZE 0
SELECT * FROM DBMS_XPLAN.DISPLAY_CURSOR();
```

This query produces the following output:

```
Plan hash value: 3693697075, SQL hash value: 2096952573, child number: 0
------------------------------------------------------------------
SELECT ename FROM emp e, dept d WHERE e.deptno = d.deptno
AND e.empno=7369
---------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>*  1</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td>16</td>
<td>6 (34)%</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*  2</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>1</td>
<td>13</td>
<td>3 (34)%</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*  3</td>
<td>TABLE ACCESS FULL</td>
<td>DEPT</td>
<td>4</td>
<td>12</td>
<td>3 (34)%</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
---------------------------------------------------
1 - access("E"."DEPTNO"="D"."DEPTNO")
2 - filter("E"."EMPNO"=7369)
```

21 rows selected.

You can also use the table function DISPLAY_CURSOR to display the execution plan for any loaded cursor stored in the cursor cache. In that case, you must supply a reference to the child cursor to the table function. This includes the SQL ID of the statement and optionally the child number.

Run a query with a distinctive comment:

```sql
SELECT /* TOTO */ ename, dname
FROM dept d join emp e USING (deptno);
```

Get `sql_id` and `child_number` for the preceding statement:

```sql
SELECT sql_id, child_number
FROM v$sql
WHERE sql_text LIKE '%TOTO%';
```

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>CHILD_NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>gwp663cqh5gbf</td>
<td>0</td>
</tr>
</tbody>
</table>
Display the execution plan for the cursor:

```sql
SELECT * FROM DBMS_XPLAN.DISPLAY_CURSOR('gwp663cqh5qbf',0);
```

Plan hash value: 3693697075, SQL ID: gwp663cqh5qbf, child number: 0

```
--------------------------------------------------------
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>7 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT GROUP BY</td>
<td></td>
<td>4</td>
<td>64</td>
<td>7 (43)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>HASH JOIN</td>
<td></td>
<td>14</td>
<td>224</td>
<td>6 (34)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>DEPT</td>
<td>4</td>
<td>44</td>
<td>3 (34)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>EMP</td>
<td>14</td>
<td>70</td>
<td>3 (34)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```

Predicate Information (identified by operation id):

```
2 - access("E"."DEPTNO"="D"."DEPTNO")
```

Instead of issuing two queries, one to the get the sql_id and child_number pair and one to display the plan, you can combine these in a single query:

```
SELECT t.*
FROM v$sq1 s, DBMS_XPLAN.DISPLAY_CURSOR(s.sql_id, s.child_number) t WHERE sql_text LIKE '%TOTO%';
```

Displaying a Plan Table with Parallel Information

By default, only relevant information is reported by the display and `display_cursor` table functions. In Displaying a Plan Table Using DBMS_XPLAN.DISPLAY, the query does not execute in parallel. Hence, information related to the parallelization of the plan is not reported. As shown in the following example, parallel information is reported only if the query executes in parallel.

```
ALTER TABLE emp PARALLEL;
EXPLAIN PLAN for
SELECT * FROM emp e, dept d
WHERE e.deptno = d.deptno
AND e.ename = 'hermann'
ORDER BY e.empno;
```

Display the plan using the `DBMS_XPLAN.DISPLAY` table function

```
SET LINESIZE 130
SET PAGE_SIZE 0
SELECT * FROM DBMS_XPLAN.DISPLAY();
Plan hash value: 3693697345
```

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>TQ</th>
<th>INOUT</th>
<th>PQ Distrib</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>117</td>
<td>6 (50)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PX COORDINATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PX SEND QC (ORDER)</td>
<td>:TQ10003</td>
<td>1</td>
<td>117</td>
<td>6 (50)</td>
<td>00:00:01</td>
<td>Q1.03</td>
<td>P-&gt;S</td>
<td>QC (ORDER)</td>
</tr>
<tr>
<td>3</td>
<td>SORT ORDER BY</td>
<td></td>
<td>1</td>
<td>117</td>
<td>6 (50)</td>
<td>00:00:01</td>
<td>Q1.03</td>
<td>PCWP</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PX RECEIVE</td>
<td></td>
<td>1</td>
<td>117</td>
<td>5 (40)</td>
<td>00:00:01</td>
<td>Q1.03</td>
<td>PCWP</td>
<td></td>
</tr>
</tbody>
</table>
```
Predicate Information (identified by operation id):
---------------------------------------------------
6 - access("E"."DEPTNO"="D"."DEPTNO")
10 - filter("E"."ENAME"='hermann')
---------------------------------------------------

When the query is parallel, information related to parallelism is reported: table queue number (TQ column), table queue type (INOUT) and table queue distribution method (PQ Distrib).

By default, if several plans in the plan table match the statement_id parameter passed to the display table function (default value is NULL), only the plan corresponding to the last EXPLAIN PLAN command is displayed. Hence, there is no need to purge the plan table after each EXPLAIN PLAN. However, you should purge the plan table regularly to ensure good performance in the execution of the DISPLAY table function. If no plan table is created, Oracle uses a global temporary table to store any plan information for individual users and preserves its content throughout the lifespan of a session. Note that you cannot truncate the content of a global temporary table.

For ease of use, you can define a view on top of the display table function and then use that view to display the output of the EXPLAIN PLAN command:

Using a View to Display Last Explain Plan

```sql
# define plan view
CREATE VIEW PLAN AS SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

# display the output of the last explain plan command
SELECT * FROM PLAN;
```

### 210.4 Summary of DBMS_XPLAN Subprograms

This table lists the DBMS_XPLAN subprograms and briefly describes them.

#### Table 210-1 DBMS_XPLAN Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIFF_PLAN Function</td>
<td>Compares plans</td>
</tr>
<tr>
<td>DISPLAY Function</td>
<td>Displays the contents of the plan table</td>
</tr>
<tr>
<td>DISPLAY_AWR Function</td>
<td>Displays the contents of an execution plan stored in the AWR</td>
</tr>
<tr>
<td>DISPLAY_CURSOR Function</td>
<td>Displays the execution plan of any cursor in the cursor cache</td>
</tr>
<tr>
<td>DISPLAY_PLAN Function</td>
<td>Displays the contents of the plan table in a variety of formats with CLOB output type</td>
</tr>
<tr>
<td>DISPLAY_SQL_PLAN_BASE_LINE Function</td>
<td>Displays one or more execution plans for the specified SQL handle of a SQL plan baseline</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DISPLAY_SQLSET Function</td>
<td>Displays the execution plan of a given statement stored in a SQL tuning set</td>
</tr>
</tbody>
</table>

### 210.4.1 DIFF_PLAN Function

This function compares plans.

**Syntax**

```sql
DBMS_XPLAN.DIFF_PLAN(
    plan1    IN   SPC_SRC,
    plan2    IN   SPC_SRC)
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_1</td>
<td>First plan</td>
</tr>
<tr>
<td>plan_2</td>
<td>Second plan</td>
</tr>
</tbody>
</table>

### 210.4.2 DISPLAY Function

This table function displays the contents of the plan table.

In addition, you can use this table function to display any plan (with or without statistics) stored in a table as long as the columns of this table are named the same as columns of the plan table (or `V$SQL_PLAN_STATISTICS_ALL` if statistics are included). You can apply a predicate on the specified table to select rows of the plan to display.

**Syntax**

```sql
DBMS_XPLAN.DISPLAY(
    table_name    IN  VARCHAR2  DEFAULT 'PLAN_TABLE',
    statement_id  IN  VARCHAR2  DEFAULT NULL,
    format        IN  VARCHAR2  DEFAULT 'TYPICAL',
    filter_preds  IN  VARCHAR2  DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Specifies the table name where the plan is stored. This parameter defaults to <code>PLAN_TABLE</code>, which is the default plan table for the <code>EXPLAIN PLAN</code> command. If <code>NULL</code> is specified it also defaults to <code>PLAN_TABLE</code>.</td>
</tr>
</tbody>
</table>
Table 210-3  (Cont.) DISPLAY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement_id</td>
<td>Specifies the statement_id of the plan to be displayed. This parameter defaults to NULL, which is the default when the EXPLAIN PLAN command is executed without a set statement_id clause. If no statement_id is specified, the function shows you the plan of the most recent explained statement.</td>
</tr>
</tbody>
</table>
Table 210-3 (Cont.) DISPLAY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Controls the level of details for the plan. It accepts four values:</td>
</tr>
<tr>
<td></td>
<td>- BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option.</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).</td>
</tr>
<tr>
<td></td>
<td>- SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.</td>
</tr>
<tr>
<td></td>
<td>- ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).</td>
</tr>
</tbody>
</table>

For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE). Format keywords must be separated by either a comma or a space:

- ROWS - if relevant, shows the number of rows estimated by the optimizer
- BYTES - if relevant, shows the number of bytes estimated by the optimizer
- COST - if relevant, shows optimizer cost information
- PARTITION - if relevant, shows partition pruning information
- PARALLEL - if relevant, shows PX information (distribution method and table queue information)
- PREDICATE - if relevant, shows the predicate section
- PROJECTION - if relevant, shows the projection section
- ALIAS - if relevant, shows the “Query Block Name / Object Alias” section
- REMOTE - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)
- NOTE - if relevant, shows the note section of the explain plan

Format keywords can be prefixed by the sign ‘~’ to exclude the specified information. For example, ‘~PROJECTION’ excludes projection information.

If the target plan table (see table_name parameter) also stores plan statistics columns (for example, it is a table used to capture the content of the fixed view V$SQL_PLAN_STATISTICS_ALL), additional format keywords can be used to specify which class of statistics to display when using the DISPLAY Function. These additional format keywords are IOSTATS, MEMSTATS, ALLSTATS and LAST (see the DISPLAY_CURSOR Function or the DISPLAY_SQLSET Function for a full description of these four keywords).
Table 210-3  (Cont.) DISPLAY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter_preds</td>
<td>SQL filter predicate(s) to restrict the set of rows selected from the table where the plan is stored. When value is NULL (the default), the plan displayed corresponds to the last executed explain plan. For example: filter_preds=&gt;'plan_id = 10' Can reference any column of the table where the plan is stored and can contain any SQL construct (for example, sub-query, function calls (see WARNING under Usage Notes)</td>
</tr>
</tbody>
</table>

Usage Notes

Here are some ways you might use variations on the format parameter:

- Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
- Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
- Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.

⚠️ WARNING:

Application developers should expose the filter_preds parameter to end-users only after careful consideration because this could expose the application to SQL injection. Indeed, filter_preds can potentially reference any table or execute any server function for which the database user invoking the table function has privileges.

Examples

To display the result of the last EXPLAIN PLAN command stored in the plan table:

```sql
SELECT * FROM table (DBMS_XPLAN.DISPLAY);
```

To display from other than the default plan table, "my_plan_table":

```sql
SELECT * FROM table (DBMS_XPLAN.DISPLAY('my_plan_table'));
```

To display the minimum plan information:

```sql
SELECT * FROM table (DBMS_XPLAN.DISPLAY('plan_table', null, 'basic'));
```

To display the plan for a statement identified by 'foo', such as statement_id='foo':

```sql
SELECT * FROM table (DBMS_XPLAN.DISPLAY('plan_table', 'foo'));
```
210.4.3 DISPLAY_AWR Function

This table function displays the contents of an execution plan stored in the AWR.

Syntax

```sql
DBMS_XPLAN.DISPLAY_AWR(
    sql_id            IN      VARCHAR2,
    plan_hash_value   IN      NUMBER DEFAULT NULL,
    db_id             IN      NUMBER DEFAULT NULL,
    format            IN      VARCHAR2 DEFAULT TYPICAL);
```

Parameters

Table 210-4  DISPLAY_AWR Table Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>Specifies the SQL_ID of the SQL statement. You can retrieve the appropriate value for the SQL statement of interest by querying the column SQL_ID in DBA_HIST_SQLTEXT.</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Specifies the PLAN_HASH_VALUE of a SQL statement. This parameter is optional. If omitted, the table function returns all stored execution plans for a given SQL_ID.</td>
</tr>
<tr>
<td>db_id</td>
<td>Specifies the database_id for which the plan of the SQL statement, identified by SQL_ID should be displayed. If not supplied, the database_id of the local database is used, as shown in V$DATABASE.</td>
</tr>
</tbody>
</table>
Table 210-4  (Cont.) DISPLAY_AWR Table Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Controls the level of details for the plan. It accepts four values:</td>
</tr>
<tr>
<td></td>
<td>• BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option.</td>
</tr>
<tr>
<td></td>
<td>• TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).</td>
</tr>
<tr>
<td></td>
<td>• SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.</td>
</tr>
<tr>
<td></td>
<td>• ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).</td>
</tr>
</tbody>
</table>

For finer control on the display output, the following keywords can be added to the above four standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE). Format keywords must be separated by either a comma or a space:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWS</td>
<td>if relevant, shows the number of rows estimated by the optimizer</td>
</tr>
<tr>
<td>BYTES</td>
<td>if relevant, shows the number of bytes estimated by the optimizer</td>
</tr>
<tr>
<td>COST</td>
<td>if relevant, shows optimizer cost information</td>
</tr>
<tr>
<td>PARTITION</td>
<td>if relevant, shows partition pruning information</td>
</tr>
<tr>
<td>PARALLEL</td>
<td>if relevant, shows PX information (distribution method and table queue information)</td>
</tr>
<tr>
<td>PREDICATE</td>
<td>if relevant, shows the predicate section</td>
</tr>
<tr>
<td>PROJECTION</td>
<td>if relevant, shows the projection section</td>
</tr>
<tr>
<td>ALIAS</td>
<td>if relevant, shows the &quot;Query Block Name / Object Alias&quot; section</td>
</tr>
<tr>
<td>REMOTE</td>
<td>if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)</td>
</tr>
<tr>
<td>NOTE</td>
<td>if relevant, shows the note section of the explain plan</td>
</tr>
</tbody>
</table>

Format keywords can be prefixed by the sign ‘-’ to exclude the specified information. For example, ‘-PROJECTION’ excludes projection information.

Usage Notes

- To use the DISPLAY_AWR functionality, the calling user must have SELECT or READ privilege on DBA_HIST_SQL_PLAN, DBA_HIST_SQLTEXT, and V$DATABASE, otherwise it shows an appropriate error message.

- Here are some ways you might use variations on the format parameter:
  - Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
- Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
- Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.

Examples

To display the different execution plans associated with the SQL ID 'atfwcg8anrykp':

```sql
SELECT * FROM table(DBMS_XPLAN.DISPLAY_AWR('atfwcg8anrykp'));
```

To display all execution plans of all stored SQL statements containing the string 'TOTO':

```sql
SELECT tf.* FROM DBA_HIST_SQLTEXT ht, table
  (DBMS_XPLAN.DISPLAY_AWR(ht.sql_id,null, null, 'ALL' )) tf
WHERE ht.sql_text like '%TOTO%';
```

### 210.4.4 DISPLAY_CURSOR Function

This table function displays the explain plan of any cursor loaded in the cursor cache. In addition to the explain plan, various plan statistics (such as I/O, memory and timing) can be reported (based on the V$SQL_PLAN_STATISTICS_ALL VIEWS).

**Syntax**

```sql
DBMS_XPLAN.DISPLAY_CURSOR(
    sql_id            IN  VARCHAR2  DEFAULT  NULL,
    cursor_child_no   IN  NUMBER    DEFAULT  0,
    format            IN  VARCHAR2  DEFAULT  'TYPICAL');
```

**Parameters**

**Table 210-5 DISPLAY_CURSOR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_id</td>
<td>Specifies the SQL_ID of the SQL statement in the cursor cache. You can retrieve the appropriate value by querying the column SQL_ID in V$SQL or V$SQLAREA. Alternatively, you could choose the column PREV_SQL_ID for a specific session out of V$SESSION. This parameter defaults to NULL in which case the plan of the last cursor executed by the session is displayed.</td>
</tr>
<tr>
<td>cursor_child_no</td>
<td>Child number of the cursor to display. If not supplied, the execution plan of the child_number=0 cursor matching the supplied sql_id parameter are displayed. The child_number can be specified only if sql_id is specified.</td>
</tr>
</tbody>
</table>
### Table 210-5  (Cont.) DISPLAY_CURSOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Controls the level of details for the plan. It accepts five values:</td>
</tr>
<tr>
<td></td>
<td>- <strong>BASIC</strong>: Displays the minimum information in the plan—the operation ID, the operation name and its option.</td>
</tr>
<tr>
<td></td>
<td>- <strong>TYPICAL</strong>: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).</td>
</tr>
<tr>
<td></td>
<td>- <strong>SERIAL</strong>: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.</td>
</tr>
<tr>
<td></td>
<td>- <strong>ALL</strong>: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).</td>
</tr>
<tr>
<td></td>
<td>- <strong>ADAPTIVE</strong>:</td>
</tr>
<tr>
<td></td>
<td>- Displays the final plan, or the current plan if the execution has not completed. This section includes notes about runtime optimizations that affect the plan, such as switching from a Nested Loops join to a Hash join.</td>
</tr>
<tr>
<td></td>
<td>- Plan lineage. This section shows the plans that were run previously due to automatic reoptimization. It also shows the default plan, if the plan changed due to dynamic plans.</td>
</tr>
<tr>
<td></td>
<td>- Recommended plan. In reporting mode, the plan is chosen based on execution statistics displayed. Note that displaying the recommended plan for automatic reoptimization requires re-compiling the query with the optimizer adjustments collected in the child cursor. Displaying the recommended plan for a dynamic plan does not require this.</td>
</tr>
<tr>
<td></td>
<td>- Dynamic plans. This summarizes the portions of the plan that differ from the default plan chosen by the optimizer.</td>
</tr>
</tbody>
</table>

For finer control on the display output, you can add the following keywords to the preceding format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE).

Format keywords must be separated by either a comma or a space:

- **ROWS** - if relevant, shows the number of rows estimated by the optimizer
- **BYTES** - if relevant, shows the number of bytes estimated by the optimizer
- **COST** - if relevant, shows optimizer cost information
- **PARTITION** - if relevant, shows partition pruning information
- **PARALLEL** - if relevant, shows PX information (distribution method and table queue information)
- **PREDICATE** - if relevant, shows the predicate section
- **PROJECTION** - if relevant, shows the projection section
- **ALIAS** - if relevant, shows the "Query Block Name / Object Alias" section
- **REMOTE** - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)
- **NOTE** - if relevant, shows the note section of the explain plan
Table 210-5  (Cont.) DISPLAY_CURSOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOSTATS</td>
<td>Assuming that basic plan statistics are collected when SQL statements are executed (either by using the gather_plan_statistics hint or by setting the parameter statistics_level to ALL), this format shows IO statistics for ALL (or only for the LAST as shown below) executions of the cursor.</td>
</tr>
<tr>
<td>MEMSTATS</td>
<td>Assuming that PGA memory management is enabled (that is, pga_aggregate_target parameter is set to a non 0 value), this format allows to display memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators.</td>
</tr>
<tr>
<td>ALLSTATS</td>
<td>A shortcut for 'IOSTATS MEMSTATS'</td>
</tr>
<tr>
<td>LAST</td>
<td>By default, plan statistics are shown for all executions of the cursor. The keyword LAST can be specified to see only the statistics for the last execution.</td>
</tr>
</tbody>
</table>

The following formats are deprecated but supported for backward compatibility:
- RUNSTATS_TOT - Same as IOSTATS, that is, displays IO statistics for all executions of the specified cursor.
- RUNSTATS_LAST - Same as IOSTATS LAST, that is, displays the runtime statistics for the last execution of the cursor.

You can prefix format keywords with the sign '-\-' to exclude the specified information. For example, '-\-PROJECTION' excludes projection information.

Usage Notes

- To use the DISPLAY_CURSOR functionality, the calling user must have SELECT or READ privilege on the fixed views V$SQL_PLAN_STATISTICS_ALL, V$SQL, and V$SQL_PLAN, otherwise it shows an appropriate error message.
- Here are some ways you might use variations on the format parameter:
  - Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
  - Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
  - Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
  - Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.
Examples

To display the execution plan of the last SQL statement executed by the current session:

```sql
SELECT * FROM table (
    DBMS_XPLAN.DISPLAY_CURSOR);
```

To display the execution plan of all children associated with the SQL ID 'atfwcg8anrykp':

```sql
SELECT * FROM table (
    DBMS_XPLAN.DISPLAY_CURSOR('atfwcg8anrykp'));
```

To display runtime statistics for the cursor included in the preceding statement:

```sql
SELECT * FROM table (
    DBMS_XPLAN.DISPLAY_CURSOR('atfwcg8anrykp', NULL, 'ALLSTATS LAST'));
```

210.4.5 DISPLAY_PLAN Function

This table function displays the contents of the plan table in a variety of formats with CLOB output type.

Syntax

```sql
DBMS_XPLAN.DISPLAY_PLAN (  
    table_name       IN    VARCHAR2   DEFAULT 'PLAN_TABLE',  
    statement_id     IN    VARCHAR2   DEFAULT NULL,  
    format           IN    VARCHAR2   DEFAULT 'TYPICAL',  
    filter_preds     IN    VARCHAR2   DEFAULT NULL,  
    type             IN    VARCHAR2   DEFAULT 'TEXT')  
RETURN CLOB;
```

Parameters

Table 210-6  DISPLAY_PLAN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Specifies the table name where the plan is stored. This parameter defaults to PLAN_TABLE, which is the default plan table for the EXPLAIN PLAN command. If NULL is specified it also defaults to PLAN_TABLE.</td>
</tr>
<tr>
<td>statement_id</td>
<td>Specifies the statement id of the plan to be displayed. This parameter defaults to NULL, which is the default when the EXPLAIN PLAN command is executed without a set statement_id clause. If no statement_id is specified, the function shows you the plan of the most recent explained statement.</td>
</tr>
</tbody>
</table>
Table 210-6  (Cont.) DISPLAY_PLAN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| filter_preds| SQL filter predicate(s) to restrict the set of rows selected from the table where the plan is stored. When value is NULL (the default), the plan displayed corresponds to the last executed explain plan. For example: filter_preds=>'plan_id = 10'

Can reference any column of the table where the plan is stored and can contain any SQL construct (for example, sub-query, function calls (see WARNING under Usage Notes)
### Table 210-6  (Cont.) DISPLAY_PLAN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Controls the level of details for the plan. It accepts five values:</td>
</tr>
<tr>
<td></td>
<td>- BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option.</td>
</tr>
<tr>
<td></td>
<td>- TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).</td>
</tr>
<tr>
<td></td>
<td>- SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.</td>
</tr>
<tr>
<td></td>
<td>- ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).</td>
</tr>
<tr>
<td></td>
<td>- ADAPTIVE: Displays the default plan, and for each dynamic subplan (if stipulated):</td>
</tr>
<tr>
<td></td>
<td>- A list of the rowsources from the original which may be replaced, and the rowsources to replace them</td>
</tr>
<tr>
<td></td>
<td>- If outline display is specified in the format argument, the hints for each option in the dynamic subplan are displayed</td>
</tr>
<tr>
<td></td>
<td>For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE). Format keywords must be separated by either a comma or a space:</td>
</tr>
<tr>
<td></td>
<td>- ROWS - if relevant, shows the number of rows estimated by the optimizer</td>
</tr>
<tr>
<td></td>
<td>- BYTES - if relevant, shows the number of bytes estimated by the optimizer</td>
</tr>
<tr>
<td></td>
<td>- COST - if relevant, shows optimizer cost information</td>
</tr>
<tr>
<td></td>
<td>- PARTITION - if relevant, shows partition pruning information</td>
</tr>
<tr>
<td></td>
<td>- PARALLEL - if relevant, shows PX information (distribution method and table queue information)</td>
</tr>
<tr>
<td></td>
<td>- PREDICATE - if relevant, shows the predicate section</td>
</tr>
<tr>
<td></td>
<td>- PROJECTION - if relevant, shows the projection section</td>
</tr>
<tr>
<td></td>
<td>- ALIAS - if relevant, shows the &quot;Query Block Name / Object Alias&quot; section</td>
</tr>
<tr>
<td></td>
<td>- REMOTE - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)</td>
</tr>
<tr>
<td></td>
<td>- NOTE - if relevant, shows the note section of the explain plan</td>
</tr>
<tr>
<td></td>
<td>Format keywords can be prefixed by the sign <code>'-'</code> to exclude the specified information. For example, <code>'--PROJECT'</code> excludes projection information.</td>
</tr>
<tr>
<td></td>
<td>If the target plan table (see table_name parameter) also stores plan statistics columns (for example, it is a table used to capture the content of the fixed view <code>V$SQL_PLAN_STATISTICS_ALL</code>), additional format keywords can be used to specify which class of statistics to display when using the DISPLAY Function. These additional format keywords are IOSTATS, MEMSTATS, ALLSTATS and LAST (see the DISPLAY_CURSOR Function or the DISPLAY_SQLSET Function for a full description of these four keywords).</td>
</tr>
</tbody>
</table>
Table 210-6 (Cont.) DISPLAY_PLAN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Output type, one of: 'TEXT', 'ACTIVE', 'HTML', or 'XML' (see Usage Notes regarding type ACTIVE).</td>
</tr>
</tbody>
</table>

Return Values

Returns the requested report as CLOB

Usage Notes

Active reports have a rich, interactive user interface akin to that found in Enterprise Manager while not requiring any EM installation. The report file built is in HTML format, so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.

⚠️ WARNING:

Application developers should expose the filter_preds parameter to end-users only after careful consideration because this could expose the application to SQL injection. Indeed, filter_preds can potentially reference any table or execute any server function for which the database user invoking the table function has privileges.

210.4.6 DISPLAY_SQL_PLAN_BASELINE Function

This table function displays one or more execution plans for the specified SQL handle of a SQL plan baseline.

Syntax

```
DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE (
    sql_handle      IN VARCHAR2 := NULL,
    plan_name       IN VARCHAR2 := NULL,
    format          IN VARCHAR2 := 'TYPICAL')
RETURN dbms_xplan_type_table;
```

Parameters

Table 210-7 DISPLAY_SQL_PLAN_BASELINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_handle</td>
<td>SQL statement handle. It identifies a SQL statement whose plan(s) are to be displayed.</td>
</tr>
<tr>
<td>plan_name</td>
<td>Plan name. It identifies a specific plan. Default NULL means all plans associated with identified SQL statement are explained and displayed.</td>
</tr>
</tbody>
</table>
### Table 210-7 (Cont.) DISPLAY_SQL_PLAN_BASELINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Format string determines what information stored in the plan displayed. One of three format values ('BASIC', 'TYPICAL', 'ALL') can be used, each representing a common use case.</td>
</tr>
</tbody>
</table>

### Return Values

A PL/SQL type table

### Usage Notes

This procedure uses plan information stored in the plan baseline to explain and display the plans. It is possible that the plan_id stored in the SQL management base may not match with the plan_id of the generated plan. A mismatch between stored plan_id and generated plan_id means that it is a non-reproducible plan. Such a plan is deemed invalid and is bypassed by the optimizer during SQL compilation.

### Examples

Display all plans of a SQL statement identified by the SQL handle 'SYS_SQL_b1d49f6074ab95af' using TYPICAL format

```sql
SET LINESIZE 150
SET PAGESIZE 2000
SELECT t.*
FROM TABLE(DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE('SYS_SQL_b1d49f6074ab95af')) t;
```

Display all plans of one or more SQL statements containing the string 'HR2' using BASIC format

```sql
SET LINESIZE 150
SET PAGESIZE 2000
SELECT t.*
FROM (SELECT DISTINCT sql_handle FROM dba_sql_plan_baselines WHERE sql_text like '%HR2%') pb,
     TABLE(DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(pb.sql_handle, NULL, 'BASIC')) t;
```

### 210.4.7 DISPLAY_SQLSET Function

This table function displays the execution plan of a given statement stored in a SQL tuning set.

#### Syntax

```sql
DBMS_XPLAN.DISPLAY_SQLSET(
    sqlset_name IN VARCHAR2,
    sql_id IN VARCHAR2,
    plan_hash_value IN NUMBER := NULL,
    format IN VARCHAR2 := 'TYPICAL',
    sqlset_owner IN VARCHAR2 := NULL)
RETURN DBMS_XPLAN_TYPE_TABLE PIPELINED;
```
### Table 210-8  DISPLAY_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlset_name</td>
<td>Name of the SQL Tuning Set</td>
</tr>
<tr>
<td>sql_id</td>
<td>Specifies the sql_id value for a SQL statement having its plan stored in the SQL tuning set. You can find all stored SQL statements by querying table function DBMS_SQLTUNE.SELECT_SQLSET</td>
</tr>
<tr>
<td>plan_hash_value</td>
<td>Optional parameter. Identifies a specific stored execution plan for a SQL statement. If suppressed, all stored execution plans are shown.</td>
</tr>
</tbody>
</table>
### Table 210-8  (Cont.) DISPLAY_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| format    | Controls the level of details for the plan. It accepts four values:  
  - **BASIC**: Displays the minimum information in the plan—the operation ID, the operation name and its option.  
  - **TYPICAL**: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).  
  - **SERIAL**: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.  
  - **ALL**: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).  
  For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE). Format keywords must be separated by either a comma or a space:  
  - **ROWS**: if relevant, shows the number of rows estimated by the optimizer  
  - **BYTES**: if relevant, shows the number of bytes estimated by the optimizer  
  - **COST**: if relevant, shows optimizer cost information  
  - **PARTITION**: if relevant, shows partition pruning information  
  - **PARALLEL**: if relevant, shows PX information (distribution method and table queue information)  
  - **PREDICATE**: if relevant, shows the predicate section  
  - **PROJECTION**: if relevant, shows the projection section  
  - **ALIAS**: if relevant, shows the "Query Block Name/Object Alias" section  
  - **REMOTE**: if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)  
  - **NOTE**: if relevant, shows the note section of the explain plan  
  - **IOSTATS**: assuming that basic plan statistics are collected when SQL statements are executed (either by using the gather_plan_statistics hint or by setting the parameter statistics_level to ALL), this format shows IO statistics for ALL (or only for the LAST as shown below) executions of the cursor.  
  - **MEMSTATS**: Assuming that PGA memory management is enabled (that is, pga_aggregate_target parameter is set to a non 0 value), this format allows to display...
Table 210-8  (Cont.) DISPLAY_SQLSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators.</td>
<td></td>
</tr>
<tr>
<td>ALLSTATS - A shortcut for 'IOSTATS MEMSTATS'</td>
<td></td>
</tr>
<tr>
<td>LAST - By default, plan statistics are shown for all executions of the cursor. The keyword LAST can be specified to see only the statistics for the last execution.</td>
<td></td>
</tr>
</tbody>
</table>

The following two formats are deprecated but supported for backward compatibility:

| RUNSTATS_TOT - Same as IOSTATS, that is, displays IO statistics for all executions of the specified cursor. |
| RUNSTATS_LAST - Same as IOSTATS LAST, that is, displays the runtime statistics for the last execution of the cursor |

Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information.

Usage Notes

Here are some ways you might use variations on the format parameter:

- Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
- Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
- Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.

Examples

To display the execution plan for the SQL statement associated with SQL ID 'gwp663cqh5qbf' and PLAN HASH 3693697075 in the SQL Tuning Set called 'OLTP_optimization_0405':

```sql
SELECT * FROM table (DBMS_XPLAN.DISPLAY_SQLSET('OLTP_optimization_0405','gwp663cqh5qbf', 3693697075));
```

To display all execution plans of the SQL ID 'atfwcg8anrykp' stored in the SQL tuning set:
To display runtime statistics for the SQL statement included in the preceding statement:

```
SELECT * FROM table {
    DBMS_XPLAN.DISPLAY_SQLSET(
        'OLTP_optimization_0405', 'gwp663cqh5qbf');
}
```
The DBMS_XSLPROCESSOR package provides an interface to manage the contents and structure of XML documents.

This chapter contains the following topics:

- DBMS_XSLPROCESSOR Overview
- DBMS_XSLPROCESSOR Deprecated Subprograms
- DBMS_XSLPROCESSOR Security Model
- Summary of DBMS_XSLPROCESSOR Subprograms

See Also:
- Oracle XML DB Developer’s Guide

211.1 DBMS_XSLPROCESSOR Overview

The DBMS_XSLPROCESSOR package provides an interface to manage the contents and structure of XML documents.

Standards

This PL/SQL implementation of the XSL processor follows the W3C XSLT working draft rev WD-xslt-19990813 and includes the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformation it must effect.

Concepts

The Extensible Stylesheet Language Transformation (XSLT) describes rules for transforming a source tree into a result tree. A transformation expressed in XSLT is called a stylesheet. The transformation specified is achieved by associating patterns with templates defined in the stylesheet. A template is instantiated to create part of the result tree.

Implementation

The following is the default behavior for this PL/SQL XSL Processor:

- A result tree which can be accessed by DOM programmatic interface
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails
211.2 DBMS_XSLPROCESSOR Deprecated Subprograms

These subprograms are deprecated from the DBMS_XSLPROCESSOR package in Oracle Database 12c release 12.2.

**Note:**
Oracle recommends that do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

Starting in this release, these subprograms are available in the DBMS_LOB package.

- CLOB2FILE Procedure
- READ2CLOB Function

**See Also:**
- Summary of DBMS_LOB Subprograms

211.3 DBMS_XSLPROCESSOR Security Model

Owned by XDB, the DBMS_XSLPROCESSOR package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

211.4 Summary of DBMS_XSLPROCESSOR Subprograms

This table lists the DBMS_XSLPROCESSOR subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOB2FILE Procedure</td>
<td>Writes content of a CLOB into a file</td>
</tr>
<tr>
<td></td>
<td>This procedure has been deprecated starting in Oracle Database 12c release 12.2.</td>
</tr>
<tr>
<td>FREEPROCESSOR Procedure</td>
<td>Frees a processor object</td>
</tr>
<tr>
<td>FREESTYLESHEET Procedure</td>
<td>Frees a stylesheet object</td>
</tr>
<tr>
<td>NEWPROCESSOR Function</td>
<td>Returns a new processor instance</td>
</tr>
<tr>
<td>NEWSTYLESHEET Functions</td>
<td>Creates a new stylesheet from input and reference URLs</td>
</tr>
<tr>
<td>PROCESSXSL Functions and Procedures</td>
<td>Transforms an input XML document</td>
</tr>
</tbody>
</table>
Table 211-1 (Cont.) DBMS_XSLPROCESSOR Package Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ2CLOB Function</td>
<td>Reads content of the file into a CLOB</td>
</tr>
<tr>
<td></td>
<td>This procedure has been deprecated starting in Oracle Database 12c release 12.2.</td>
</tr>
<tr>
<td>REMOVEPARAM Procedure</td>
<td>Removes a top-level stylesheet parameter</td>
</tr>
<tr>
<td>RESETPARAMS Procedure</td>
<td>Resets the top-level stylesheet parameters</td>
</tr>
<tr>
<td>SELECTNODES Function</td>
<td>Selects nodes from a DOM tree that match a pattern</td>
</tr>
<tr>
<td>SELECTSINGLENODE Function</td>
<td>Selects the first node from the tree that matches a pattern</td>
</tr>
<tr>
<td>SETERRORLOG Procedure</td>
<td>Sets errors to be sent to the specified file</td>
</tr>
<tr>
<td>SETPARAM Procedure</td>
<td>Sets a top-level parameter in the stylesheet</td>
</tr>
<tr>
<td>SHOWWARNINGS Procedure</td>
<td>Turns warnings on or off</td>
</tr>
<tr>
<td>TRANSFORMNODE Function</td>
<td>Transforms a node in a DOM tree using a stylesheet</td>
</tr>
<tr>
<td>VALUEOF Function and Procedure</td>
<td>Gets the value of the first node that matches a pattern</td>
</tr>
</tbody>
</table>

211.4.1 CLOB2FILE Procedure

This procedure writes content of a CLOB into a file. This procedure has moved to the DBMS_LOB package, starting in Oracle Database 12c release 12.2.

Syntax

```sql
DBMS_XSLPROCESSOR.CLOB2FILE(
    cl          IN  CLOB;
    flocation   IN  VARCHAR2,
    fname       IN  VARCHAR2,
    csid        IN  NUMBER:=0);
```

Parameters

Table 211-2 CLOB2FILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOB</td>
<td>File directory</td>
</tr>
<tr>
<td>flocation</td>
<td>File directory</td>
</tr>
<tr>
<td>fname</td>
<td>File name</td>
</tr>
<tr>
<td>csid</td>
<td>Character set id of the file</td>
</tr>
<tr>
<td></td>
<td>• Must be a valid Oracle id; otherwise returns an error</td>
</tr>
<tr>
<td></td>
<td>• If 0, content of the output file will be in the database character set</td>
</tr>
</tbody>
</table>
211.4.2 FREEPROCESSOR Procedure

This procedure frees a Processor object.

Syntax

DBMS_XSLPROCESSOR.FREEPROCESSOR(
    p IN Processor);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Processor</td>
</tr>
</tbody>
</table>

211.4.3 FREESTYLESHEET Procedure

This procedure frees a Stylesheet object.

Syntax

DBMS_XSLPROCESSOR.FREESTYLESHEET(
    ss IN Stylesheet);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss</td>
<td>Stylesheet</td>
</tr>
</tbody>
</table>

211.4.4 NEWPROCESSOR Function

This function returns a new Processor instance.

The function must be called before the default behavior of Processor can be changed and if other processor methods need to be used.

Syntax

DBMS_XSLPROCESSOR.NEWPROCESSOR
    RETURN Processor;

211.4.5 NEWSTYLESHEET Functions

This function creates and returns a new Stylesheet instance.

The options are described in the following table.
Syntax

Creates and returns a new stylesheet instance using the given DOMDocument and reference URLs:

```sql
DBMS_XSLPROCESSOR.NEWSTYLESHEET(
    xmldoc IN DOMDOCUMENT,
    ref IN VARCHAR2)
RETURN Stylesheet;
```

Creates and returns a new Stylesheet instance using the given input and reference URLs:

```sql
DBMS_XSLPROCESSOR.NEWSTYLESHEET(
    inp IN VARCHAR2,
    ref IN VARCHAR2)
RETURN Stylesheet;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmldoc</td>
<td>DOMDocument to use for construction</td>
</tr>
<tr>
<td>inp</td>
<td>Input URL to use for construction</td>
</tr>
<tr>
<td>ref</td>
<td>Reference URL</td>
</tr>
</tbody>
</table>

211.4.6 PROCESSXSL Functions and Procedures

This function transforms input XMLDocument.

Any changes to the default processor behavior should be effected before calling this procedure. An application error is raised if processing fails.

Syntax

Transforms input XMLDocument using given DOMDocument and stylesheet, and returns the resultant document fragment:

```sql
DBMS_XSLPROCESSOR.PROCESSXSL(
    p IN Processor,
    ss IN Stylesheet,
    xmldoc IN DOMDOCUMENT),
RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input XMLDocument using given document as URL and the Stylesheet, and returns the resultant document fragment:

```sql
DBMS_XSLPROCESSOR.PROCESSXSL(
    p IN Processor,
    ss IN Stylesheet,
    url IN VARCHAR2,
RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input XMLDocument using given document as CLOB and the Stylesheet, and returns the resultant document fragment:
Transforms input XMLDocument using given DOMDOCUMENT and the stylesheet, and writes the output to the specified file:

```sql
DBMS_XSLPROCESSOR.DBMS_XSLPROCESSOR(
    p IN Processor,
    ss IN Stylesheet,
    xmldoc IN DOMDOCUMENT,
    dir IN VARCHAR2,
    fileName IN VARCHAR2);
```

Transforms input XMLDocument using given URL and the stylesheet, and writes the output to the specified file in a specified directory:

```sql
DBMS_XSLPROCESSOR.PROCESSXSL(
    p IN Processor,
    ss IN Stylesheet,
    url IN VARCHAR2,
    dir IN VARCHAR2,
    fileName IN VARCHAR2);
```

Transforms input XMLDocument using given DOMDOCUMENT and the stylesheet, and writes the output to a CLOB:

```sql
DBMS_XSLPROCESSOR.PROCESSXSL(
    p IN Processor,
    ss IN Stylesheet,
    xmldoc IN DOMDOCUMENT,
    cl IN OUT CLOB);
```

Transforms input XMLDocument using given DOMDOCUMENTFRAGMENT and the stylesheet, and returns the resultant document fragment:

```sql
DBMS_XSLPROCESSOR.PROCESSXSL(
    p IN Processor,
    ss IN Stylesheet,
    xmldf IN DOMDOCUMENTFRAGMENT)
RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input XMLDocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to the specified file in a specified directory:

```sql
DBMS_XSLPROCESSOR.PROCESSXSL(
    p IN Processor,
    ss IN Stylesheet,
    xmldf IN DOMDOCUMENTFRAGMENT,
    dir IN VARCHAR2,
    filename IN VARCHAR2);
```

Transforms input XMLDocumentFragment using given DOMDOCUMENTFRAGMENT and the stylesheet, and writes the output to a buffer:

```sql
DBMS_XSLPROCESSOR.PROCESSXSL(
    p IN Processor,
    ss IN Stylesheet,
    cl IN OUT CLOB);
```
Transforms input XMLDocumentFragment using given DOMDOCUMENTFRAGMENT and the stylesheet, and writes the output to a CLOB:

```
DBMS_XSLPROCESSOR.PROCESSXSL(
  p          IN      Processor,
  ss         IN      Stylesheet,
  xmldf      IN      DOMDOCUMENTFRAGMENT,
  cl         IN OUT  CLOB);
```

### Parameters

#### Table 211-6  PROCESSXSL Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Processor instance</td>
</tr>
<tr>
<td>ss</td>
<td>Stylesheet instance</td>
</tr>
<tr>
<td>xmldoc</td>
<td>XML document being transformed</td>
</tr>
<tr>
<td>url</td>
<td>URL for the information being transformed</td>
</tr>
<tr>
<td>clb</td>
<td>CLOB containing information to be transformed</td>
</tr>
<tr>
<td>dir</td>
<td>Directory where processing output file is saved</td>
</tr>
<tr>
<td>filename</td>
<td>Processing output file</td>
</tr>
<tr>
<td>cl</td>
<td>CLOB to which the processing output is saved</td>
</tr>
<tr>
<td>xmldf</td>
<td>XMLDocumentFragment being transformed</td>
</tr>
</tbody>
</table>

### 211.4.7 READ2CLOB Function

This function reads content of a file into a CLOB.

#### Note:

This procedure has moved to the DBMS_LOB package, starting in Oracle Database 12c release 12.2.

#### Syntax

```
DBMS_XSLPROCESSOR.READ2CLOB(
  flocation     IN   VARCHAR2,
  fname         IN   VARCHAR2,
  csid          IN   NUMBER:=0)
RETURN CLOB;
```
Parameters

Table 211-7  READ2CLOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flocation</td>
<td>File directory</td>
</tr>
<tr>
<td>fname</td>
<td>File name</td>
</tr>
<tr>
<td>csid</td>
<td>Character set id of the file</td>
</tr>
<tr>
<td></td>
<td>• Must be a valid Oracle id; otherwise returns an error</td>
</tr>
<tr>
<td></td>
<td>• If 0, input file is assumed to be in the database character set</td>
</tr>
</tbody>
</table>

211.4.8 REMOVEPARAM Procedure

This procedure removes a top level stylesheet parameter.

Syntax

DBMS_XSLPROCESSOR.REMOVEPARAM(
  ss     IN  Stylesheet,
  name   IN  VARCHAR2);

Parameters

Table 211-8  REMOVEPARAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss</td>
<td>Stylesheet instance</td>
</tr>
<tr>
<td>name</td>
<td>Name of the parameter</td>
</tr>
</tbody>
</table>

211.4.9 RESETPARAMS Procedure

This procedure resets the top-level stylesheet parameters.

Syntax

DBMS_XSLPROCESSOR.RESETPARAMS(
  ss IN Stylesheet);

Parameters

Table 211-9  RESETPARAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss</td>
<td>Stylesheet instance</td>
</tr>
</tbody>
</table>
211.4.10 SELECTNODES Function

This function selects nodes which match the supplied path expression from a DOM tree, and returns the result of the selection.

Syntax

```sql
DBMS_XSLPROCESSOR.SELECTNODES(
    n           IN   DBMS_XMLDOM.DOMNODE,
    pattern     IN   VARCHAR2,
    namespace   IN VARCHAR2 := NULL)
RETURN DBMS_XMLDOM.DOMNODELIST;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Root DOMNode of the tree</td>
</tr>
<tr>
<td>pattern</td>
<td>Pattern to use</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace declared</td>
</tr>
</tbody>
</table>

211.4.11 SELECTSINGLENODE Function

This function selects the first node from the tree that match the supplied path expression, and returns that node.

Syntax

```sql
DBMS_XSLPROCESSOR.SELECTSINGLENODE(
    n           IN   DBMS_XMLDOM.DOMNODE,
    pattern     IN   VARCHAR2,
    namespace   IN VARCHAR2 := NULL)
RETURN DBMS_XMLDOM.DOMNODE;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Root DOMNode of the tree</td>
</tr>
<tr>
<td>pattern</td>
<td>Pattern to use</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace declared</td>
</tr>
</tbody>
</table>
211.4.12 SETERRORLOG Procedure

This deprecated procedure sets errors to be sent to the specified file.

**Note:**

This subprogram has been deprecated, and is included only for reasons of backward compatibility.

**Syntax**

```sql
DBMS_XSLPROCESSOR.SETERRORLOG(
    p         IN   Processor,
    fileName  IN   VARCHAR2);
```

**Parameters**

**Table 211-12 SETERRORLOG Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Processor instance</td>
</tr>
<tr>
<td>fileName</td>
<td>Complete path of the file to use as the error log</td>
</tr>
</tbody>
</table>

211.4.13 SETPARAM Procedure

This procedure sets a top level parameter in the stylesheet.

The parameter value must be a valid XPath expression. Literal string values must be quoted.

**Syntax**

```sql
DBMS_XSLPROCESSOR.SETPARAM(
    ss      IN   Stylesheet,
    name    IN   VARCHAR2,
    value   IN   VARCHAR2);
```

**Parameters**

**Table 211-13 SETPARAM Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss</td>
<td>Stylesheet instance</td>
</tr>
<tr>
<td>name</td>
<td>Name of the parameter</td>
</tr>
<tr>
<td>value</td>
<td>Value of the parameter</td>
</tr>
</tbody>
</table>
211.4.14 SHOWWARNINGS Procedure

This procedure turns warnings on (TRUE) or off (FALSE).

Syntax

```sql
DBMS_XSLPROCESSOR.SHOWWARNINGS(
    p     IN   Processor,
    yes   IN   BOOLEAN);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Processor instance</td>
</tr>
<tr>
<td>yes</td>
<td>Mode to set: TRUE to show warnings, FALSE otherwise</td>
</tr>
</tbody>
</table>

211.4.15 TRANSFORMNODE Function

This function transforms a node in a DOM tree using the given stylesheet, and returns the result of the transformation as a DOMDocumentFragment.

Syntax

```sql
DBMS_XSLPROCESSOR.TRANSFORMNODE(
    n    IN  DOMNODE,
    ss   IN  Stylesheet)
RETURN DOMDocumentFragment;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>DOMNode to transform</td>
</tr>
<tr>
<td>ss</td>
<td>Stylesheet to use</td>
</tr>
</tbody>
</table>

211.4.16 VALUEOF Function and Procedure

This subprogram retrieves the value of the first node from the tree that matches the given pattern. You can use either a function or a procedure.

Syntax

```sql
DBMS_XSLPROCESSOR.VALUEOF(
    n           IN    DBMS_XMLDOM.DOMNODE,
    pattern     IN    VARCHAR2,  
    namespace   IN    VARCHAR2 := NULL)
RETURN VARCHAR2;
```
DBMS_XSLPROCESSOR.VALUEOF(
    n       IN    DBMS_XMLDOM.DOMNODE,
    pattern IN    VARCHAR2,
    val    OUT    VARCHAR2,
    namespace IN    VARCHAR2 := NULL);

Parameters

Table 211-16  VALUEOF Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Node whose value is being retrieved</td>
</tr>
<tr>
<td>pattern</td>
<td>Pattern to use</td>
</tr>
<tr>
<td>val</td>
<td>Retrieved value</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace to use</td>
</tr>
</tbody>
</table>
This **DBMS_XSTREAM_ADM** package provides interfaces for streaming database changes between an Oracle database and other systems. XStream enables applications to stream out or stream in database changes.

This chapter contains the following topic:

- **Overview**
- **Security Model**
- **Operational Notes**
- **Summary of DBMS_XSTREAM_ADM Subprograms**

### See Also:

- *Oracle Database XStream Guide*
- *Oracle Call Interface Programmer's Guide*
- *Oracle Database XStream Java API Reference*

#### 212.1 DBMS_XSTREAM_ADM Overview

The package provides interfaces for configuring outbound servers that stream database changes from an Oracle database to other systems. The package also provides interfaces for configuring inbound servers that stream database changes from other systems to an Oracle database.

In both cases, the database changes are encapsulated in logical change records (LCRs). Also, the other systems can be Oracle systems or a non-Oracle systems, such as non-Oracle databases or file systems.

XStream outbound servers can stream out LCRs from an Oracle database programatically using C or Java. After receiving the LCRs, the other system can process them in any customized way. For example, the other system can save the contents of the LCRs to a file, send the LCRs to an Oracle database through an XStream inbound server, or generate SQL statements and execute them on any Oracle or non-Oracle databases.

XStream inbound servers accept LCRs from another system and either apply them to an Oracle database or process them in a customized way using apply handlers.

XStream can be used in a multitenant container database (CDB). A CDB is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs).
212.2 DBMS_XSTREAM_ADM Security Model

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an XStream administrator and connect as the XStream administrator when using this package.

An administrator must be granted the **DBA** role when the administrator is performing any of the following actions:

- Running the `ADD_OUTBOUND` procedure while connected as a user that is different from the configured connect user for an outbound server
- Running the `ALTER_OUTBOUND` procedure to change the capture user for a capture process or the connect user for an outbound server
- Running the `CREATE_OUTBOUND` procedure, because this procedure creates a capture process
- Running the `ALTER_INBOUND` procedure to change the apply user for an inbound server
- Running the `ADD_INBOUND` procedure while connected as a user that is different from the configured apply user for an inbound server

When the administrator does not need to perform the preceding tasks, the **DBA** role is not required.

See Also:

- `GRANT_ADMIN_PRIVILEGE` Procedure
- *Oracle Database XStream Guide, Chapter 4, "XStream Out and Security"* for more information about XStream and security.
212.3 DBMS_XSTREAM_ADM Operational Notes

Some subprograms in the DBMS_APPLY_ADM package can manage XStream outbound servers, and some subprograms in the DBMS_APPLY_ADM package can manage XStream inbound servers.

- **ADD_COLUMN Procedure**: Either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule.
- **ADD_GLOBAL_PROPAGATION_RULES Procedure**: Either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.
- **ADD_GLOBAL_RULES Procedure**: Adds global rules to either the positive or negative rule set of a capture process or apply process, and creates the specified capture process or apply process if it does not exist.
- **ADD_OUTBOUND Procedure**: Creates an XStream outbound server that dequeues LCRs from the specified queue.
- **ADD_SCHEMA_PROPAGATION_RULES Procedure**: Either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.
- **ADD_SCHEMA_RULES Procedure**: Adds rules to a rule set of XStream clients.
- **ADD_SUBSET_OUTBOUND_RULES Procedure**: Adds subset rules to an outbound server configuration.
- **ADD_SUBSET_PROPAGATION_RULES Procedure**: Adds subset rules to the positive rule set for a propagation, and creates the specified propagation if it does not exist.
- **ADD_SUBSET_RULES Procedure**: Adds subset rules to the positive rule set of a capture process or apply process, and creates the specified capture process or apply process if it does not exist.

**See Also:**
DBMS_APPLY_ADM for details about which subprograms can manage outbound servers and inbound servers.

212.4 Summary of DBMS_XSTREAM_ADM Subprograms

This table lists the DBMS_XSTREAM_ADM subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_COLUMN Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which adds a</td>
</tr>
<tr>
<td></td>
<td>column to a row logical change record (row LCR) that satisfies the specified</td>
</tr>
<tr>
<td></td>
<td>rule.</td>
</tr>
<tr>
<td>ADD_GLOBAL_PROPAGATION_RULES</td>
<td>Either adds global rules to the positive rule set for a propagation, or</td>
</tr>
<tr>
<td>Procedure</td>
<td>adds global rules to the negative rule set for a propagation, and creates</td>
</tr>
<tr>
<td></td>
<td>the specified propagation if it does not exist.</td>
</tr>
<tr>
<td>ADD_GLOBAL_RULES Procedure</td>
<td>Adds global rules to either the positive or negative rule set of a capture</td>
</tr>
<tr>
<td></td>
<td>process or apply process, and creates the specified capture process or apply</td>
</tr>
<tr>
<td></td>
<td>process if it does not exist.</td>
</tr>
<tr>
<td>ADD_OUTBOUND Procedure</td>
<td>Creates an XStream outbound server that dequeues LCRs from the specified</td>
</tr>
<tr>
<td></td>
<td>queue.</td>
</tr>
<tr>
<td>ADD_SCHEMA_PROPAGATION_RULES</td>
<td>Either adds schema rules to the positive rule set for a propagation, or</td>
</tr>
<tr>
<td>Procedure</td>
<td>adds schema rules to the negative rule set for a propagation, and creates</td>
</tr>
<tr>
<td></td>
<td>the specified propagation if it does not exist.</td>
</tr>
<tr>
<td>ADD_SCHEMA_RULES Procedure</td>
<td>Adds rules to a rule set of XStream clients.</td>
</tr>
<tr>
<td>ADD_SUBSET_OUTBOUND_RULES Procedure</td>
<td>Adds subset rules to an outbound server configuration.</td>
</tr>
<tr>
<td>ADD_SUBSET_PROPAGATION_RULES</td>
<td>Adds subset rules to the positive rule set for a propagation, and creates</td>
</tr>
<tr>
<td>Procedure</td>
<td>the specified propagation if it does not exist.</td>
</tr>
<tr>
<td>ADD_SUBSET_RULES Procedure</td>
<td>Adds subset rules to the positive rule set of a capture process or apply</td>
</tr>
<tr>
<td></td>
<td>process, and creates the specified capture process or apply process if it</td>
</tr>
<tr>
<td></td>
<td>does not exist.</td>
</tr>
</tbody>
</table>
### Table 212-1  (Cont.) DBMS_XSTREAM_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADD_TABLE_PROPAGATION_RULES Procedure</strong></td>
<td>Either adds table rules to the positive rule set for a propagation, or adds table rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist</td>
</tr>
<tr>
<td><strong>ADD_TABLE_RULES Procedure</strong></td>
<td>This procedure adds rules to a rule set of an XStream client.</td>
</tr>
<tr>
<td><strong>ALTER_INBOUND Procedure</strong></td>
<td>Modifies an XStream inbound server</td>
</tr>
<tr>
<td><strong>ALTER_OUTBOUND Procedure</strong></td>
<td>Modifies an XStream outbound server</td>
</tr>
<tr>
<td><strong>CREATE_INBOUND Procedure</strong></td>
<td>Creates an XStream inbound server and its queue</td>
</tr>
<tr>
<td><strong>CREATE_OUTBOUND Procedure</strong></td>
<td>Creates an XStream outbound server, queue, and capture process to enable XStream client applications to stream out Oracle database changes encapsulated in LCRs</td>
</tr>
<tr>
<td><strong>DELETE_COLUMN Procedure</strong></td>
<td>Either adds or removes a declarative rule-based transformation which deletes a column from a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td><strong>DROP_INBOUND Procedure</strong></td>
<td>Removes an inbound server configuration</td>
</tr>
<tr>
<td><strong>DROP_OUTBOUND Procedure</strong></td>
<td>Removes an outbound server configuration</td>
</tr>
<tr>
<td><strong>ENABLE_GG_XSTREAM_FOR_STREAMS Procedure</strong></td>
<td>Enables XStream performance optimizations for Oracle Streams components</td>
</tr>
<tr>
<td><strong>GET_MESSAGE_TRACKING Function</strong></td>
<td>Returns the tracking label for the current session</td>
</tr>
<tr>
<td><strong>GET_TAG Function</strong></td>
<td>Gets the binary tag for all redo entries generated by the current session</td>
</tr>
<tr>
<td><strong>IS_GG_XSTREAM_FOR_STREAMS Function</strong></td>
<td>Returns <strong>TRUE</strong> if XStream performance optimizations are enabled for Oracle Streams components, or returns <strong>FALSE</strong> if XStream performance optimizations are disabled for Oracle Streams components</td>
</tr>
<tr>
<td><strong>KEEP_COLUMNS Procedure</strong></td>
<td>Either adds or removes a declarative rule-based transformation which keeps a list of columns in a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td><strong>MERGE_STREAMS Procedure</strong></td>
<td>Merges a stream flowing from one capture process with a stream flowing from another capture process</td>
</tr>
<tr>
<td><strong>MERGE_STREAMS_JOB Procedure</strong></td>
<td>Determines whether the original capture process and the cloned capture are within the specified merge threshold and, if they are, runs the <strong>MERGE_STREAMS</strong> procedure to merge the two streams</td>
</tr>
<tr>
<td><strong>PURGE_SOURCE_CATALOG Procedure</strong></td>
<td>Removes all Oracle Streams data dictionary information at the local database for the specified object</td>
</tr>
<tr>
<td><strong>RECOVER_OPERATION Procedure</strong></td>
<td>Provides options for a split and merge operation that stopped because it encountered an error. This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation.</td>
</tr>
<tr>
<td><strong>REMOVE_QUEUE Procedure</strong></td>
<td>Removes the specified <strong>ANYDATA</strong> queue</td>
</tr>
</tbody>
</table>
### Table 212-1 (Cont.) DBMS_XSTREAM_ADM Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOVE_RULE Procedure</td>
<td>Removes the specified rule or all rules from the rule set associated with the specified capture process, apply process, or propagation.</td>
</tr>
<tr>
<td>REMOVE_SUBSET_OUTBOUND_RULES Procedure</td>
<td>Removes subset rules from an outbound server configuration</td>
</tr>
<tr>
<td>REMOVE_XSTREAM_CONFIGURATION Procedure</td>
<td>Removes the XStream configuration at the local database</td>
</tr>
<tr>
<td>RENAME_COLUMN Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which renames a column in a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td>RENAME_SCHEMA Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which renames a schema in a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td>RENAME_TABLE Procedure</td>
<td>Either adds or removes a declarative rule-based transformation which renames a table in a row LCR that satisfies the specified rule</td>
</tr>
<tr>
<td>SET_MESSAGE_TRACKING Procedure</td>
<td>Sets the tracking label for logical change records (LCRs) produced by the current session</td>
</tr>
<tr>
<td>SET_PARAMETER Procedure</td>
<td>Sets a parameter for an outbound server, an inbound server, or an outbound server©s capture process</td>
</tr>
<tr>
<td>SET_TAG Procedure</td>
<td>Sets the binary tag for all redo entries subsequently generated by the current session</td>
</tr>
<tr>
<td>SET_UP_QUEUE Procedure</td>
<td>Creates a queue table and a queue for use with the capture, propagate, and apply functionality of XStream</td>
</tr>
<tr>
<td>SPLIT_STREAMS Procedure</td>
<td>Splits one stream flowing from a capture process off from all of the other streams flowing from the capture process</td>
</tr>
<tr>
<td>START_OUTBOUND Procedure</td>
<td>Starts an XStream outbound server</td>
</tr>
<tr>
<td>STOP_OUTBOUND Procedure</td>
<td>Stops an XStream outbound server</td>
</tr>
</tbody>
</table>

**Note:**

All subprograms commit unless specified otherwise.

### 212.4.1 ADD_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to **TRUE**, the rule must be in the positive rule set of an XStream client. XStream clients include capture processes, propagations, and apply processes.

This procedure is overloaded. The `column_value` and `column_function` parameters are mutually exclusive.
Note:

- ADD_COLUMN transformations cannot add columns of the following data types: BLOB, CLOB, NCLOB, BFILE, LONG, LONG RAW, ROWID, user-defined types (including object types, REFS, varrays, nested tables), and Oracle-supplied types (including any types, XML types, spatial types, and media types).

- Declarative transformations can transform row LCRs only. Therefore, a DML rule must be specified when you run this procedure. If a DDL rule is specified, then the procedure raises an error.

See Also:

Oracle Database XStream Guide for more information about declarative rule-based transformations

Syntax

```sql
DBMS_XSTREAM_ADM.ADD_COLUMN(
    rule_name     IN  VARCHAR2,
    table_name    IN  VARCHAR2,
    column_name   IN  VARCHAR2,
    column_value  IN  ANYDATA,
    value_type    IN  VARCHAR2     DEFAULT 'NEW',
    step_number   IN  NUMBER       DEFAULT 0,
    operation     IN  VARCHAR2     DEFAULT 'ADD');
```

```sql
DBMS_XSTREAM_ADM.ADD_COLUMN(
    rule_name        IN  VARCHAR2,
    table_name       IN  VARCHAR2,
    column_name      IN  VARCHAR2,
    column_function  IN  VARCHAR2,
    value_type       IN  VARCHAR2     DEFAULT 'NEW',
    step_number      IN  NUMBER       DEFAULT 0,
    operation        IN  VARCHAR2     DEFAULT 'ADD');
```

Parameters

Table 212-2  ADD_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
### Table 212-2  (Cont.) ADD_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table to which the column is added in the row LCR, specified as [schema_name].object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column added to each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>column_value</td>
<td>The value of the added column. Specify the appropriate ANYDATA function for the column datatype and the column value. For example, if the datatye of the column being added is NUMBER and the value is NULL, then specify the ANYDATA.ConvertNumber(NULL) function. This parameter cannot be specified if the column_function parameter is specified.</td>
</tr>
<tr>
<td>column_function</td>
<td>Either the 'SYSDATE' or the 'SYSTIMESTAMP' SQL function.</td>
</tr>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to add the column to the new values in the row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

### Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the add column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the ADD_COLUMN procedures when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>table_name</th>
<th>column_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all add column transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all add column transformations with the specified step_number for the specified rule.</td>
</tr>
</tbody>
</table>
212.4.2 ADD_GLOBAL_PROPAGATION_RULES Procedure

This procedure either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

Syntax

```
DBMS_XSTREAM_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

```
DBMS_XSTREAM_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```
Parameters

Table 212-3 ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner.</td>
</tr>
<tr>
<td></td>
<td>If the specified propagation does not exist, then the procedure creates it</td>
</tr>
<tr>
<td></td>
<td>automatically.</td>
</tr>
<tr>
<td></td>
<td>If NULL and a propagation exists for the same source queue and</td>
</tr>
<tr>
<td></td>
<td>destination queue (including database link), then the procedure uses this</td>
</tr>
<tr>
<td></td>
<td>propagation.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no propagation exists for the same source queue and</td>
</tr>
<tr>
<td></td>
<td>destination queue (including database link), then the procedure creates a</td>
</tr>
<tr>
<td></td>
<td>propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.].queue_name. The</td>
</tr>
<tr>
<td></td>
<td>current database must contain the source queue, and the queue must be</td>
</tr>
<tr>
<td></td>
<td>ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a source queue named streams_queue in the</td>
</tr>
<tr>
<td></td>
<td>strmadmin schema, enter strmadmin.streams_queue for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link,</td>
</tr>
<tr>
<td></td>
<td>specified as [schema_name.].queue_name[@dblink_name], if the destination</td>
</tr>
<tr>
<td></td>
<td>queue is in a remote database. The queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a destination queue named streams_queue in the</td>
</tr>
<tr>
<td></td>
<td>strmadmin schema and use a database link named dbs2.net, enter strmadmin.</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:streams_queue@dbs2.net">streams_queue@dbs2.net</a> for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>If the database link is omitted, then the procedure uses the global name</td>
</tr>
<tr>
<td></td>
<td>of the current database, and the source queue and destination queue must</td>
</tr>
<tr>
<td></td>
<td>be in the same database.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then</td>
</tr>
<tr>
<td></td>
<td>the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then</td>
</tr>
<tr>
<td></td>
<td>the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
</tbody>
</table>
Table 212-3  (Cont.) ADD_GLOBAL_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the changes originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation. In either case, the system creates the rule set if it does not exist.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| and_condition | If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: 

\[(\text{system
d_condition}) \text{ AND } (\text{and
d_condition})\]  

The variable in the specified condition must be :lcr. For example, to specify that the global rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:  

:lcr.get_tag() = HEXTORAW('02')  

The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure the procedure only generates the appropriate rule.  

Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.  

See Also: Logical Change Record TYPES

| queue_to_queue | If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.  

If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.  

The procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  

- If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
- If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
- If NULL, then the procedure does not change the queue to queue property of the propagation.  

See Also: Oracle Streams Concepts and Administration for more information about queue-to-queue propagations

Usage Notes  
This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.
This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. Each rule has a system-generated rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A propagation uses the rules for filtering.

**Examples**

The following is an example of a global rule condition created for DML changes:

```sql
(:dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET')
```

### 212.4.3 ADD_GLOBAL_RULES Procedure

This procedure adds rules to an XStream clients rule set.

It adds rules to a rule set of one of the following types of XStream clients:

- **When the streams_type parameter is set to capture**, this procedure adds capture process rules for capturing changes to an entire database. This procedure creates the specified capture process if it does not exist.

- **When the streams_type parameter is set to apply** and the streams_name parameter specifies the name of an apply process, outbound server, or inbound server, this procedure adds apply rules for applying all logical change records (LCRs) it receives. The rules can specify that the LCRs must be from a particular source database. This procedure creates an apply process if no apply process, outbound server, or inbound server exists with the specified streams_name. This procedure can add rules to an outbound server or inbound server, but it cannot create an outbound server or inbound server.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

**Note:**

If you add global rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects that are not support by Oracle Streams. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

**Syntax**

```sql
DBMS_XSTREAM_ADM.ADD_GLOBAL_RULES(
    streams_type           IN   VARCHAR2,
    streams_name           IN   VARCHAR2  DEFAULT NULL,
    queue_name             IN   VARCHAR2  DEFAULT 'streams_queue',
    include_dml            IN   BOOLEAN   DEFAULT TRUE,
    include_ddl            IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr     IN   BOOLEAN   DEFAULT FALSE,
)
```
Parameters

Table 212-4  ADD_GLOBAL_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_type</td>
<td>The type of XStream client:</td>
</tr>
<tr>
<td></td>
<td>• Specify capture for a capture process.</td>
</tr>
<tr>
<td></td>
<td>• Specify apply for an apply process.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the capture process or apply process. Do not specify an owner.</td>
</tr>
<tr>
<td></td>
<td>If NULL, if streams_type is capture, and if one relevant capture process</td>
</tr>
<tr>
<td></td>
<td>for the queue exists, then the relevant XStream client is used. If no</td>
</tr>
<tr>
<td></td>
<td>relevant XStream client exists for the queue, then an XStream client is</td>
</tr>
<tr>
<td></td>
<td>created automatically with a system-generated name. If NULL and multiple</td>
</tr>
<tr>
<td></td>
<td>XStream clients of the specified streams_type for the queue exist, then</td>
</tr>
<tr>
<td></td>
<td>the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If NULL, if streams_type is apply, and if one relevant apply process</td>
</tr>
<tr>
<td></td>
<td>exists, then the procedure uses the relevant apply process. The relevant</td>
</tr>
<tr>
<td></td>
<td>apply process is identified in one of the following ways:</td>
</tr>
<tr>
<td></td>
<td>• If one existing apply process has the source database specified in</td>
</tr>
<tr>
<td></td>
<td>source_database and uses the queue specified in queue_name, then the</td>
</tr>
<tr>
<td></td>
<td>procedure uses this apply process.</td>
</tr>
<tr>
<td></td>
<td>• If source_database is NULL and one existing apply process is using the</td>
</tr>
<tr>
<td></td>
<td>queue specified in queue_name, then the procedure uses this apply process.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no relevant apply process exists, then the procedure creates</td>
</tr>
<tr>
<td></td>
<td>an apply process automatically with a system-generated name.</td>
</tr>
<tr>
<td></td>
<td>If NULL and multiple relevant apply processes exist, then the procedure</td>
</tr>
<tr>
<td></td>
<td>raises an error.</td>
</tr>
<tr>
<td></td>
<td>Each apply process must have a unique name.</td>
</tr>
</tbody>
</table>
Table 212-4 (Cont.) ADD_GLOBAL_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>queue_name</strong></td>
<td>The name of the local queue, specified as <code>[schema_name.]queue_name</code>. The current database must contain the queue, and the queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a queue named <em>streams_queue</em> in the <em>strmadmin</em> schema, enter <em>strmadmin.streams_queue</em> for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>For capture process rules, this is the queue into which a capture process enqueues LCRs. For outbound server rules, this is the queue from which the outbound server dequeues LCRs. For inbound server rules, this is the queue into which an inbound server enqueues error transactions.</td>
</tr>
<tr>
<td><strong>include_dml</strong></td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td><strong>include_ddl</strong></td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td><strong>include_tagged_lcr</strong></td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
</tbody>
</table>

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags.
Table 212-4  (Cont.) ADD_GLOBAL_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules.</td>
</tr>
<tr>
<td></td>
<td>For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.</td>
</tr>
<tr>
<td></td>
<td>For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured LCRs, then the apply process can apply LCRs from only one capture process at one source database.</td>
</tr>
<tr>
<td></td>
<td>In a CDB, specify the global name of the container to which the rules pertain. The container can be the root or a PDB. For example, mycdb.example.com or hrpdb.example.com. See Oracle Database XStream Guide for more information about setting this parameter in a CDB.</td>
</tr>
<tr>
<td></td>
<td>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name.  If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name.  If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the XStream client.  If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the XStream client.  In either case, the system creates the rule set if it does not exist.</td>
</tr>
</tbody>
</table>
**Table 212-4  (Cont.) ADD_GLOBAL_RULES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and_condition</td>
<td>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</td>
</tr>
<tr>
<td></td>
<td>((\text{system_condition}) \text{ AND } (\text{and_condition}))</td>
</tr>
<tr>
<td></td>
<td>The variable in the specified condition must be :lcr. For example, to specify that the global rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</td>
</tr>
<tr>
<td></td>
<td>:lcr.get_tag() = HEXTORAW('02')</td>
</tr>
<tr>
<td></td>
<td>The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</td>
</tr>
<tr>
<td></td>
<td>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</td>
</tr>
</tbody>
</table>

**See Also:** Logical Change Record TYPES

<table>
<thead>
<tr>
<th>source_root_name</th>
<th>The global name of the root in the source CDB. For example, mycdb.example.com.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If this parameter is NULL, then the global name of the root in the local CDB is used. If you are configuring downstream capture, then this parameter must be a non-NULL value, and it must specify the global name of the root in the remote source CDB. See Oracle Database XStream Guide for more information about setting this parameter in a CDB.</td>
</tr>
<tr>
<td></td>
<td>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This parameter only applies to a CDB.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source_container_name</th>
<th>The short name of the source container. The container can be the root or a PDB. For example, CDB$ROOT or hrpdb. See Oracle Database XStream Guide for more information about setting this parameter in a CDB.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Note:</strong> This parameter only applies to a CDB.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. Each rule has a system-generated rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A capture process or apply process uses the rules for filtering.
Examples

The following is an example of a global rule condition created for DML changes:

\[ (:\text{dml.is_null_tag()}) = 'Y' \text{ and } :\text{dml.get_source_database_name()} = 'DBS1.NET' \]

212.4.4 ADD_OUTBOUND Procedure

This procedure creates an XStream outbound server that dequeues LCRs from the specified queue. The outbound server streams out the LCRs to an XStream client application.

This procedure creates neither a capture process nor a queue. To create an outbound server, a capture process, and a queue with one procedure call, use the CREATE_OUTBOUND Procedure.

To create the capture process individually, use one of the following packages:

- DBMS_XSTREAM_ADM
- DBMS_CAPTURE_ADM

To create a queue individually, use the SET_UP_QUEUE procedure in the DBMS_XSTREAM_ADM package.

This procedure is overloaded. One table_names parameter is type VARCHAR2 and the other table_names parameter is type DBMS.Utility.UNCL_ARRAY. Also, one schema_names parameter is type VARCHAR2 and the other schema_names parameter is type DBMS.Utility.UNCL_ARRAY. These parameters enable you to enter the lists of tables and schemas in different ways and are mutually exclusive.

Note:

- A client application can create multiple sessions. Each session can attach to only one outbound server, and each outbound server can serve only one session at a time. However, different client application sessions can connect to different outbound servers. See Oracle Call Interface Programmer's Guide and Oracle Database XStream Java API Reference for information about attaching to an outbound server.

- This procedure enables the outbound server that it creates.

- Starting with Oracle Database 11g Release 2 (11.2.0.2), the capture_name, start_scn, and start_time parameters are included in this procedure.
Syntax

DBMS_XSTREAM_ADM.ADD_OUTBOUND(
    server_name            IN  VARCHAR2,
    queue_name             IN  VARCHAR2   DEFAULT NULL,
    source_database        IN  VARCHAR2   DEFAULT NULL,
    table_names            IN  DBMS_UTILITY.UNCL_ARRAY,
    schema_names           IN  DBMS_UTILITY.UNCL_ARRAY,
    connect_user           IN  VARCHAR2   DEFAULT NULL,
    comment                IN  VARCHAR2   DEFAULT NULL,
    capture_name           IN  VARCHAR2   DEFAULT NULL,
    start_scn              IN  NUMBER     DEFAULT NULL,
    start_time             IN  TIMESTAMP  DEFAULT NULL,
    include_dml            IN  BOOLEAN   DEFAULT TRUE,
    include_ddl            IN  BOOLEAN   DEFAULT FALSE,
    source_root_name       IN  VARCHAR2  DEFAULT NULL,
    source_container_name  IN  VARCHAR2  DEFAULT NULL,
    lcrid_version          IN  NUMBER    DEFAULT NULL);

Parameters

Table 212-5  ADD_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the outbound server being created. A NULL specification is not allowed. Do not specify an owner.</td>
</tr>
<tr>
<td>queue_name</td>
<td>The name of the local queue from which the outbound server dequeues LCRs, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANyyDATA type.</td>
</tr>
<tr>
<td>source_database</td>
<td>Note: The server_name setting cannot exceed 30 bytes, and it cannot be altered after the outbound server is created.</td>
</tr>
<tr>
<td>table_names</td>
<td>For example, to specify a queue named xstream_queue in the xstrmadmin schema, enter xstrmadmin.xstream_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>schema_names</td>
<td>If NULL, the procedure raises an error.</td>
</tr>
</tbody>
</table>
## Table 212-5 (Cont.) ADD_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>source_database</strong></td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. In a CDB, specify the global name of the container to which the rules pertain. The container can be the root or a PDB. For example, mycdb.example.com or hrpdb.example.com. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
</tbody>
</table>
| **table_names**   | The tables for which data manipulation language (DML) and data definition language (DDL) changes are streamed out to the XStream client application. The tables can be specified in the following ways:  
  - Comma-delimited list of type VARCHAR2.  
  - A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table. Specify the first table in position 1. The last position must be NULL.  
  Each table should be specified as [schema_name.]table_name. For example, you can specify hr.employees. If the schema is not specified, then the current user is the default. |
| **schema_names**  | The schemas for which DML and DDL changes are streamed out to the XStream client application. The schemas can be specified in the following ways:  
  - Comma-delimited list of type VARCHAR2.  
  - A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a schema. Specify the first schema in position 1. The last position must be NULL.  
  Note: This procedure does not concatenate the schema_names parameter with the table_names parameter. To specify tables, enter fully qualified table names in the table_names parameter (schema_name.table_name). |
| **connect_user** | The user who can attach to the specified outbound server to retrieve the LCR stream. The client application must attach to the outbound server as the specified connect user. See “CREATE_OUTBOUND Procedure” for information about the privileges required by a connect user.  
  If NULL, then the current user is the default. |
| **comment**      | An optional comment associated with the outbound server. |
### Table 212-5  (Cont.) ADD_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the capture process configured to capture changes for the outbound server. Do not specify an owner. If the specified name matches the name of an existing capture process for another outbound server, then the procedure uses the existing capture process and adds the rules for capturing changes to the database to the positive capture process rule set. If the specified name matches the name of an existing capture process for an apply process, then an error is raised. If the specified name does not match the name of an existing capture process, then an error is raised. If NULL, then the outbound server is created without a capture process.</td>
</tr>
<tr>
<td>start_scn</td>
<td>A valid system change number (SCN) for the database from which the capture process starts capturing changes. If the <code>capture_name</code> parameter is NULL, then this parameter is ignored. If NULL and the <code>capture_name</code> parameter is non-NULL, then the start SCN of the capture process is not changed. An error is returned if an invalid SCN is specified. The <code>start_scn</code> and <code>start_time</code> parameters are mutually exclusive.</td>
</tr>
<tr>
<td>start_time</td>
<td>A valid time from which the capture process starts capturing changes. If the <code>capture_name</code> parameter is NULL, then this parameter is ignored. If NULL and the <code>capture_name</code> parameter is non-NULL, then the start SCN of the capture process is not changed. The <code>start_scn</code> and <code>start_time</code> parameters are mutually exclusive.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td>source_root_name</td>
<td>The global name of the root in the source CDB. For example, mycdb.example.com. If this parameter is NULL, then the global name of the root in the local CDB is used. If you are configuring downstream capture, then this parameter must be a non-NULL value, and it must specify the global name of the root in the remote source CDB. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. Note: This parameter only applies to a CDB.</td>
</tr>
<tr>
<td>source_container_name</td>
<td>The short name of the source container. The container can be the root or a PDB. For example, CDB$ROOT or hrpdb. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. Note: This parameter only applies to a CDB.</td>
</tr>
</tbody>
</table>
Table 212-5  (Cont.) ADD_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcrid_version</td>
<td>The LCRID version for captured LCRs, either 1 or 2.</td>
</tr>
<tr>
<td></td>
<td>If 2, then the LCRs are compatible with a database with its compatibility level at 12.2.0 or higher.</td>
</tr>
<tr>
<td></td>
<td>If 1, then the LCRs are compatible with a database with its compatibility level at 12.1.0 or lower.</td>
</tr>
<tr>
<td></td>
<td>If NULL, the default, and the database compatibility level is 12.2.0 or higher, then the lcrid_version is set to 2 internally. If the database</td>
</tr>
<tr>
<td></td>
<td>compatibility level is 12.1.0 or lower, then the lcrid_version is set to 1 internally.</td>
</tr>
</tbody>
</table>

Usage Notes

The following list describes the behavior of the outbound server for various combinations of the table_names and schema_names parameters:

• If both the table_names and schema_names parameters are NULL or empty, then the outbound server streams all DML and DDL changes to the client application.

  This procedure is overloaded. The table_names and schema_names parameters are defaulted to NULL. Do not specify NULL for both table_names and schema_names in the same call; otherwise, error PLS-00307 is returned.

• If both the table_names and schema_names parameters are specified, then the outbound server streams DML and DDL changes for the specified tables and schemas.

• If the table_names parameter is specified and the schema_names parameter is NULL or empty, then the outbound server streams DML and DDL changes for the specified tables.

• If the table_names parameter is NULL or empty and the schema_names parameter is specified, then the outbound server streams DML and DDL changes for the specified schemas.

For the procedure that uses the DBMS_UTILITY.UNCL_ARRAY type for the table_names and schema_names parameters, both parameters must be specified. To specify only tables, the schema_names parameter must be specified and empty. To specify only schemas, the table_names parameter must be specified and empty.

Note:

An empty array includes one NULL entry.

212.4.5 ADD_SCHEMA_PROPAGATION_RULES Procedure

This procedure either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.
Syntax

```
DBMS_XSTREAM_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

```
DBMS_XSTREAM_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```

Parameters

Table 212-6 ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema. For example, hr.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically.</td>
</tr>
<tr>
<td></td>
<td>If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue, and the queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
Table 212-6 (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| destination_queue_name | The name of the destination queue, including a database link, specified as [schema_name.]queue_name@dblink_name, if the destination queue is in a remote database. The queue must be ANYDATA type.  
  For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter strmadmin.streams_queue@dbs2.net for this parameter.  
  If the schema is not specified, then the current user is the default.  
  If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.  
  **Note:** Connection qualifiers are not allowed. |
| include_dml            | If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.                                                                     |
| include_ddl            | If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.                                                                     |
| include_tagged_lcr     | If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag.  
  If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag.  
  If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database.  
  If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR.  
  If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag.  
  If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag.  
  If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database.  
  If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag.  
  Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.  
  **See Also:** Oracle Streams Replication Administrator’s Guide for more information about tags |
Table 212-6  (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| source_database    | The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules.  
If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.  
Oracle recommends that you specify a source database for propagation rules.                                                                 |
| dml_rule_name      | If include_dml is TRUE, then this parameter contains the DML rule name.  
If include_dml is FALSE, then this parameter contains a NULL.                                                                                                                                                 |
| ddl_rule_name      | If include_ddl is TRUE, then this parameter contains the DDL rule name.  
If include_ddl is FALSE, then this parameter contains a NULL.                                                                                                                                                 |
| inclusion_rule     | If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation.  
If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation.  
In either case, the system creates the rule set if it does not exist.                                                                                                                                               |
Table 212-6  (Cont.) ADD_SCHEMA_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| and_condition        | If non-NULL, appends the specified condition to the system-generated rule condition using an **AND** clause in the following way:  
  
  \[(system\_condition) \text{ AND } (and\_condition)\]  
  
  The variable in the specified condition must be :lcr. For example, to specify that the schema rules generated by the procedure evaluate to **TRUE** only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:  
  
  :lcr.get_tag() = HEXTORAW('02')  
  
  The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.  
  
  Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify **TRUE** for the include_dml parameter and **FALSE** for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify **FALSE** for the include_dml parameter and **TRUE** for the include_ddl parameter.  
  
  **See Also:** Logical Change Record TYPES  

| queue_to_queue       | If **TRUE** or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.  
  
  If **FALSE**, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.  
  
  This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  
  
  - **If TRUE** and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
  - **If FALSE** and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
  - **If NULL**, then the procedure does not change the queue to queue property of the propagation.  
  
  **See Also:** Oracle Streams Concepts and Administration for more information about queue-to-queue propagations |
Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A propagation uses the rules for filtering.

Examples

The following is an example of a schema rule condition created for DML changes:

```
(:dml.get_object_owner() = 'HR') and
:dml.is_null_tag() = 'Y'
and
:dml.get_source_database_name() = 'DBS1.NET'
```

212.4.6 ADD_SCHEMA_RULES Procedure

This procedure adds rules to a rule set of one of the following types of XStream clients:

- **When the streams_type parameter is set to capture**, this procedure adds capture process rules for capturing changes to a specified schema.
  
  This procedure creates the specified capture process if it does not exist.

- **When the streams_type parameter is set to apply** and the streams_name parameter specifies the name of an apply process, outbound server, or inbound server, this procedure adds apply rules for applying logical change records (LCRs) that contain changes to a specified schema. The rules can specify that the LCRs must be from a particular source database.

  This procedure creates an apply process if no apply process, outbound server, or inbound server exists with the specified streams_name. This procedure can add rules to an outbound server or inbound server, but it cannot create an outbound server or inbound server.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

**Note:**

If you add schema rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects in the schema that are not supported by Oracle Streams. Query the DBA_STREAMS_UNSUPPORTED data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
Syntax

DBMS_XSTREAM_ADM.ADD_SCHEMA_RULES(
    schema_name            IN   VARCHAR2,
    streams_type           IN   VARCHAR2,
    streams_name           IN   VARCHAR2  DEFAULT NULL,
    queue_name             IN   VARCHAR2  DEFAULT 'streams_queue',
    include_dml            IN   BOOLEAN   DEFAULT TRUE,
    include_ddl            IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr     IN   BOOLEAN   DEFAULT FALSE,
    source_database        IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name          OUT  VARCHAR2,
    ddl_rule_name          OUT  VARCHAR2,
    inclusion_rule         IN   BOOLEAN   DEFAULT TRUE,
    and_condition          IN   VARCHAR2  DEFAULT NULL,
    source_root_name       IN  VARCHAR2  DEFAULT NULL,
    source_container_name  IN  VARCHAR2  DEFAULT NULL);

Parameters

Table 212-7 ADD_SCHEMA_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>The name of the schema. For example, hr.</td>
</tr>
<tr>
<td></td>
<td>You can specify a schema that does not yet exist, because Oracle Streams does not validate the existence of the schema.</td>
</tr>
<tr>
<td>streams_type</td>
<td>The type of XStream client:</td>
</tr>
<tr>
<td></td>
<td>Specify capture for a capture process.</td>
</tr>
<tr>
<td></td>
<td>Specify apply for an apply process.</td>
</tr>
</tbody>
</table>

Chapter 212
Summary of DBMS_XSTREAM_ADM Subprograms
Table 212-7  (Cont.) ADD_SCHEMA_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| streams_name | The name of the capture process or apply process. Do not specify an owner. If NULL, if streams_type is capture, and if one relevant capture process for the queue exists, then the relevant XStream client is used. If no relevant XStream client exists for the queue, then an XStream client is created automatically with a system-generated name. If NULL and multiple XStream clients of the specified streams_type for the queue exist, then the procedure raises an error. If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:  
  • If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.  
  • If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process. If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error.  
Each apply process must have a unique name.                                                                 |
| queue_name   | The name of the local queue, specified as [schema_name.]queue_name.  
The current database must contain the queue, and the queue must be ANYDATA type.  
For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.  
For capture process rules, this is the queue into which a capture process enqueues LCRs. For outbound server rules, this is the queue from which the outbound server dequeues LCRs. For inbound server rules, this is the queue into which an inbound server enqueues error transactions. If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error.  
Each apply process must have a unique name. |
| include_dml  | If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted. |
| include_ddl  | If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted. |
### Table 212-7 (Cont.) ADD_SCHEMA_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>include_tagged_lcr</code></td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE. <strong>See Also:</strong> <a href="#">Oracle Streams Replication Administrator’s Guide</a> for more information about tags.</td>
</tr>
<tr>
<td><code>source_database</code></td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured LCRs, then the apply process can apply LCRs from only one capture process at one source database. In a CDB, specify the global name of the container to which the rules pertain. The container can be the root or a PDB. For example, <code>mycdb.example.com</code> or <code>hrpdb.example.com</code>. <strong>See Oracle Database XStream Guide</strong> for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</td>
</tr>
<tr>
<td><code>dml_rule_name</code></td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
</tbody>
</table>
Table 212-7  (Cont.) ADD_SCHEMA_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the XStream client. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the XStream client. In either case, the system creates the rule set if it does not exist.</td>
</tr>
<tr>
<td>and_condition</td>
<td>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (system_condition) AND (and_condition) The variable in the specified condition must be :lcr. For example, to specify that the schema rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW('02') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</td>
</tr>
<tr>
<td>source_root_name</td>
<td>The global name of the root in the source CDB. For example, mycdb.example.com. If this parameter is NULL, then the global name of the root in the local CDB is used. If you are configuring downstream capture, then this parameter must be a non-NULL value, and it must specify the global name of the root in the remote source CDB. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically.</td>
</tr>
<tr>
<td>source_container_name</td>
<td>The short name of the source container. The container can be the root or a PDB. For example, CDB$ROOT or hrpdb. See Oracle Database XStream Guide for more information about setting this parameter in a CDB.</td>
</tr>
</tbody>
</table>

Note: This parameter only applies to a CDB.

Usage Notes

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule.
name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A capture process or apply process uses the rules for filtering.

See Also:

- "Operational Notes"
- "Security Model"

Examples

The following is an example of a schema rule condition created for DML changes:

```
((:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET')
```

212.4.7 ADD_SUBSET_OUTBOUND_RULES Procedure

This procedure adds subset rules to an outbound server configuration. Subset rules instruct the outbound server to stream out a subset of the changes to the specified tables. Outbound servers can stream out a subset of both rows and columns.

This procedure is overloaded. One column_list parameter is type VARCHAR2 and the other column_list parameter is type DBMS_UTILITY.LNAME_ARRAY. These parameters enable you to enter the list of columns in different ways and are mutually exclusive.

Note:

This procedure does not add rules to the outbound server's capture process.

Syntax

```
DBMS_XSTREAM_ADM.ADD_SUBSET_OUTBOUND_RULES(
    server_name      IN VARCHAR2,
    table_name       IN VARCHAR2,
    condition        IN VARCHAR2  DEFAULT NULL,
    column_list      IN DBMS_UTILITY.LNAME_ARRAY,
    keep             IN BOOLEAN   DEFAULT TRUE,
    source_database  IN VARCHAR2  DEFAULT NULL);
```

```
DBMS_XSTREAM_ADM.ADD_SUBSET_OUTBOUND_RULES(
    server_name      IN VARCHAR2,
    table_name       IN VARCHAR2,
    condition        IN VARCHAR2  DEFAULT NULL,
    column_list      IN VARCHAR2  DEFAULT NULL,
    keep             IN BOOLEAN   DEFAULT TRUE,
    source_database  IN VARCHAR2  DEFAULT NULL);
```
## Parameters

Table 212-8  ADD_SUBSET_OUTBOUND_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the outbound server to which rules are being added. Specify an existing outbound server. Do not specify an owner.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name.]object_name. For example, you can specify hr.employees. If the schema is not specified, then the current user is the default. If the outbound server configuration uses a local capture process, then the table must exist at the local source database. If the outbound server configuration uses a downstream capture process, then the table must exist at both the source database and at the downstream capture database. The specified table cannot have any LOB, LONG, or LONG RAW columns currently or in the future.</td>
</tr>
<tr>
<td>condition</td>
<td>The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL. For example, to specify rows in the hr.employees table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition: 'salary &gt; 4000 and job_id = ''SA_MAN'' ' If NULL, then the procedure raises an error. Note: The quotation marks in the preceding example are all single quotation marks.</td>
</tr>
<tr>
<td>column_list</td>
<td>The list of columns either to include in the outbound server configuration or to exclude from the outbound server configuration. Whether the columns are included or excluded depends on the setting for the keep parameter. The columns can be specified in the following ways:</td>
</tr>
<tr>
<td></td>
<td>· Comma-delimited list of type VARCHAR2.</td>
</tr>
<tr>
<td></td>
<td>· A PL/SQL associative array of type DBMS_UTILITY.LNAME_ARRAY, where each element is the name of a column. Specify the first column in position 1. The last position must be NULL.</td>
</tr>
<tr>
<td></td>
<td>To include or exclude all of the columns in a table, specify each column in the table in the list or array. If NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>keep</td>
<td>If TRUE, then the columns specified in the column_list parameter are kept as part of the outbound server configuration. Therefore, changes to these columns that satisfy the condition in the condition parameter are streamed to the outbound server's client application. If FALSE, then the columns specified in the column_list parameter are excluded from the outbound server configuration. Therefore, changes to these columns are not streamed to the outbound server's client application. See Also: &quot;Usage Notes&quot;</td>
</tr>
</tbody>
</table>
Table 212-8 (Cont.) ADD_SUBSET_OUTBOUND_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the container where the specified table_names and schema_names are located. If non-NULL, then a condition is added to the outbound server's rules to filter the LCRs based on the global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. In a CDB, specify the global name of the container to which the rules pertain. The container can be the root or a PDB. For example, mycdb.example.com or hrpdb.example.com. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically.</td>
</tr>
</tbody>
</table>

Usage Notes

When the keep parameter is set to TRUE, this procedure creates a keep columns declarative rule-based transformation for the columns listed in column_list.

When the keep parameter is set to FALSE, this procedure creates a delete column declarative rule-based transformation for each column listed in column_list.

See Also:
Oracle Database XStream Guide for information about declarative rule-based transformations

212.4.8 ADD_SUBSET_PROPAGATION_RULES Procedure

This procedures adds propagation rules that propagate the logical change records (LCRs) related to a subset of the rows in the specified table in a source queue to a destination queue, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains three OUT parameters, and the other does not.

Syntax

```
DBMS_XSTREAM_ADM.ADD_SUBSET_PROPAGATION_RULES(
    table_name               IN   VARCHAR2,
    dml_condition            IN   VARCHAR2,
    streams_name             IN   VARCHAR2  DEFAULT NULL,
    source_queue_name        IN   VARCHAR2,
    destination_queue_name   IN   VARCHAR2,
    include_tagged_lcr       IN   BOOLEAN   DEFAULT FALSE,
    source_database          IN   VARCHAR2  DEFAULT NULL,
    insert_rule_name         OUT  VARCHAR2,
    update_rule_name         OUT  VARCHAR2,
)
```
Parameters

Table 212-9  ADD_SUBSET_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default. The specified table must exist in the same database as the propagation. Also, the specified table cannot have any LOB, LONG, LONG RAW, or XMLType columns currently or in the future.</td>
</tr>
<tr>
<td>dml_condition</td>
<td>The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL. For example, to specify rows in the hr.employees table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition: ' salary &gt; 4000 and job_id = ''SA_MAN'' ' Note: The quotation marks in the preceding example are all single quotation marks.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
Table 212-9  (Cont.) ADD_SUBSET_PROPAGATION_RULES Procedure
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link, specified as [schema_name.]queue_name[dblink_name], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter <a href="mailto:strmadmin.streams_queue@dbs2.net">strmadmin.streams_queue@dbs2.net</a> for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. <strong>Note:</strong> Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database. If FALSE, then an LCR is considered for propagation only when the LCR contains a NULL tag. A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database. <strong>See Also:</strong> Oracle Streams Replication Administrator's Guide for more information about tags</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.</td>
</tr>
<tr>
<td>insert_rule_name</td>
<td>Contains the system-generated INSERT rule name. This rule handles inserts and updates that must be converted into inserts.</td>
</tr>
<tr>
<td>update_rule_name</td>
<td>Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.</td>
</tr>
<tr>
<td>delete_rule_name</td>
<td>Contains the system-generated DELETE rule name. This rule handles deletes and updates that must be converted into deletes</td>
</tr>
</tbody>
</table>
Table 212-9  (Cont.) ADD_SUBSET_PROPAGATION_RULES Procedure

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| queue_to_queue        | If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database. If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  
  • If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
  • If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
  • If NULL, then the procedure does not change the queue to queue property of the propagation.                                                                 |
positive rule set for the propagation are not affected. Additional rules can be added using either the DBMS_XSTREAM_ADM package or the DBMS_RULE_ADM package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The ADD_SUBSET_RULES procedure is overloaded, and the system-generated rule names for INSERT, UPDATE, and DELETE statements are returned.

When you create propagation subset rules for a table, you should create an unconditional supplemental log group at the source database with all the columns in the table. Supplemental logging is required if an update must be converted to an insert. The propagation rule must have all the column values to be able to perform this conversion correctly.

Note:
Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so might have unpredictable results because row migration would not be performed on LCRs that are not discarded by the negative rule set.

Examples
The following is an example of a rule condition created for filtering a row LCR containing an update operation when the dml_condition is region_id = 2, the table_name is hr.regions, and the source_database is dbs1.net:

:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)

212.4.9 ADD_SUBSET_RULES Procedure

This procedure adds rules to an XStream client.

It adds rules to a rule set of one of the following types of XStream clients:

- When the streams_type parameter is set to capture, this procedure adds capture process rules for capturing changes to a subset of rows in a specified table.
  This procedure creates the specified capture process if it does not exist.

- When the streams_type parameter is set to apply and the streams_name parameter specifies the name of an apply process, outbound server, or inbound server, this procedure adds apply rules for applying logical change records (LCRs) that contain changes to a subset of rows in a specified table. The rules can specify that the LCRs must be from a particular source database.
This procedure creates an apply process if no apply process, outbound server, or inbound server exists with the specified streams_name. This procedure can add rules to an outbound server or inbound server, but it cannot create an outbound server or inbound server.

This procedure is overloaded. One version of this procedure contains three OUT parameters, and the other does not.

Syntax

```
DBMS_XSTREAM_ADM.ADD_SUBSET_RULES(
    table_name             IN   VARCHAR2,
    dml_condition          IN   VARCHAR2,
    streams_type           IN   VARCHAR2 DEFAULT 'apply',
    streams_name           IN   VARCHAR2 DEFAULT NULL,
    queue_name             IN   VARCHAR2 DEFAULT 'streams_queue',
    include_tagged_lcr     IN   BOOLEAN  DEFAULT FALSE,
    source_database        IN   VARCHAR2 DEFAULT NULL,
    insert_rule_name       OUT  VARCHAR2,
    update_rule_name       OUT  VARCHAR2,
    delete_rule_name       OUT  VARCHAR2,
    source_root_name       IN  VARCHAR2  DEFAULT NULL,
    source_container_name  IN  VARCHAR2  DEFAULT NULL);
```

```
DBMS_XSTREAM_ADM.ADD_SUBSET_RULES(
    table_name             IN   VARCHAR2,
    dml_condition          IN   VARCHAR2,
    streams_type           IN   VARCHAR2 DEFAULT 'apply',
    streams_name           IN   VARCHAR2 DEFAULT NULL,
    queue_name             IN   VARCHAR2 DEFAULT 'streams_queue',
    include_tagged_lcr     IN   BOOLEAN  DEFAULT FALSE,
    source_database        IN   VARCHAR2 DEFAULT NULL,
    source_root_name       IN  VARCHAR2  DEFAULT NULL,
    source_container_name  IN  VARCHAR2  DEFAULT NULL);
```

Parameters

**Table 212-10  ADD_SUBSET_RULES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name].object_name. For example, hr.employees. If the schema is not specified, then the current user is the default. The specified table must exist in the same database as the capture process or apply process. Also, the specified table cannot have any LOB, LONG, LONG RAW, or XMLType columns currently or in the future.</td>
</tr>
<tr>
<td>dml_condition</td>
<td>The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL. For example, to specify rows in the hr.employees table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition: ' salary &gt; 4000 and job_id = ''SA_MAN'' '</td>
</tr>
</tbody>
</table>

**Note:** The quotation marks in the preceding example are all single quotation marks.
### Table 212-10  (Cont.) ADD_SUBSET_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_type</td>
<td>The type of XStream client:</td>
</tr>
<tr>
<td></td>
<td>• Specify capture for a capture process.</td>
</tr>
<tr>
<td></td>
<td>• Specify apply for an apply process.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the capture process or apply process. Do not specify an owner.</td>
</tr>
<tr>
<td></td>
<td>If NULL, if streams_type is capture, and if one relevant capture process for the queue exists, then the procedure uses the relevant XStream client. If no relevant XStream client exists for the queue, then the procedure creates an XStream client automatically with a system-generated name. If NULL and multiple XStream clients of the specified streams_type for the queue exist, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:</td>
</tr>
<tr>
<td></td>
<td>• If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.</td>
</tr>
<tr>
<td></td>
<td>• If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name.</td>
</tr>
<tr>
<td></td>
<td>If NULL and multiple relevant apply processes exist, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>Each apply process must have a unique name.</td>
</tr>
<tr>
<td>queue_name</td>
<td>The name of the local queue, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>For capture process rules, this is the queue into which a capture process enqueues LCRs. For outbound server rules, this is the queue from which the outbound server dequeues LCRs. For inbound server rules, this is the queue into which an inbound server enqueues error transactions.</td>
</tr>
</tbody>
</table>
### Table 212-10 (Cont.) ADD_SUBSET_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the XStream client performs its action regardless of the tag:</td>
</tr>
<tr>
<td></td>
<td>- A redo entry is always considered for capture by a capture process, regardless of whether the redo entry has a non-NULL tag.</td>
</tr>
<tr>
<td></td>
<td>- An LCR is always considered for apply by an apply process, regardless of whether redo entry or LCR has a non-NULL tag.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then an XStream client performs its action only when the tag is NULL:</td>
</tr>
<tr>
<td></td>
<td>- A redo entry is considered for capture by a capture process only when the redo entry contains a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>- An LCR is considered for apply by an apply process only if the LCR contains a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> Oracle Streams Replication Administrator's Guide for more information about tags</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules.</td>
</tr>
<tr>
<td></td>
<td>For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.</td>
</tr>
<tr>
<td></td>
<td>For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured LCRs, then the apply process can apply LCRs from only one capture process at one source database. In a CDB, specify the global name of the container to which the rules pertain. The container can be the root or a PDB. For example, mycdb.example.com or hrpdb.example.com. <strong>See Oracle Database XStream Guide</strong> for more information about setting this parameter in a CDB.</td>
</tr>
<tr>
<td></td>
<td>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>insert_rule_name</td>
<td>Contains the system-generated INSERT rule name. This rule handles inserts and updates that must be converted into inserts.</td>
</tr>
<tr>
<td>update_rule_name</td>
<td>Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.</td>
</tr>
<tr>
<td>delete_rule_name</td>
<td>Contains the system-generated DELETE rule name. This rule handles deletes and updates that must be converted into deletes.</td>
</tr>
</tbody>
</table>
Table 212-10  (Cont.) ADD_SUBSET_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_root_name</td>
<td>The global name of the root in the source CDB. For example, mycdb.example.com. If this parameter is NULL, then the global name of the root in the local CDB is used. If you are configuring downstream capture, then this parameter must be a non-NULL value, and it must specify the global name of the root in the remote source CDB. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. Note: This parameter only applies to a CDB.</td>
</tr>
<tr>
<td>source_container_name</td>
<td>The short name of the source container. The container can be the root or a PDB. For example, CDB$ROOT or hrpdb. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. Note: This parameter only applies to a CDB.</td>
</tr>
</tbody>
</table>

Usage Notes

Running this procedure generates three rules for the specified capture process or apply process: one for INSERT statements, one for UPDATE statements, and one for DELETE statements. For INSERT and DELETE statements, only DML changes that satisfy the condition specified for the dml_condition parameter are captured or applied. For UPDATE statements, the following variations are possible:

- If both the new and old values in a DML change satisfy the specified dml_condition, then the DML change is captured or applied without any changes.
- If neither the new or old values in a DML change satisfy the specified dml_condition, then the DML change is not captured or applied.
- If the old values for a DML change satisfy the specified dml_condition, but the new values do not, then the DML change is converted into a delete.
- If the new values for a DML change satisfy the specified dml_condition, but the old values do not, then the DML change is converted to an insert.

When an update is converted into an insert or a delete, it is called row migration.

A capture process or apply process uses the rules for filtering. If the XStream client does not have a positive rule set, then this procedure creates a positive rule set automatically, and adds the rules for the table to the positive rule set. A subset rule can be added to positive rule set only, not to a negative rule set. Other rules in an existing rule set for the process are not affected. Additional rules can be added using either the DBMS_XSTREAM_ADM package or the DBMS_RULE_ADM package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name
plus the sequence number is too long, then the table name is truncated. The 
**ADD_SUBSET_RULES** procedure is overloaded, and the system-generated rule names for 
**INSERT**, **UPDATE**, and **DELETE** statements are returned.

**Note:**

Subset rules should only reside in positive rule sets. You should not add 
subset rules to negative rule sets. Doing so might have unpredictable results 
because row migration would not be performed on LCRs that are not 
discarded by the negative rule set.

**Examples**

The following is an example of a rule condition created for filtering DML changes 
containing an update operation when the **dml_condition** is `region_id = 2`, the 
table name is `hr.regions`, and the source database is `dbs1.net`:

```sql
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```

---

**212.4.10 ADD_TABLE_PROPAGATION_RULES Procedure**

This procedure adds table rules to the positive rule set for a propagation, or adds 
table rules to the negative rule set for a propagation, and creates the specified 
propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two **OUT** 
parameters, and the other does not.

**Syntax**

```sql
DBMS_XSTREAM_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);

DBMS_XSTREAM_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              IN   VARCHAR2,
    streams_name            IN   VARCHAR2  DEFAULT NULL,
    source_queue_name       IN   VARCHAR2,
    destination_queue_name  IN   VARCHAR2,
    include_dml             IN   BOOLEAN   DEFAULT TRUE,
    include_ddl             IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr      IN   BOOLEAN   DEFAULT FALSE,
    source_database         IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name           OUT  VARCHAR2,
    ddl_rule_name           OUT  VARCHAR2,
    inclusion_rule          IN   BOOLEAN   DEFAULT TRUE,
    and_condition           IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue          IN   BOOLEAN   DEFAULT NULL);
```
include_dml IN BOOLEAN DEFAULT TRUE,
include_ddl IN BOOLEAN DEFAULT FALSE,
include_tagged_lcr IN BOOLEAN DEFAULT FALSE,
source_database IN VARCHAR2 DEFAULT NULL,
inclusion_rule IN BOOLEAN DEFAULT TRUE,
and_condition IN VARCHAR2 DEFAULT NULL,
queue_to_queue IN BOOLEAN DEFAULT NULL);

Parameters

Table 212-11  ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table specified as [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</td>
</tr>
<tr>
<td>source_queue_name</td>
<td>The name of the source queue, specified as [schema_name.]queue_name. The current database must contain the source queue, and the queue must be ANYDATA type. For example, to specify a source queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>destination_queue_name</td>
<td>The name of the destination queue, including a database link, specified as [schema_name.]queue_name[@dblink_name], if the destination queue is in a remote database. The queue must be ANYDATA type. For example, to specify a destination queue named streams_queue in the strmadmin schema and use a database link named dbs2.net, enter <a href="mailto:strmadmin.streams_queue@dbs2.net">strmadmin.streams_queue@dbs2.net</a> for this parameter. If the schema is not specified, then the current user is the default. If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database. <strong>Note:</strong> Connection qualifiers are not allowed.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
</tbody>
</table>
Table 212-11  (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted. The generated rule evaluates to TRUE for any DDL change that operates on the table or on an object that is part of the table, such as an index or trigger on the table. The rule evaluates to FALSE for any DDL change that either does not refer to the table or refers to the table in a subordinate way. For example, the rule evaluates to FALSE for changes that create synonyms or views based on the table. The rule also evaluates to FALSE for a change to a PL/SQL subprogram that refers to the table.</td>
</tr>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR. If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag. Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically. Oracle recommends that you specify a source database for propagation rules.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
</tbody>
</table>
Table 212-11  (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the propagation. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the propagation. In either case, the system creates the rule set if it does not exist.</td>
</tr>
<tr>
<td>and_condition</td>
<td>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way: (system_condition) AND (and_condition) The variable in the specified condition must be :lcr. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition: :lcr.get_tag() = HEXTORAW('02') The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</td>
</tr>
</tbody>
</table>

See Also: Logical Change Record TYPES
Table 212-11  (Cont.) ADD_TABLE_PROPAGATION_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| queue_to_queue  | If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database. If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment. This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:  
  • If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.  
  • If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.  
  • If NULL, then the procedure does not change the queue to queue property of the propagation. |

See Also: Oracle Streams Concepts and Administration for more information about queue-to-queue propagations

Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A propagation uses the rules for filtering.

Examples

The following is an example of a table rule condition created for filtering DML statements:

```sql
{{:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'}
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' }
```

212.4.11 ADD_TABLE_RULES Procedure

This procedure adds rules to an XStream client rule set.

It adds rules to a rule set of one of the following types of XStream clients:

• When the streams_type parameter is set to capture, this procedure adds capture process rules for capturing changes to a specified table.
  
  This procedure creates the specified capture process if it does not exist.
• When the `streams_type` parameter is set to apply and the `streams_name` parameter specifies the name of an apply process, outbound server, or inbound server, this procedure adds apply rules for applying logical change records (LCRs) that contain changes to a specified table. The rules can specify that the LCRs must be from a particular source database.

This procedure creates an apply process if no apply process, outbound server, or inbound server exists with the specified `streams_name`. This procedure can add rules to an outbound server or inbound server, but it cannot create an outbound server or inbound server.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

**Syntax**

```sql
DBMS_XSTREAM_ADM.ADD_TABLE_RULES(
  table_name             IN   VARCHAR2,
  streams_type           IN   VARCHAR2,
  streams_name           IN   VARCHAR2  DEFAULT NULL,
  queue_name             IN   VARCHAR2  DEFAULT 'streams_queue',
  include_dml            IN   BOOLEAN   DEFAULT TRUE,
  include_ddl            IN   BOOLEAN   DEFAULT FALSE,
  include_tagged_lcr     IN   BOOLEAN   DEFAULT FALSE,
  source_database        IN   VARCHAR2  DEFAULT NULL,
  dml_rule_name          OUT  VARCHAR2,
  ddl_rule_name          OUT  VARCHAR2,
  inclusion_rule         IN   BOOLEAN   DEFAULT TRUE,
  and_condition          IN   VARCHAR2  DEFAULT NULL,
  source_root_name       IN  VARCHAR2   DEFAULT NULL,
  source_container_name  IN  VARCHAR2   DEFAULT NULL);

DBMS_XSTREAM_ADM.ADD_TABLE_RULES(
  table_name             IN   VARCHAR2,
  streams_type           IN   VARCHAR2,
  streams_name           IN   VARCHAR2  DEFAULT NULL,
  queue_name             IN   VARCHAR2  DEFAULT 'streams_queue',
  include_dml            IN   BOOLEAN   DEFAULT TRUE,
  include_ddl            IN   BOOLEAN   DEFAULT FALSE,
  include_tagged_lcr     IN   BOOLEAN   DEFAULT FALSE,
  source_database        IN   VARCHAR2  DEFAULT NULL,
  inclusion_rule         IN   BOOLEAN   DEFAULT TRUE,
  and_condition          IN   VARCHAR2  DEFAULT NULL,
  source_root_name       IN  VARCHAR2   DEFAULT NULL,
  source_container_name  IN  VARCHAR2   DEFAULT NULL);
```

**Parameters**

**Table 212-12  ADD_TABLE_RULES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table_name</code></td>
<td>The name of the table specified as <code>[schema_name.]object_name</code>. For example, <code>hr.employees</code>. If the schema is not specified, then the current user is the default. You can specify a table that does not yet exist, because Oracle Streams does not validate the existence of the table.</td>
</tr>
</tbody>
</table>
### Table 212-12  (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_type</td>
<td>The type of XStream client:</td>
</tr>
<tr>
<td></td>
<td>- Specify capture for a capture process.</td>
</tr>
<tr>
<td></td>
<td>- Specify apply for an apply process.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the capture process or apply process. Do not specify an owner.</td>
</tr>
<tr>
<td></td>
<td>If NULL, if streams_type is capture, and if one relevant capture process for the queue exists, then the procedure uses the relevant XStream client. If no relevant XStream client exists for the queue, then the procedure creates an XStream client automatically with a system-generated name. If NULL and multiple XStream clients of the specified streams_type for the queue exist, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:</td>
</tr>
<tr>
<td></td>
<td>- If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.</td>
</tr>
<tr>
<td></td>
<td>- If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process.</td>
</tr>
<tr>
<td></td>
<td>If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name. If NULL and multiple relevant apply processes exist, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>Each apply process must have a unique name.</td>
</tr>
<tr>
<td>queue_name</td>
<td>The name of the local queue, specified as [schema_name.]queue_name. The current database must contain the queue, and the queue must be ANYDATA type.</td>
</tr>
<tr>
<td></td>
<td>For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>For capture process rules, this is the queue into which a capture process enqueues LCRs. For outbound server rules, this is the queue from which the outbound server dequeues LCRs. For inbound server rules, this is the queue into which an inbound server enqueues error transactions.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a DML rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a DDL rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td></td>
<td>The generated rule evaluates to TRUE for any DDL change that operates on the table or on an object that is part of the table, such as an index or trigger on the table. The rule evaluates to FALSE for any DDL change that either does not refer to the table or refers to the table in a subordinate way. For example, the rule evaluates to FALSE for changes that create synonyms or views based on the table. The rule also evaluates to FALSE for a change to a PL/SQL subprogram that refers to the table.</td>
</tr>
</tbody>
</table>
Table 212-12 (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_tagged_lcr</td>
<td>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry, session, or LCR has a non-NULL tag. If the rules are added to the positive rule set for the XStream client, then the XStream client performs its action regardless of the tag:</td>
</tr>
<tr>
<td></td>
<td>• A redo entry is always considered for capture by a capture process, regardless of whether the redo entry has a non-NULL tag.</td>
</tr>
<tr>
<td></td>
<td>• An LCR is always considered for apply by an apply process, regardless of whether redo entry or LCR has a non-NULL tag.</td>
</tr>
<tr>
<td></td>
<td>If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the XStream client, then whether a database change is discarded does not depend on the tag.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry, session, or LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for an XStream client, then the XStream client performs its action only when the tag is NULL:</td>
</tr>
<tr>
<td></td>
<td>• A redo entry is considered for capture by a capture process only when the redo entry contains a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>• An LCR is considered for apply by an apply process only if the LCR contains a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the XStream client, then a database change can be discarded only if it has a NULL tag.</td>
</tr>
<tr>
<td></td>
<td>A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database.</td>
</tr>
<tr>
<td></td>
<td>Usually, specify TRUE for this parameter if the inclusion_rule parameter is set to FALSE.</td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle Streams Replication Administrator's Guide for more information about tags</td>
</tr>
</tbody>
</table>
**Table 212-12  (Cont.) ADD_TABLE_RULES Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured. For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured LCRs, then the apply process can apply LCRs from only one capture process at one source database. In a CDB, specify the global name of the container to which the rules pertain. The container can be the root or a PDB. For example, mycdb.example.com or hrpdb.example.com. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</td>
</tr>
<tr>
<td>dml_rule_name</td>
<td>If include_dml is TRUE, then this parameter contains the DML rule name. If include_dml is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>ddl_rule_name</td>
<td>If include_ddl is TRUE, then this parameter contains the DDL rule name. If include_ddl is FALSE, then this parameter contains a NULL.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the XStream client. If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the XStream client. In either case, the system creates the rule set if it does not exist.</td>
</tr>
</tbody>
</table>
Table 212-12  (Cont.) ADD_TABLE_RULES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and_condition</td>
<td>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</td>
</tr>
<tr>
<td></td>
<td>(system_condition) AND (and_condition)</td>
</tr>
<tr>
<td></td>
<td>The variable in the specified condition must be :lcr. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</td>
</tr>
<tr>
<td></td>
<td>:lcr.get_tag() = HEXTORAW('02')</td>
</tr>
<tr>
<td></td>
<td>The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</td>
</tr>
</tbody>
</table>

See Also: Logical Change Record TYPES

| source_root_name  | The global name of the root in the source CDB. For example, mycdb.example.com. If this parameter is NULL, then the global name of the root in the local CDB is used. If you are configuring downstream capture, then this parameter must be a non-NULL value, and it must specify the global name of the root in the remote source CDB. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. Note: This parameter only applies to a CDB. |

| source_container_name | The short name of the source container. The container can be the root or a PDB. For example, CDB$ROOT or hrpdb. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. Note: This parameter only applies to a CDB. |

Usage Notes

This procedure creates DML and DDL rules automatically based on include_dml and include_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A capture process or apply process uses the rules for filtering.
Examples

The following is an example of a table rule condition created for DML changes:

```sql
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET')
```

### 212.4.12 ALTER_INBOUND Procedure

This procedure modifies an XStream inbound server.

#### Syntax

```sql
DBMS_XSTREAM_ADM.ALTER_INBOUND(
    server_name IN VARCHAR2,
    apply_user  IN VARCHAR2  DEFAULT NULL,
    comment     IN VARCHAR2  DEFAULT NULL);
```

#### Parameters

**Table 212-13  ALTER_INBOUND Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the inbound server being altered. Specify an existing inbound server. Do not specify an owner.</td>
</tr>
<tr>
<td>apply_user</td>
<td>The user who applies all DML and DDL changes that satisfy the inbound server rule sets, who runs user-defined apply handlers, and who runs custom rule-based transformations configured for inbound server rules. The client application must attach to the inbound server as the apply user. Specify a user to change the apply user. In this case, the user who invokes the <strong>ALTER_INBOUND</strong> procedure must be granted the <strong>DBA</strong> role. Only the <strong>SYS</strong> user can set the <strong>apply_user</strong> to <strong>SYS</strong>. If <strong>NULL</strong>, then the apply user is not changed. See &quot;<strong>CREATE_INBOUND Procedure</strong>&quot; for information about the required privileges for an apply user.</td>
</tr>
<tr>
<td>comment</td>
<td>An optional comment associated with the inbound server. If <strong>non-NULL</strong>, then the specified comment replaces the existing comment. If <strong>NULL</strong>, then the existing comment is not changed.</td>
</tr>
</tbody>
</table>
212.4.13 ALTER_OUTBOUND Procedure

This procedure modifies an XStream outbound server configuration.

This procedure always alters the specified outbound server. This procedure can also alter the outbound server's capture process when either of the following conditions is met:

- The capture process was created by the CREATE_OUTBOUND procedure in this package.
- The queue used by the capture process was created by the CREATE_OUTBOUND procedure.

To check whether this procedure can alter the outbound server's capture process, query the CAPTURE_NAME column in the ALL_XSTREAM_OUTBOUND view. When the name of the capture process appears in the CAPTURE_NAME column of this view, the ALTER_OUTBOUND procedure can manage the capture process's rules or change the capture user for the capture process. When the CAPTURE_NAME column of this view is NULL, the ALTER_OUTBOUND procedure cannot manage the capture process.

This procedure is overloaded. One table_names parameter is type VARCHAR2 and the other table_names parameter is type DBMS_UTILITY.UNCL_ARRAY. Also, one schema_names parameter is type VARCHAR2 and the other schema_names parameter is type DBMS_UTILITY.UNCL_ARRAY. These parameters enable you to enter the list of tables and schemas in different ways and are mutually exclusive.

Note:
Starting with Oracle Database 11g Release 2 (11.2.0.2), the start_scn and start_time parameters are included in this procedure.

Syntax

```sql
DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name           IN VARCHAR2,
    table_names           IN DBMS_UTILITY.UNCL_ARRAY,
    schema_names          IN DBMS_UTILITY.UNCL_ARRAY,
    add                   IN BOOLEAN    DEFAULT TRUE,
    capture_user          IN VARCHAR2   DEFAULT NULL,
    connect_user          IN VARCHAR2   DEFAULT NULL,
    comment               IN VARCHAR2   DEFAULT NULL,
    inclusion_rule        IN BOOLEAN    DEFAULT TRUE,
    start_scn             IN NUMBER     DEFAULT NULL,
    start_time            IN TIMESTAMP  DEFAULT NULL,
    include_dml           IN BOOLEAN    DEFAULT TRUE,
    include_ddl           IN BOOLEAN    DEFAULT FALSE,
    source_database       IN VARCHAR2   DEFAULT NULL,
    source_container_name IN VARCHAR2   DEFAULT NULL);

DBMS_XSTREAM_ADM.ALTER_OUTBOUND(
    server_name           IN VARCHAR2,
    table_names           IN VARCHAR2   DEFAULT NULL,
    schema_names          IN VARCHAR2   DEFAULT NULL,
    add                   IN BOOLEAN    DEFAULT TRUE,
    capture_user          IN VARCHAR2   DEFAULT NULL,
    connect_user          IN VARCHAR2   DEFAULT NULL,
    comment               IN VARCHAR2   DEFAULT NULL,
)```
Parameters

Table 212-14  ALTER_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the outbound server being altered. Specify an existing outbound server. Do not specify an owner.</td>
</tr>
</tbody>
</table>
| table_names        | The tables that are either added to or removed from the XStream Out configuration. Whether the tables are added or removed depends on the setting for the add parameter. The tables can be specified in the following ways:  
  - Comma-delimited list of type VARCHAR2.  
  - A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table. Specify the first table in position 1. The last position must be NULL. Each table should be specified as [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default. |
| schema_names       | The schemas that are either added to or removed from the XStream Out configuration. Whether the schemas are added or removed depends on the setting for the add parameter. The schemas can be specified in the following ways:  
  - Comma-delimited list of type VARCHAR2.  
  - A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a schema. Specify the first schema in position 1. The last position must be NULL. Note: This procedure does not concatenate the schema_names parameter with the table_names parameter. To specify tables, enter fully qualified table names in the table_names parameter (schema_name.table_name). |

Note: This procedure does not concatenate the schema_names parameter with the table_names parameter. To specify tables, enter fully qualified table names in the table_names parameter (schema_name.table_name).

See Also: "Usage Notes" for more information about this parameter
Table 212-14  (Cont.) ALTER_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_user</td>
<td>The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. Specify a user to change the capture user. In this case, the user who invokes the ALTER_OUTBOUND procedure must be granted the DBA role. Only the SYS user can set the capture_user to SYS. If NULL, then the capture user is not changed. If you change the capture user, then this procedure grants the new capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user. Ensure that the capture user is granted the other required privileges. See &quot;CREATE_OUTBOUND Procedure&quot; for information about the privileges required by a capture user. The capture process is stopped and restarted automatically when you change the value of this parameter. <strong>Note:</strong> If the capture user for a capture process is dropped using DROP USER . . . CASCADE, then the capture process is also dropped automatically.</td>
</tr>
<tr>
<td>connect_user</td>
<td>The user who can attach to the specified outbound server to retrieve the change stream. The XStream client application must attach to the outbound server as the specified connect user. Specify a user to change the connect user. In this case, the user who invokes the ALTER_OUTBOUND procedure must be granted the DBA role. Only the SYS user can set the connect_user to SYS. If NULL, then the connect user is not changed. If you change the connect user, then this procedure grants the new connect user dequeue privileges on the queue used by the outbound server and configures the user as a secure queue user. Ensure that the connect user is granted the other required privileges. See &quot;CREATE_OUTBOUND Procedure&quot; for information about the privileges required by a connect user.</td>
</tr>
<tr>
<td>comment</td>
<td>An optional comment associated with the outbound server. If non-NULL, then the specified comment replaces the existing comment. If NULL, then the existing comment is not changed.</td>
</tr>
</tbody>
</table>


Table 212-14 (Cont.) ALTER_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inclusion_rule</td>
<td>If TRUE and the add parameter is set to TRUE, then the procedure adds rules for the tables specified in the table_names parameter and the schemas specified in the schema_names parameter to the positive rule sets in the XStream Out configuration. When rules for tables and schemas are in positive rule sets, the XStream Out configuration streams DML and DDL changes to the tables and schemas out to the client application. If TRUE and the add parameter is set to FALSE, then the procedure removes rules for the tables specified in the table_names parameter and the schemas specified in the schema_names parameter from the positive rule sets in the XStream Out configuration. If FALSE and the add parameter is set to TRUE, then the procedure adds rules for the tables specified in the table_names parameter and the schemas specified in the schema_names parameter to the negative rule sets in the XStream Out configuration. When rules for tables and schemas are in negative rule sets, the XStream Out configuration does not stream changes to the tables and schemas out to the client application. If FALSE and the add parameter is set to FALSE, then the procedure removes rules for the tables specified in the table_names parameter and the schemas specified in the schema_names parameter from the negative rule sets in the XStream Out configuration.</td>
</tr>
</tbody>
</table>
| start_scn      | A valid SCN for the database from which the capture process starts capturing changes. To be valid, the SCN value must be greater than or equal to the first SCN for the capture process. An error is returned if an invalid SCN is specified. If NULL and the start_time parameter is NULL, then the start SCN is not changed. If NULL and the start_time parameter is non-NULL, then the start SCN is changed to match the specified start time. The start_scn and start_time parameters are mutually exclusive.  

**Note:** If the capture process is enabled, then the ALTER_OUTBOUND procedure automatically stops and restarts the capture process when the start_scn parameter is non-NULL. If the capture process is disabled, then the ALTER_OUTBOUND procedure automatically starts the capture process when the start_scn parameter is non-NULL.
Table 212-14  (Cont.) ALTER_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_time</td>
<td>A valid time from which the capture process starts capturing changes. To be valid, the time must correspond to an SCN value that is greater than or equal to the first SCN for the capture process. If a valid time is specified, then the capture process captures changes from the specified time when it is restarted. An error is returned if an invalid time is specified. If NULL and the start_scn parameter is NULL, then the start time is not changed. If NULL and the start_scn parameter is non-NULL, then the start time is changed to match the specified start SCN. The start_scn and start_time parameters are mutually exclusive. Note: If the capture process is enabled, then the ALTER_OUTBOUND procedure automatically stops and restarts the capture process when the start_time parameter is non-NULL. If the capture process is disabled, then the ALTER_OUTBOUND procedure automatically starts the capture process when the start_time parameter is non-NULL.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a DML rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a DDL rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td>source_database</td>
<td>The global name of the container where the specified table_names and schema_names are located. If source_database is non-NULL, then a condition is added to the outbound server's rules to filter the LCRs based on the global name of the source database.</td>
</tr>
<tr>
<td>source_container_name</td>
<td>The short name of the source container. The container can be the root or a PDB. For example, CDB$ROOT or hrpdb. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. Note: This parameter only applies to a CDB.</td>
</tr>
</tbody>
</table>

Usage Notes

The following list describes the behavior of the outbound server for various combinations of the table_names and schema_names parameters:

- If both the table_names and schema_names parameters are NULL or empty, then no rules are changed for the XStream Out configuration. This procedure is overloaded. The table_names and schema_names parameters are defaulted to NULL. Do not specify NULL for both table_names and schema_names in the same call; otherwise, error PLS-00307 is returned.

- If both the table_names and schema_names parameters are specified, then the rules for the tables and schemas are added to or removed from the XStream Out configuration, depending on the setting of the add parameter.
If the `table_names` parameter is specified and the `schema_names` parameter is NULL or empty, then the rules for the tables are added to or removed from the XStream Out configuration, depending on the setting of the `add` parameter. The existing rules for schemas are not changed for the XStream Out configuration.

If the `table_names` parameter is NULL or empty and the `schema_names` parameter is specified, then the rules for the schemas are added to or removed from the XStream Out configuration, depending on the setting of the `add` parameter. The existing rules for tables are not changed for the XStream Out configuration.

For the procedure that uses the `DBMS_UTILITY.UNCL_ARRAY` type for the `table_names` and `schema_names` parameters, both parameters must be specified. To specify only tables, the `schema_names` parameter must be specified and empty. To specify only schemas, the `table_names` parameter must be specified and empty.

**Note:**
An empty array includes one NULL entry.

### 212.4.14 CREATE_INBOUND Procedure

This procedure creates an XStream inbound server and its queue.

**Note:**
A client application can create multiple sessions. Each session can attach to only one inbound server, and each inbound server can serve only one session at a time. However, different client application sessions can connect to different inbound servers. See *Oracle Call Interface Programmer’s Guide* and *Oracle Database XStream Java API Reference* for information about attaching to an inbound server.

**Syntax**

```sql
DBMS_XSTREAM_ADM.CREATE_INBOUND(
  server_name  IN VARCHAR2,
  queue_name   IN VARCHAR2,
  apply_user   IN VARCHAR2 DEFAULT NULL,
  comment      IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 212-15  CREATE_INBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the inbound server being created. A NULL specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing outbound server, inbound server, apply process, or messaging client. <strong>Note:</strong> The server_name setting cannot exceed 30 bytes, and it cannot be altered after the inbound server is created.</td>
</tr>
<tr>
<td>queue_name</td>
<td>The name of the local queue used by the inbound server, specified as [schema_name.]queue_name. If the specified queue exists, then it is used. If the specified queue does not exist, then the procedure creates it. For example, to specify a queue named xstream_queue in the xstrmadmin schema, enter xstrmadmin.xstream_queue for this parameter. If the schema is not specified, then the current user is the default. <strong>Note:</strong> An inbound server's queue is used only to store error transactions.</td>
</tr>
</tbody>
</table>
| apply_user  | The apply user. If NULL, then the current user is the default. The client application must attach to the inbound server as the apply user. The apply user is the user in whose security domain an inbound server evaluates whether LCRs satisfy its rule sets, applies DML and DDL changes directly to database objects, runs custom rule-based transformations configured for inbound server rules, and runs apply handlers configured for the inbound server. This user must have the necessary privileges to perform these actions. This procedure grants the apply user dequeue privileges on the queue used by the inbound server and configures the user as a secure queue user.
In addition to the privileges granted by this procedure, you must grant the following privileges to the apply user:
- The necessary privileges to perform DML and DDL changes on the apply objects
- EXECUTE privilege on the rule sets used by the inbound server
- EXECUTE privilege on all rule-based transformation functions used in the rule set
- EXECUTE privilege on all apply handler procedures
You can grant these privileges directly to the apply user, or you can grant them through roles.
In addition, the apply user must be granted EXECUTE privilege on all packages, including Oracle supplied packages, that are invoked in subprograms run by the inbound server. These privileges must be granted directly to the apply user. They cannot be granted through roles. **Note:** If the apply user for an inbound server is dropped using DROP USER . . . CASCADE, then the inbound server is also dropped automatically. |
| comment     | An optional comment associated with the inbound server. |

Usage Notes

By default, an inbound server does not use rules or rule sets. Therefore, an inbound server applies all of the LCRs sent to it by an XStream client application. However, to filter the LCRs
sent to an inbound server, you can add rules and rule sets to an inbound server using
the DBMS_XSTREAM_ADM and DBMS_RULE_ADM packages.

In a CDB, you can execute the CREATE_INBOUND procedure from either the root or a
PDB. The inbound server is restricted to receiving LCRs from one source database
and only applying the changes to its local container. If the inbound server is at the root
level, then the apply user must be a common user.

See Also:
Oracle Database XStream Guide

212.4.15 CREATE_OUTBOUND Procedure

This procedure creates an XStream outbound server, queue, and capture process to
enable client applications to stream out Oracle database changes.

This procedure is overloaded. One table_names parameter is type VARCHAR2 and the
other table_names parameter is type DBMS_UTILITY.UNCL_ARRAY. Also, one
schema_names parameter is type VARCHAR2 and the other schema_names parameter is
type DBMS_UTILITY.UNCL_ARRAY. These parameters enable you to enter the list of
tables and schemas in different ways and are mutually exclusive.

Note:

• A client application can create multiple sessions. Each session can
  attach to only one outbound server, and each outbound server can serve
  only one session at a time. However, different client application sessions
  can connect to different outbound servers. See
  "OCIXStreamOutAttach()" in the Oracle Call Interface Programmer's
  Guide and Oracle Database XStream Java API Reference for
  information about attaching to an outbound server.

• If the capture_name parameter is NULL, then this procedure automatically
  generates a name for the capture process that it creates.

• This procedure automatically generates a name for the queue that it
  creates.

• This procedure enables both the capture process and outbound server
  that it creates.

• Starting with Oracle Database 11g Release 2 (11.2.0.2), the
capture_name parameter is included in this procedure.

Syntax

DBMS_XSTREAM_ADM.CREATE_OUTBOUND(
  server_name            IN VARCHAR2,
  source_database        IN VARCHAR2  DEFAULT NULL,
  table_names            IN DBMS_UTILITY.UNCL_ARRAY,
  schema_names           IN DBMS_UTILITY.UNCL_ARRAY,
Parameters

Table 212-16 CREATE_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the outbound server being created. A NULL specification is not allowed. Do not specify an owner. The specified name must not match the name of an existing outbound server, inbound server, apply process, or messaging client. <strong>Note:</strong> The server_name setting cannot exceed 30 bytes, and it cannot be altered after the outbound server is created.</td>
</tr>
</tbody>
</table>
Table 212-16 (Cont.) CREATE_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>The global name of the source database. The source database is where the changes to be captured originated. If non-NULL, then a condition is added to the outbound server's rules to filter the LCRs based on the global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules. In a CDB, specify the global name of the container to which the rules pertain. The container can be the root or a PDB. For example, mycdb.example.com or hrpdb.example.com. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. In a non-CDB, if non-NULL and the specified name is different from the global name of the current database, then downstream capture is assumed. In this case, configure the transmission of redo data from the source database to the downstream database before running the CREATE_OUTBOUND procedure. See Oracle Database XStream Guide for instructions. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically.</td>
</tr>
</tbody>
</table>
| table_names | The tables for which DML and DDL changes are streamed out to the XStream client application. The tables can be specified in the following ways:  
  - Comma-delimited list of type VARCHAR2.  
  - A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table. Specify the first table in position 1. The last position must be NULL. Each table should be specified as [schema_name.]table_name. For example, hr.employees. If the schema is not specified, then the current user is the default. See Also: "Usage Notes" for more information about this parameter. |
| schema_names | The schemas for which DML and DDL changes are streamed out to the XStream client application. The schemas can be specified in the following ways:  
  - Comma-delimited list of type VARCHAR2.  
  - A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a schema. Specify the first schema in position 1. The last position must be NULL. Note: This procedure does not concatenate the schema_names parameter with the table_names parameter. To specify tables, enter fully qualified table names in the table_names parameter (schema_name.table_name). See Also: "Usage Notes" for more information about this parameter. |
Table 212-16  (Cont.) CREATE_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| capture_user| The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If NULL, then the current user is the default. This procedure grants the capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user. In addition, ensure that the capture user has the following privileges:  
  · EXECUTE privilege on the rule sets used by the capture process  
  · EXECUTE privilege on all rule-based transformation functions used in the positive rule set  
You can grant these privileges directly to the apply user, or you can grant them through roles. In addition, the capture user must be granted EXECUTE privilege on all packages, including Oracle supplied packages, that are invoked in rule-based transformations run by the capture process. These privileges must be granted directly to the capture user. They cannot be granted through roles. Only a user who is granted the DBA role can set a capture user. Only the SYS user can set the capture_user to SYS. A capture user does not require privileges on a database object to capture changes made to it. The capture process can pass these changes to a custom rule-based transformation function. Therefore, ensure that you consider security implications when you configure a capture process. |
| connect_user| The user who can attach to the specified outbound server to retrieve the change stream. The client application must attach to the outbound server as the specified connect user. If NULL, then the current user is the default. The connect user is the user in whose security domain an outbound server evaluates LCRs against its rule sets and runs custom rule-based transformations configured for outbound server rules. This user must have the necessary privileges to perform these actions. This procedure grants the connect user dequeue privileges on the queue used by the outbound server and configures the user as a secure queue user. In addition to the privileges granted by this procedure, grant the following privileges to the connect user:  
  · EXECUTE privilege on the rule sets used by the outbound server  
  · EXECUTE privilege on all rule-based transformation functions used in the rule set  
You can grant these privileges directly to the connect user, or you can grant them through roles. In addition, the connect user must be granted EXECUTE privilege on all packages, including Oracle supplied packages, that are invoked in subprograms run by the outbound server. These privileges must be granted directly to the apply user. They cannot be granted through roles. |
<p>| comment     | An optional comment associated with the outbound server.                                                                                                                                                      |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_name</td>
<td>The name of the capture process configured to capture changes for the outbound server. Do not specify an owner. The capture process must not exist. If the specified name matches the name of an existing capture process, then an error is raised. If the name does not match the name of an existing capture process, then the procedure creates a new capture process with the specified name. If NULL, then the system creates a new capture process with a system-generated name. Note: The capture process name cannot be altered after the capture process is created.</td>
</tr>
<tr>
<td>include_dml</td>
<td>If TRUE, then the procedure creates a DML rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</td>
</tr>
<tr>
<td>include_ddl</td>
<td>If TRUE, then the procedure creates a DDL rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</td>
</tr>
<tr>
<td>source_root_name</td>
<td>The global name of the root in the source CDB. For example, mycdb.example.com. If this parameter is NULL, then the global name of the root in the local CDB is used. If you are configuring downstream capture, then this parameter must be a non-NULL value, and it must specify the global name of the root in the remote source CDB. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. Note: This parameter only applies to a CDB.</td>
</tr>
<tr>
<td>source_container_name</td>
<td>The short name of the source container. The container can be the root or a PDB. For example, CDB$ROOT or hrpdb. See Oracle Database XStream Guide for more information about setting this parameter in a CDB. Note: This parameter only applies to a CDB.</td>
</tr>
<tr>
<td>lcrid_version</td>
<td>The LCRID version for captured LCRs, either 1 or 2. If 2, then the LCRs are compatible with a database with its compatibility level at 12.2.0 or higher. If 1, then the LCRs are compatible with a database with its compatibility level at 12.1.0 or lower. If NULL, the default, and the database compatibility level is 12.2.0 or higher, then the lcrid_version is set to 2 internally. If the database compatibility level is 12.1.0 or lower, then the lcrid_version is set to 1 internally.</td>
</tr>
</tbody>
</table>
Usage Notes

The following list describes the behavior of the outbound server for various combinations of the `table_names` and `schema_names` parameters:

- If both the `table_names` and `schema_names` parameters are `NULL` or empty, then the outbound server streams all DML and DDL changes to the client application.
  This procedure is overloaded. The `table_names` and `schema_names` parameters are defaulted to `NULL`. Do not specify `NULL` for both `table_names` and `schema_names` in the same call; otherwise, error `PLS-00307` is returned.
- If both the `table_names` and `schema_names` parameters are specified, then the outbound server streams DML and DDL changes for the specified tables and schemas.
- If the `table_names` parameter is specified and the `schema_names` parameter is `NULL` or empty, then the outbound server streams DML and DDL changes for the specified tables.
- If the `table_names` parameter is `NULL` or empty and the `schema_names` parameter is specified, then the outbound server streams DML and DDL changes for the specified schema.

For the procedure that uses the `DBMS_UTLILITY.UNCL_ARRAY` type for the `table_names` and `schema_names` parameters, both parameters must be specified. To specify only tables, the `schema_names` parameter must be specified and empty. To specify only schemas, the `table_names` parameter must be specified and empty.

Note:
An empty array includes one `NULL` entry.

212.4.16 DELETE_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which deletes a column from a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an XStream client. XStream clients include capture processes, propagations, and apply processes.
Note:

- The `DELETE_COLUMN` procedure supports the same data types supported by Oracle Streams capture processes.
- The `DELETE_COLUMN` procedure is useful when you want to delete a relatively small number of columns in a row LCR. To delete most of the columns in a row LCR and keep a relatively small number of columns, consider using the `KEEP_COLUMNS` procedure in this package.
- Declarative transformations can transform row LCRs only. Therefore, a DML rule must be specified when you run this procedure. If a DDL rule is specified, then the procedure raises an error.

See Also:

- `Oracle Database XStream Guide` for more information about declarative rule-based transformations and about the data types supported by capture processes
- `KEEP_COLUMNS Procedure`

Syntax

```sql
DBMS_XSTREAM_ADM.DELETE_COLUMN(
    rule_name IN VARCHAR2,
    table_name IN VARCHAR2,
    column_name IN VARCHAR2,
    value_type IN VARCHAR2 DEFAULT '*',
    step_number IN NUMBER DEFAULT 0,
    operation IN VARCHAR2 DEFAULT 'ADD');
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as <code>[schema_name.]rule_name</code>. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table from which the column is deleted in the row LCR, specified as <code>[schema_name.]object_name</code>. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column deleted from each row LCR that satisfies the rule.</td>
</tr>
</tbody>
</table>
Table 212-17  (Cont.) DELETE_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to delete the column from the new values in the row LCR.</td>
</tr>
<tr>
<td></td>
<td>Specify 'OLD' to delete the column from the old values in the row LCR.</td>
</tr>
<tr>
<td></td>
<td>Specify '*' to delete the column from both the old and new values in the</td>
</tr>
<tr>
<td></td>
<td>row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle Database XStream Guide for more information about</td>
</tr>
<tr>
<td></td>
<td>transformation ordering</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule.</td>
</tr>
<tr>
<td></td>
<td>Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the delete column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the DELETE_COLUMN procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>table_name</th>
<th>column_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all delete column transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all delete column transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all delete column transformations with the specified column_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all delete column transformations with the specified column_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all delete column transformations with the specified column_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all delete column transformations with the specified table_name and column_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all delete column transformations with the specified table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all delete column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>
212.4.17 DROP_INBOUND Procedure

This procedure removes an inbound server configuration.

This procedure always removes the specified inbound server. This procedure also removes the queue for the inbound server if all of the following conditions are met:

- One call to the CREATE_INBOUND procedure created the queue.
- The inbound server is the only subscriber to the queue.

See Also:
"CREATE_INBOUND Procedure"

Syntax

DBMS_XSTREAM_ADM.DROP_INBOUND(
server_name IN VARCHAR2);

Parameters

Table 212-18  DROP_INBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the inbound server being removed. Specify an existing inbound server. Do not specify an owner.</td>
</tr>
</tbody>
</table>

212.4.18 DROP_OUTBOUND Procedure

This procedure removes an outbound server configuration.

This procedure always drops the specified outbound server. This procedure also drops the queue used by the outbound server if both of the following conditions are met:

- The queue was created by the CREATE_OUTBOUND procedure in this package.
- The outbound server is the only subscriber to the queue.

If either one of the preceding conditions is not met, then the DROP_OUTBOUND procedure only drops the outbound server. It does not drop the queue.

This procedure also drops the capture process for the outbound server if both of the following conditions are met:

- The procedure can drop the outbound server's queue.
- The capture process was created by the CREATE_OUTBOUND procedure.

If the procedure can drop the queue but cannot manage the capture process, then it drops the queue without dropping the capture process.
See Also:

- "ADD_OUTBOUND Procedure"
- "CREATE_OUTBOUND Procedure"

Syntax

```sql
DBMS_XSTREAM_ADM.DROP_OUTBOUND(
    server_name IN VARCHAR2);
```

Parameters

**Table 212-19  DROP_OUTBOUND Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the outbound server being removed. Specify an existing outbound server. Do not specify an owner.</td>
</tr>
</tbody>
</table>

212.4.19 ENABLE_GG_XSTREAM_FOR_STREAMS Procedure

This procedure enables XStream optimizations and performance optimizations for Oracle Streams components.

This procedure is intended for users of Oracle Streams who want to enable XStream optimizations and optimizations. For example, you can enable the optimizations for an Oracle Streams replication configuration that uses capture processes and apply processes to replicate changes between Oracle databases.

These capabilities and optimizations are enabled automatically for XStream components, such as outbound servers, inbound servers, and capture processes that send changes to outbound servers. It is not necessary to run this procedure for XStream components.

When XStream optimizations are enabled, Oracle Streams components can stream ID key LCRs and sequence LCRs. The XStream performance optimizations improve efficiency in various areas, including:

- LCR processing
- Handling large transactions
- DML execution during apply
- Dependency computation and scheduling
- Capture process parallelism

Syntax

```sql
DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS(
    enable IN BOOLEAN  TRUE);
```
Parameters

Table 212-20  ENABLE_GG_XSTREAM_FOR_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>If TRUE, then enable XStream performance optimizations for Oracle Streams components. If FALSE, then disable XStream performance optimizations for Oracle Streams components.</td>
</tr>
</tbody>
</table>

Usage Notes

The following usage notes apply to this procedure:

- When you run this procedure, all capture processes and apply processes are restarted.

- After you run this procedure, the PURPOSE column in the following views displays XStream Streams:
  - ALL_APPLY
  - DBA_APPLY
  - ALL_CAPTURE
  - DBA_CAPTURE

- A license for the Oracle GoldenGate product is required to enable XStream performance optimizations for Oracle Streams components.

See Also:

- IS_GG_XSTREAM_FOR_STREAMS Function
- Oracle Database XStream Guide, Chapter 1, Prerequisites for XStream

212.4.20 GET_MESSAGE_TRACKING Function

The GET_MESSAGE_TRACKING Function returns the tracking label for the current session.

See Also:

- SET_MESSAGE_TRACKING Procedure

Syntax

```sql
DBMS_STREAMS_ADM.GET_MESSAGE_TRACKING
RETURN VARCHAR2;
```
212.4.21 GET_TAG Function

This function gets the binary tag for all redo entries generated by the current session.

See Also:

- "SET_TAG Procedure"
- Oracle Streams Replication Administrator's Guide for more information about tags

Syntax

```
DBMS_STREAMS_ADM.GET_TAG
RETURN RAW;
```

Examples

The following example illustrates how to display the current logical change record (LCR) tag as output:

```
SET SERVEROUTPUT ON
DECLARE
  raw_tag RAW(2000);
BEGIN
  raw_tag := DBMS_STREAMS_ADM.GET_TAG();
  DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/
```

You can also display the value by querying the DUAL view:

```
SELECT DBMS_STREAMS_ADM.GET_TAG FROM DUAL;
```

212.4.22 IS_GG_XSTREAM_FOR_STREAMS Function

This function returns TRUE if XStream performance optimizations are enabled for Oracle Streams components, or this function returns FALSE if XStream performance optimizations are disabled for Oracle Streams components.

See Also:

"ENABLE_GG_XSTREAM_FOR_STREAMS Procedure"

Syntax

```
DBMS_XSTREAM_ADM.IS_GG_XSTREAM_FOR_STREAMS
RETURN BOOLEAN;
```
212.4.23 KEEP_COLUMNS Procedure

This procedure either adds or removes a declarative rule-based transformation which keeps a list of columns in a row logical change record (LCR) that satisfies the specified rule. The transformation deletes columns that are not in the list from the row LCR.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an XStream client. XStream clients include capture processes, propagations, and apply processes.

This procedure is overloaded. The `column_list` parameter is type `VARCHAR2` and the `column_table` parameter is type `DBMS_UTILITY.LNAME_ARRAY`. These parameters enable you to enter the list of columns in different ways and are mutually exclusive.

**Note:**

- The `KEEP_COLUMNS` procedure supports the same data types supported by Oracle Streams capture processes.
- The `KEEP_COLUMNS` procedure is useful when you want to keep a relatively small number of columns in a row LCR. To keep most of the columns in a row LCR and delete a relatively small number of columns, consider using the `DELETE_COLUMN` procedure in this package.
- Declarative transformations can transform row LCRs only. Therefore, a DML rule must be specified when you run this procedure. If a DDL rule is specified, then the procedure raises an error.

**See Also:**

- `Oracle Database XStream Guide` for more information about declarative rule-based transformations and about the data types supported by Oracle Streams capture processes
- `DELETE_COLUMN Procedure`

**Syntax**

```sql
DBMS_XSTREAM_ADM.KEEP_COLUMNS(
    rule_name     IN  VARCHAR2,
    table_name    IN  VARCHAR2,
    column_list   IN  VARCHAR2,
    value_type    IN  VARCHAR2 DEFAULT '*',
    step_number   IN  NUMBER DEFAULT 0,
    operation     IN  VARCHAR2 DEFAULT 'ADD');

DBMS_XSTREAM_ADM.KEEP_COLUMNS(
    rule_name     IN  VARCHAR2,
    table_name    IN  VARCHAR2,
    column_table  IN  DBMS_UTILITY.LNAME_ARRAY,
    value_type    IN  VARCHAR2 DEFAULT '*');
```
Parameters

Table 212-21  KEEP_COLUMNS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as <code>[schema_name.]rule_name</code>. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table for which the columns are kept in the row LCR, specified as <code>[schema_name.]object_name</code>. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>column_list</td>
<td>The names of the columns kept for each row LCR that satisfies the rule. Specify a comma-delimited list of type VARCHAR2. The transformation removes columns that are not in the list from the row LCR. If this parameter is set to NULL, and the column_table parameter is also set to NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>column_table</td>
<td>The names of the columns kept for each row LCR that satisfies the rule. Specify a PL/SQL associative array of type DBMS_UTILITY.LNAME_ARRAY, where each element is the name of a column. The first schema should be in position 1. The last position must be NULL. The transformation removes columns that are not in the table from the row LCR. If this parameter is set to NULL, and the column_list parameter is also set to NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to keep the columns in the new values in the row LCR. Specify 'OLD' to keep the columns in the old values in the row LCR. Specify '*' to keep the columns in both the old and new values in the row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation. See Also: Oracle Database XStream Guide for more information about transformation ordering</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the keep columns declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_list, column_table, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the KEEP_COLUMNS procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>table_name</th>
<th>column_list/ column_table</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations for the specified rule.</td>
</tr>
</tbody>
</table>
### 212.4.24 MERGE_STREAMS Procedure

This procedure merges a stream that is flowing from one capture process with a stream that is flowing from another capture process.

Typically, this procedure is used to merge two streams that were split using the SPLIT_STREAMS procedure in this package. The SPLIT_STREAMS procedure clones components of the original stream when it splits the streams. Therefore, the information in this section uses the following terminology:

- The stream before it was split off has the original queue, original capture process, and original propagation.
- The stream that was split off by the SPLIT_STREAMS procedure has a cloned queue, cloned capture process, and cloned propagation.

This procedure is called by the MERGE_STREAMS_JOB procedure. The MERGE_STREAMS_JOB procedure determines whether the streams are within a user-specified merge threshold so that the streams can be merged safely. If the streams are not within the merge threshold, then the MERGE_STREAMS_JOB procedure does nothing. Typically, it is best to run the MERGE_STREAMS_JOB procedure instead of running the MERGE_STREAMS procedure directly.

However, you can choose to run the MERGE_STREAMS procedure directly when the following conditions are met:

<table>
<thead>
<tr>
<th>table_name</th>
<th>column_list/column_table</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified column_list/column_table and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations with the specified column_list/column_table for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations with the specified table_name and column_list/column_table for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all keep columns transformations with the specified table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all keep columns transformations with the specified table_name, column_list/column_table, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>
• The problem at the destination of the split stream has been corrected, and the destination queue can accept changes.

• The cloned capture process used by the split stream is started and is capturing changes.

• The apply process at the destination database is applying the changes captured by the cloned capture process.

• The CAPTURE_MESSAGE_CREATE_TIME in the GV$STREAMS_CAPTURE view of the cloned capture process has caught up to, or nearly caught up to, the CAPTURE_MESSAGE_CREATE_TIME of the original capture process. The cloned capture process might never completely catch up to the original capture process. Therefore, you can merge the split stream when the cloned capture process has nearly caught up to the original capture process.

The MERGE_STREAMS procedure performs the following actions:

1. Stops the cloned capture process.
2. Stops the original capture process.
3. Copies the cloned propagation back to the original propagation. The propagation has the same name as the original propagation after it is copied back.
4. Starts the original capture process from the lower SCN value of these two SCN values:
   - The acknowledged SCN of the cloned propagation.
   - The lowest acknowledged SCN of the other propagations that propagate changes captured by the original capture process.

When the original capture process is started, it might recapture changes that it already captured, or it might capture changes that were already captured by the cloned capture process. In either case, the relevant apply processes will discard any duplicate changes they receive.

5. Drops the cloned propagation.
6. Drops the cloned capture process.
7. Drops the cloned queue.

See Also:

• MERGE_STREAMS Procedure
• SPLIT_STREAMS Procedure

Syntax

```sql
DBMS_XSTREAM_ADM.MERGE_STREAMS(
  cloned_propagation_name  IN  VARCHAR2,
  propagation_name         IN  VARCHAR2  DEFAULT NULL,
  queue_name               IN  VARCHAR2  DEFAULT NULL,
  perform_actions          IN  BOOLEAN   DEFAULT TRUE,
  script_name              IN  VARCHAR2  DEFAULT NULL,
  script_directory_object  IN  VARCHAR2  DEFAULT NULL);
```
### Parameters

#### Table 212-22  MERGE_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloned_propagation_name</td>
<td>The name of the cloned propagation used by the stream that was split off from the original stream using the SPLIT_STREAMS procedure. The name of the cloned propagation also identifies the cloned queue and capture process used by the cloned propagation. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
</tbody>
</table>
| propagation_name        | The name of the propagation that is merged back to the original stream. If NULL, then the name of the original propagation in the original stream is used. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value to use a name that is different from the original propagation name or if you are merging two streams that were not split by the SPLIT_STREAMS procedure. If a non-NULL value is specified, then an error is raised under either of the following conditions:  
  - The queue specified in the queue_name parameter does not exist.  
  - The queue specified in the queue_name parameter exists but is not used by a capture process. |
| queue_name               | The name of the queue that is the source queue for the propagation that is merged back. If NULL, then the existing, original queue is the source queue for the propagation that is merged back. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value if you are merging two streams that were not split by the SPLIT_STREAMS procedure. Specify the name of the existing queue used by the capture process that will capture changes in the merged stream. |
| perform_actions          | If TRUE, then the procedure performs the necessary actions to merge the streams directly. If FALSE, then the procedure does not perform the necessary actions to merge the streams directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:  
  - script_name  
  - script_directory_object |
Table 212-22  (Cont.) MERGE_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to merge the streams. If a file with the specified script name exists in the specified directory for the script directory object parameter, then the procedure appends the statements to the existing file. If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to split the stream directly. If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to merge the streams directly and does not generate a script. If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.</td>
</tr>
<tr>
<td>script_directory_object</td>
<td>The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
</tbody>
</table>

Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.

Usage Notes

You can use the MERGE_STREAMS procedure to merge two streams that were not split using the SPLIT_STREAMS procedure. Merging streams in this way can save resources and improve performance when a single database is running two or more capture processes.

The DBA_STREAMS_SPLIT_MERGE view contains information about split and merge operations.

212.4.25 MERGE_STREAMS_JOB Procedure

This procedure determines whether the original capture process and the cloned capture process are within the specified merge threshold. If they are within the merge threshold, then this procedure runs the MERGE_STREAMS procedure to merge the two streams.

Typically, this procedure is used to merge two streams that were split using the SPLIT_STREAMS procedure in this package. The SPLIT_STREAMS procedure clones components of the original stream when it splits the streams. Therefore, the information in this section uses the following terminology:

- The stream before it was split off has the original queue, original capture process, and original propagation.
- The stream that was split off by the SPLIT_STREAMS procedure has a cloned queue, cloned capture process, and cloned propagation.

If the auto_merge_threshold parameter was set to a positive number in the SPLIT_STREAMS procedure that split the streams, then a merge job runs the MERGE_STREAMS_JOB procedure.
automatically according to its schedule. The schedule name is specified for the schedule_name parameter, and the merge job name is specified for the merge_job_name parameter when the MERGE_STREAMS_JOB procedure is run automatically. The merge job and its schedule were created by the SPLIT_STREAMS procedure.

If the auto_merge_threshold parameter was set to NULL or 0 (zero) in the SPLIT_STREAMS procedure that split the streams, then you can run the MERGE_STREAMS_JOB procedure manually. In this case, it is not run automatically.

See Also:
- MERGE_STREAMS Procedure
- SPLIT_STREAMS Procedure
- Oracle Streams Replication Administrator’s Guide for instructions on using the MERGE_STREAMS_JOB procedure

Syntax

DBMS_XSTREAM_ADM.MERGE_STREAMS_JOB(
    cloned_propagation_name IN VARCHAR2,
    propagation_name        IN VARCHAR2 DEFAULT NULL,
    queue_name              IN VARCHAR2 DEFAULT NULL,
    merge_threshold         IN NUMBER,
    schedule_name           IN VARCHAR2 DEFAULT NULL,
    merge_job_name          IN VARCHAR2 DEFAULT NULL);

Parameters

Table 212-23 MERGE_STREAMS_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloned_propagation_name</td>
<td>The name of the cloned propagation used by the stream that was split off from the original stream using the SPLIT_STREAMS procedure. The name of the cloned propagation also identifies the cloned queue and capture process used by the cloned propagation. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
</tbody>
</table>
Table 212-23 (Cont.) MERGE_STREAMS_JOB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| propagation_name    | The name of the propagation that is merged back to the original stream. If NULL, then the name of the original propagation in the original stream is used. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value to use a name that is different from the original propagation name or if you are merging two streams that were not split by the SPLIT_STREAMS procedure. If a non-NULL value is specified, then an error is raised under either of the following conditions:  
  - The queue specified in the queue_name parameter does not exist.  
  - The queue specified in the queue_name parameter exists but is not used by a capture process. |
| queue_name          | The name of the queue that is the source queue for the propagation that is merged back. If NULL, then the existing, original queue is the source queue for the propagation that is merged back. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure. Specify a non-NULL value if you are merging two streams that were not split by the SPLIT_STREAMS procedure. Specify the name of the existing queue used by the capture process that will capture changes in the merged stream. |
| merge_threshold     | The merge threshold in seconds. The value of the CAPTURE_MESSAGE_CREATE_TIME column for each capture process in the GV$STREAMS_CAPTURE dynamic performance view determines whether the streams are merged. Specifically, if the difference, in seconds, between the CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process and the original capture process is less than or equal to the value specified for this parameter, then this procedure runs the MERGE_STREAMS procedure to merge the streams. If the difference is greater than the value specified by this parameter, then this procedure does nothing. |
| schedule_name       | The name of the schedule for the merge job. If NULL, then no schedule name is specified. Typically, you set this parameter to NULL when the auto_merge_threshold parameter was set to NULL or 0 (zero) in the SPLIT_STREAMS procedure that split the streams. Specify NULL if you run this procedure manually. |
| merge_job_name      | The name of the job that merges the streams. If NULL, then no merge job name is specified. Typically, you set this parameter to NULL when the auto_merge_threshold parameter was set to NULL or 0 (zero) in the SPLIT_STREAMS procedure that split the streams. Specify NULL if you run this procedure manually. |

Chapter 212  
Summary of DBMS_XSTREAM_ADM Subprograms
Usage Notes

You can use the `MERGE_STREAMS_JOB` procedure to merge two streams that were not split using the `SPLIT_STREAMS` procedure. Merging streams in this way can save resources and improve performance when a single database is running two or more capture processes.

After the `MERGE_STREAMS_JOB` procedure completes, you can query the `DBA_CAPTURE` and `DBA_PROPAGATION` views to determine whether the streams were merged. If the streams were merged, then the cloned capture process and cloned propagation do not appear in these views.

If the streams were merged and the `schedule_name` and `merge_job_name` parameters were non-NULL, then the specified schedule and merge job are deleted automatically.

The `DBA_STREAMS_SPLIT_MERGE` view contains information about split and merge operations.

212.4.26 PURGE_SOURCE_CATALOG Procedure

This procedure removes all Oracle Streams data dictionary information at the local database for the specified object.

You can use this procedure to remove Oracle Streams metadata that is not needed currently and will not be needed in the future.

Syntax

```sql
DBMS_XSTREAM_ADM.PURGE_SOURCE_CATALOG(
    source_database     IN  VARCHAR2,
    source_object_name  IN  VARCHAR2,
    source_object_type  IN  VARCHAR2,
    source_root_name    IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database</td>
<td>In a non-CDB, specify the global name of the source database containing the database object. In a CDB, specify the global name of the container containing the database object. The container can be the root or a PDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</td>
</tr>
<tr>
<td>source_object_name</td>
<td>The name of the object specified as <code>[schema_name.]object_name</code>. For example, <code>hr.employees</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>source_object_type</td>
<td>Type of the object. Currently, <code>TABLE</code> is the only possible object type.</td>
</tr>
</tbody>
</table>
Table 212-24  (Cont.) PURGE_SOURCE_CATALOG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_root_name</td>
<td>The global name of the source root containing the object in a CDB. The source root is where the changes being captured originated in a CDB. If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. If the source_root_name parameter is NULL, then the global name of the local root is the default. <strong>Note:</strong> This parameter only applies to a CDB.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The global name of the source database containing the object must be specified for the source_database parameter. If the current database is not the source database for the object, then the procedure removes data dictionary information about the object from the current database, not the source database.

For example, suppose changes to the hr.employees table at the dbs1.net source database are being applied to the hr.employees table at the dbs2.net destination database. Also, suppose hr.employees at dbs2.net is not a source at all. In this case, specifying dbs2.net as the source_database for this table results in an error. However, specifying dbs1.net as the source_database for this table while running the PURGE_SOURCE_CATALOG procedure at the dbs2.net database removes data dictionary information about the table at dbs2.net.

Do not run this procedure at a database if either of the following conditions is true:

- Logical change records (LCRs) captured by the capture process for the object are or might be applied locally without reinstantiating the object.
- LCRs captured by the capture process for the object are or might be forwarded by the database without reinstantiating the object.

**Note:** These conditions do not apply to LCRs that were not created by the capture process. That is, these conditions do not apply to user-created LCRs.

### 212.4.27 RECOVER_OPERATION Procedure

This procedure provides options for split and merge operations that stopped because they encountered an errors.

This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation. Split and merge operations might be run in an XStream Out environment in which multiple outbound servers use the same capture process.
This procedure only can perform these actions for split and merge operations using the split_threshold and merge_threshold capture process parameters set to non-NULL values to enable automatic split and merge.

Information about the operation is stored in the following data dictionary views when the operation is in process:

- DBA_RECOVERABLE_SCRIPT
- DBA_RECOVERABLE_SCRIPT_PARAMS
- DBA_RECOVERABLE_SCRIPT_BLOCKS
- DBA_RECOVERABLE_SCRIPT_ERRORS

The data dictionary views are populated at the database that contains the capture process.

When the operation completes successfully, metadata about the operation is moved from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view. The other views, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS, retain information about the operation until it is purged automatically after 30 days.

When one of these operations encounters an error and stops, metadata about the operation remains in these views. In this case, you can either roll forward, roll back, or purge the metadata about the operation using the RECOVER_OPERATION procedure. If you choose to roll forward the operation, then correct conditions that caused the errors reported in DBA_RECOVERABLE_SCRIPT_ERRORS before proceeding.

Run the RECOVER_OPERATION procedure at the database that contains the capture process.

**Note:**

To run the RECOVER_OPERATION procedure, both databases must be Oracle Database 10g Release 2 or later databases.

**See Also:**

- "SPLIT_STREAMS Procedure"
- "MERGE_STREAMS Procedure"
- "MERGE_STREAMS_JOB Procedure"
- Oracle Streams Replication Administrator's Guide

**Syntax**

```sql
DBMS_XSTREAM_ADM.RECOVER_OPERATION(
    script_id       IN  RAW,
    operation_mode IN  VARCHAR2 DEFAULT 'FORWARD');
```
Parameters

Table 212-25  RECOVER_OPERATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_id</td>
<td>The operation id of the operation that is being rolled forward, rolled back, or purged. Query the SCRIPT_ID column of the DBA_RECOVERABLE_SCRIPT data dictionary view to determine the operation id.</td>
</tr>
<tr>
<td>operation_mode</td>
<td>If FORWARD, then the procedure rolls forward the operation. Specify FORWARD to try to complete the operation. If ROLLBACK, then the procedure rolls back all of the actions performed in the operation. If the rollback is successful, then this option also moves the metadata about the operation from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view. The other views retain information about the operation for 30 days. If PURGE, then the procedure moves the metadata about the operation from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view without rolling the operation back. The other views retain information about the operation for 30 days.</td>
</tr>
</tbody>
</table>

212.4.28 REMOVE_QUEUE Procedure

This procedure removes the specified ANYDATA queue.

Specifically, this procedure performs the following actions:

1. Waits until all current enqueue and dequeue transactions commit.
2. Stops the queue, which means that no further enqueues into the queue or dequeues from the queue are allowed.
3. Drops the queue.
4. If the drop_unused_queue_table parameter is set to TRUE, then drops the queue table if it is empty and no other queues are using it.
5. If the cascade parameter is set to TRUE, then drops all of the XStream clients that are using the queue.

Note:

The specified queue must be a ANYDATA queue.

Syntax

```sql
DBMS_XSTREAM_ADM.REMOVE_QUEUE(
    queue_name               IN  VARCHAR2,
    cascade                  IN  BOOLEAN  DEFAULT FALSE,
    drop_unused_queue_table  IN  BOOLEAN  DEFAULT TRUE);
```
Parameters

### Table 212-26 REMOVE_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>The name of the queue to remove, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>cascade</td>
<td>If TRUE, then the procedure drops any XStream clients that use the queue. If FALSE, then the procedure raises an error if there are any XStream clients that use the queue. Before you run this procedure with the cascade parameter set to FALSE, make sure no XStream clients are using the queue currently.</td>
</tr>
<tr>
<td>drop_unused_queue_table</td>
<td>If TRUE and the queue table for the queue is empty, then the procedure drops the queue table. The queue table is not dropped if it contains any messages or if it is used by another queue. If FALSE, then the procedure does not drop the queue table.</td>
</tr>
</tbody>
</table>

### 212.4.29 REMOVE_RULE Procedure

This procedure removes the specified rule or all rules from the rule set associated with the specified capture process, apply process, or propagation.

**Note:**

If a rule was automatically created by the system, and you want to drop the rule, then you should use this procedure to remove the rule instead of the DBMS_RULE_ADM.DROP_RULE procedure. If you use the DBMS_RULE_ADM.DROP_RULE procedure, then some metadata about the rule might remain.

**Syntax**

```sql
DBMS_XSTREAM_ADM.REMOVE_RULE(
  rule_name         IN  VARCHAR2,
  streams_type      IN  VARCHAR2,
  streams_name      IN  VARCHAR2,
  drop_unused_rule  IN  BOOLEAN DEFAULT TRUE,
  inclusion_rule    IN  BOOLEAN DEFAULT TRUE);
```
Parameters

Table 212-27   REMOVE_RULE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule to remove, specified as [schema_name.]rule_name. If NULL, then the procedure removes all rules from the specified capture process, apply process, or propagation rule set. For example, to specify a rule in the hr schema named prop_rule1, enter hr.prop_rule1. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>streams_type</td>
<td>The type of XStream client:</td>
</tr>
<tr>
<td></td>
<td>· Specify capture for a capture process.</td>
</tr>
<tr>
<td></td>
<td>· Specify propagation for a propagation.</td>
</tr>
<tr>
<td></td>
<td>· Specify apply for an apply process.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the XStream client, which can be a capture process, propagation, or apply process. Do not specify an owner. If the specified XStream client does not exist, but there is metadata in the data dictionary that associates the rule with this client, then the procedure removes the metadata. If the specified XStream client does not exist, and there is no metadata in the data dictionary that associates the rule with this client, then the procedure raises an error.</td>
</tr>
<tr>
<td>drop_unused_rule</td>
<td>If TRUE and the rule is not in any rule set, then the procedure drops the rule from the database. If TRUE and the rule exists in any rule set, then the procedure does not drop the rule from the database. If FALSE, then the procedure does not drop the rule from the database.</td>
</tr>
<tr>
<td>inclusion_rule</td>
<td>If inclusion_rule is TRUE, then the procedure removes the rule from the positive rule set for the XStream client. If inclusion_rule is FALSE, then the procedure removes the rule from the negative rule set for the XStream client.</td>
</tr>
</tbody>
</table>

212.4.30 REMOVE_SUBSET_OUTBOUND_RULES Procedure

This procedure removes subset rules from an outbound server configuration.

The names of the specified insert, update, and delete rules must match those generated by the ADD_SUBSET_OUTBOUND_RULES procedure. To view the rule names for subset rules, run the following query:

```sql
SELECT RULE_OWNER, SUBSETTING_OPERATION, RULE_NAME
FROM ALL_XSTREAM_RULES
WHERE SUBSETTING_OPERATION IS NOT NULL;
```
Note:

- This procedure removes the declarative rule-based transformation associated with each rule it removes.
- This procedure does not remove rules from the outbound server's capture process.

See Also:

"ADD_SUBSET_OUTBOUND_RULES Procedure"

Syntax

```sql
DBMS_XSTREAM_ADM.REMOVE_SUBSET_OUTBOUND_RULES(
    server_name IN VARCHAR2,
    insert_rule_name IN VARCHAR2,
    update_rule_name IN VARCHAR2,
    delete_rule_name IN VARCHAR2);
```

212.4.31 REMOVE_XSTREAM_CONFIGURATION Procedure

This procedure removes the XStream configuration at the local database.

Syntax

```sql
DBMS_XSTREAM_ADM.REMOVE_XSTREAM_CONFIGURATION(
    container IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 212-28  REMOVE_XSTREAM_CONFIGURATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>If CURRENT, then the XStream configuration is removed from the current container. CURRENT can be specified while connected to the root or to a PDB in a CDB. If ALL, then the XStream configuration is removed from all of the containers in the CDB. To specify ALL, the procedure must be invoked in the root. If a container name, then the XStream configuration is removed from the specified container. To specify root, use CDB$ROOT while connected to the root. To specify a PDB, the procedure must be invoked in the root. Note: This parameter only applies to a CDB.</td>
</tr>
</tbody>
</table>

Usage Notes

Specifically, this procedure performs the following actions at the local database:

- Drops all capture processes
• If any tables have been prepared for instantiation, then aborts preparation for instantiation for the table using the ABORT_TABLE_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package.

• If any schemas have been prepared for instantiation, then aborts preparation for instantiation for the schema using the ABORT_SCHEMA_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package.

• If the database has been prepared for instantiation, then aborts preparation for instantiation for the database using the ABORT_GLOBAL_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package.

• Drops propagations that were created using either the DBMS_XSTREAM_ADM package or the DBMS_PROPAGATION_ADM package. Before a propagation is dropped, its propagation job is disabled. Does not drop propagations that were created using the DBMS_AQADM package.

• Disables all propagation jobs used by propagations.

• Drops all apply processes. If there are apply errors in the error queue for an apply process, then this procedure deletes these apply errors before it drops the apply process.

• Removes specifications for DDL handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers.

• Removes specifications for message handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers.

• Removes specifications for precommit handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers.

• Removes the instantiation SCN and ignore SCN for each apply object and schema and for the entire database.

• Removes messaging clients.

• Unsets message notification specifications that were set using the SET_MESSAGE_NOTIFICATION procedure in the DBMS_XSTREAM_ADM package.

• Removes specifications for procedure DML handlers and error handlers, but does not delete the PL/SQL procedures used by these handlers.

• Removes update conflict handlers.

• Removes specifications for substitute key columns for apply tables.

• Drops rule sets and rules that were created using the DBMS_XSTREAM_ADM package.

• Drops unused rule sets that were used by capture processes, propagations, apply processes, and messaging clients, and removes the rules in these rule sets. These rules and rule sets are removed regardless of whether they were created using the DBMS_XSTREAM_ADM package or the DBMS_RULE_ADM package.

This procedure stops capture processes and apply processes before it drops them.

This procedure does not drop rule sets or rules if they meet both of the following conditions:

• The rule sets or rules were created using the DBMS_RULE_ADM package.

• The rule sets or rules were not used by a capture process, propagation, apply process, or messaging client.
Note:

Running this procedure is dangerous. You should run this procedure only if you are sure you want to remove the entire XStream configuration at a database. If an Oracle Streams configuration exists at the database, then this procedure also removes the entire Oracle Streams configuration.

Note:

- Running this procedure repeatedly does not cause errors. If the procedure fails to complete, then you can run it again.
- This procedure commits multiple times.

212.4.32 RENAME_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a column in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to TRUE, the rule must be in the positive rule set of an XStream client. XStream clients include capture processes, propagations, and apply processes.

Note:

- The RENAME_COLUMN procedure supports the same data types supported by Oracle Streams capture processes.
- Declarative transformations can transform row LCRs only. Therefore, a DML rule must be specified when you run this procedure. If a DDL rule is specified, then the procedure raises an error.

See Also:

Oracle Database XStream Guide for more information about declarative rule-based transformations and about the data types supported by Oracle Streams capture processes.

Syntax

```
DBMS_XSTREAM_ADM.RENAME_COLUMN(
    rule_name IN VARCHAR2,
    table_name IN VARCHAR2,
    from_column_name IN VARCHAR2,
    to_column_name IN VARCHAR2,
);```
value_type        IN  VARCHAR2  DEFAULT '*',
step_number       IN  NUMBER    DEFAULT 0,
operation         IN  VARCHAR2  DEFAULT 'ADD');

Parameters

Table 212-29  RENAME_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table in which the column is renamed in the row LCR, specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>from_column_name</td>
<td>The name of the column to be renamed in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>to_column_name</td>
<td>The new name of the column in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>value_type</td>
<td>Specify 'NEW' to rename the column in the new values in the row LCR. Specify 'OLD' to rename the column in the old values in the row LCR. Specify '*' to rename the column in both the old and new values in the row LCR.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation. See Also: Oracle Database XStream Guide for more information about transformation ordering</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename column declarative rule-based transformations for the specified rule are removed that match the specified table_name, column_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_COLUMN procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>table_name</th>
<th>from_column_name</th>
<th>to_column_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename column transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>table_name</td>
<td>from_column_name</td>
<td>to_column_name</td>
<td>step_number</td>
<td>Result</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename column transformations with the specified to_column_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename column transformations with the specified table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified column_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name and column_name for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename column transformations with the specified table_name, column_name, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>
212.4.33 RENAME_SCHEMA Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a schema in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an XStream client. XStream clients include capture processes, propagations, and apply processes.

**Note:**

Declarative transformations can transform row LCRs only. Therefore, a DML rule must be specified when you run this procedure. If a DDL rule is specified, then the procedure raises an error.

**See Also:**

*Oracle Database XStream Guide* for more information about declarative rule-based transformations

**Syntax**

```sql
DBMS_XSTREAM_ADM.RENAME_SCHEMA(
    rule_name         IN  VARCHAR2,
    from_schema_name  IN  VARCHAR2,
    to_schema_name    IN  VARCHAR2,
    step_number       IN  NUMBER    DEFAULT 0,
    operation         IN  VARCHAR2  DEFAULT 'ADD');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as <code>schema_name.[rule_name]</code>. If NULL, then the procedure raises an error. For example, to specify a rule in the <code>hr</code> schema named <code>employees12</code>, enter <code>hr.employees12</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>from_schema_name</td>
<td>The name of the schema to be renamed in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>to_schema_name</td>
<td>The new name of the schema in each row LCR that satisfies the rule.</td>
</tr>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation. See Also: <em>Oracle Database XStream Guide</em> for more information about transformation ordering.</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>
Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename schema declarative rule-based transformations for the specified rule are removed that match the specified from_schema_name, to_schema_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_SCHEMA procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>from_schema_name</th>
<th>to_schema_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename schema transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename schema transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename schema transformations with the specified to_schema_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename schema transformations with the specified from_schema_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename schema transformations with the specified to_schema_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename schema transformations with the specified from_schema_name and to_schema_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename schema transformations with the specified from_schema_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename schema transformations with the specified from_schema_name, to_schema_name, and step_number for the specified rule.</td>
</tr>
</tbody>
</table>
212.4.34 RENAME_TABLE Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a table in a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to TRUE, the rule must be in the positive rule set of an XStream client. XStream clients include capture processes, propagations, and apply processes.

**Note:**
Declarative transformations can transform row LCRs only. Therefore, a DML rule must be specified when you run this procedure. If a DDL rule is specified, then the procedure raises an error.

**See Also:**
Oracle Database XStream Guide for more information about declarative rule-based transformations

**Syntax**

```sql
DBMS_XSTREAM_ADM.RENAME_TABLE(
    rule_name        IN  VARCHAR2,
    from_table_name  IN  VARCHAR2,
    to_table_name    IN  VARCHAR2,
    step_number      IN  NUMBER    DEFAULT 0,
    operation        IN  VARCHAR2  DEFAULT 'ADD');
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The name of the rule, specified as [schema_name.]rule_name. If NULL, then the procedure raises an error. For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>from_table_name</td>
<td>The name of the table to be renamed in each row LCR that satisfies the rule, specified as [schema_name.]object_name. For example, hr.employees. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td>to_table_name</td>
<td>The new name of the table in each row LCR that satisfies the rule, specified as [schema_name.]object_name. For example, humres.staff. The transformation can rename the table only, the schema only, or the table and the schema. If the schema is not specified, then the current user is the default.</td>
</tr>
</tbody>
</table>
### Table 212-31  (Cont.) RENAME_TABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>step_number</td>
<td>The order of execution of the transformation.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> Oracle Database XStream Guide for more information about</td>
</tr>
<tr>
<td></td>
<td>transformation ordering</td>
</tr>
<tr>
<td>operation</td>
<td>Specify 'ADD' to add the transformation to the rule.</td>
</tr>
<tr>
<td></td>
<td>Specify 'REMOVE' to remove the transformation from the rule.</td>
</tr>
</tbody>
</table>

### Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename table declarative rule-based transformations for the specified rule are removed that match the specified from_table_name, to_table_name, and step_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME_TABLE procedure when one or more of these parameters is NULL:

<table>
<thead>
<tr>
<th>from_table_name</th>
<th>to_table_name</th>
<th>step_number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename table transformations with the specified step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>non-NULL</td>
<td>Remove all rename table transformations with the specified to_table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>non-NULL</td>
<td>Remove all rename table transformations with the specified from_table_name and step_number for the specified rule.</td>
</tr>
<tr>
<td>NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations with the specified to_table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>non-NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations with the specified from_table_name and to_table_name for the specified rule.</td>
</tr>
<tr>
<td>non-NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Remove all rename table transformations with the specified from_table_name for the specified rule.</td>
</tr>
</tbody>
</table>
212.4.35 SET_MESSAGE_TRACKING Procedure

This procedure sets the tracking label for logical change records (LCRs) produced by the current session.

This procedure affects only the current session. Any LCRs produced by the current session are tracked, including captured LCRs and persistent LCRs.

**Note:**

The tracking label set by this procedure does not track non-LCR messages.

**See Also:**

KAWGET_MESSAGE_TRACKING

**Syntax**

```sql
DBMS_XSTREAM_ADM.SET_MESSAGE_TRACKING(
    tracking_label IN VARCHAR2  DEFAULT 'Streams_tracking',
    actions        IN NUMBER    DEFAULT DBMS_XSTREAM_ADM.ACTION_MEMORY);
```

**Parameters**

**Table 212-32  SET_MESSAGE_TRACKING Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracking_label</td>
<td>The label used to track the LCRs produced by the session. Set this parameter to NULL to stop message tracking in the current session. The size limit for a label is 4,000 bytes.</td>
</tr>
<tr>
<td>actions</td>
<td>When DBMS_XSTREAM_ADM.ACTION_MEMORY is specified, the LCRs are tracked in memory, and the V$STREAMS_MESSAGE_TRACKING dynamic performance view is populated with information about the LCRs. Currently, DBMS_XSTREAM_ADM.ACTION_MEMORY is the only valid setting for this parameter. The value specified for this parameter is an enumerated constant. Enumerated constants must be prefixed with the package name.</td>
</tr>
</tbody>
</table>
212.4.36 SET_PARAMETER Procedure

This procedure sets a parameter for an outbound server, an inbound server, or an outbound server's capture process.

Syntax

```
DBMS_XSTREAM_ADM.SET_PARAMETER(
    streams_name    IN  VARCHAR2,
    streams_type    IN  VARCHAR2,
    parameter       IN  VARCHAR2,
    value           IN  VARCHAR2  DEFAULT NULL,
    no_wait         IN  BOOLEAN   DEFAULT FALSE,
    source_database IN  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 212-33  SET_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streams_type</td>
<td>The type of XStream client:</td>
</tr>
<tr>
<td></td>
<td>• Specify capture for a capture process.</td>
</tr>
<tr>
<td></td>
<td>• Specify apply for an outbound server or inbound server.</td>
</tr>
<tr>
<td>streams_name</td>
<td>The name of the capture process, outbound server, or inbound server. Do not specify an owner.</td>
</tr>
<tr>
<td>parameter</td>
<td>The name of the parameter you are setting.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Capture Process Parameters&quot; for information about capture process parameters.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Apply Component Parameters&quot; for information about outbound server and inbound server parameters.</td>
</tr>
<tr>
<td>value</td>
<td>The value to which the parameter is set.</td>
</tr>
<tr>
<td></td>
<td>• NULL, then the parameter is set to its default value.</td>
</tr>
<tr>
<td>no_wait</td>
<td>• TRUE, then the parameter is set immediately.</td>
</tr>
<tr>
<td></td>
<td>• FALSE, then the parameter is set after synchronizing with the running capture process, inbound server, or outbound server.</td>
</tr>
<tr>
<td></td>
<td>When you modify multiple parameters for the same process consecutively, setting this parameter to TRUE speeds up each call. However, if the process is currently running, you must set this parameter to FALSE in the last to the procedure to ensure that the process uses the modified parameter values. If the no_wait parameter is set to TRUE for the last call to the procedure, the running process might not detect the parameter changes.</td>
</tr>
</tbody>
</table>
Table 212-33  (Cont.) SET_PARAMETER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| source_database | • If CURRENT, then the parameter is set only in the container where the procedure is invoked. CURRENT can be specified while connected to the root or to a PDB.  
  • If ALL, then the parameter is set in all containers in the CDB and all PDBs created after the procedure is invoked. To specify ALL, the procedure must be invoked in the root.  
  • If a container name, then the parameter is set in the specified container. To specify root, use CDB$ROOT while connected to the root. To specify a PDB, the procedure must be invoked in the root.  
  Note: This parameter only applies to CDBs. Also, a non-null value can be specified only for the following parameters:  
    • include_objects capture parameter  
    • excludetag capture or apply parameter  
    • excludetrans capture or apply parameter  
    • excludeuser capture or apply parameter  
    • excludeuserid capture or apply parameter  
    • getreplicates capture or apply parameter  
    • getapplops capture or apply parameter |

212.4.37 SET_TAG Procedure

This procedure sets the binary tag for all redo entries subsequently generated by the current session.

Each redo entry generated by DML or DDL statements in the current session will have this tag. This procedure affects only the current session.

See Also:

Oracle Streams Replication Administrator’s Guide for more information about tags

Syntax

```sql
DBMS_XSTREAM_ADM.SET_TAG(
  tag IN RAW DEFAULT NULL);
```
Parameters

Table 212-34  SET_TAG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>The binary tag for all subsequent redo entries generated by the current session. A raw value is a sequence of bytes, and a byte is a sequence of bits. By default, the tag for a session is NULL. The size limit for a tag value is 2000 bytes.</td>
</tr>
</tbody>
</table>

Usage Notes

To set the tag to the hexadecimal value of '17' in the current session, run the following procedure:

```sql
EXEC DBMS_XSTREAM_ADM.SET_TAG(tag => HEXTORAW('17'));
```

The following are considerations for the SET_TAG procedure:

- This procedure is not transactional. That is, the effects of SET_TAG cannot be rolled back.
- If the SET_TAG procedure is run to set a non-NULL session tag before a data dictionary build has been performed on the database, then the redo entries for a transaction that started before the dictionary build might not include the specified tag value for the session. Therefore, perform a data dictionary build before using the SET_TAG procedure in a session. A data dictionary build happens when the DBMS_CAPTURE_ADM.BUILD procedure is run. The BUILD procedure can be run automatically when a capture process is created.

See Also:
BUILD Procedure

212.4.38 SET_UP_QUEUE Procedure

This procedure creates a queue table and a ANYDATA queue.

Syntax

```sql
DBMS_XSTREAM_ADM.SET_UP_QUEUE(
    queue_table  IN  VARCHAR2  DEFAULT 'streams_queue_table',
    storage_clause IN  VARCHAR2  DEFAULT NULL,
    queue_name    IN  VARCHAR2  DEFAULT 'streams_queue',
    queue_user    IN  VARCHAR2  DEFAULT NULL,
    comment       IN  VARCHAR2  DEFAULT NULL);
```
Parameters

Table 212-35   SET_UP_QUEUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>The name of the queue table specified as [schema_name.]queue_table_name. For example, strmadmin.streams_queue_table. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>If the queue table owner is not specified, then the procedure specifies the user who runs this procedure automatically as the queue table owner.</td>
</tr>
<tr>
<td></td>
<td>Queue table names can be a maximum of 24 bytes.</td>
</tr>
<tr>
<td>storage_clause</td>
<td>The storage clause for queue table</td>
</tr>
<tr>
<td></td>
<td>The storage parameter is included in the CREATE TABLE statement when the queue table is created. You can specify any valid table storage clause.</td>
</tr>
<tr>
<td></td>
<td>If a tablespace is not specified here, then the procedure creates the queue table and all its related objects in the default user tablespace of the user who runs this procedure. If a tablespace is specified here, then the procedure creates the queue table and all its related objects in the tablespace specified in the storage clause.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure uses the storage characteristics of the tablespace in which the queue table is created.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> Oracle Database SQL Language Reference for more information about storage clauses.</td>
</tr>
<tr>
<td>queue_name</td>
<td>The name of the queue that will function as the ANYDATA queue, specified as [schema_name.]queue_name. For example, strmadmin.streams_queue.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified, then the procedure uses the queue table owner. The owner of the queue table must also be the owner of the queue. The queue owner automatically has privileges to perform all queue operations on the queue.</td>
</tr>
<tr>
<td></td>
<td>If the schema is not specified for this parameter, and the queue table owner is not specified in queue_table, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>Queue names can be a maximum of 24 bytes.</td>
</tr>
<tr>
<td>queue_user</td>
<td>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</td>
</tr>
<tr>
<td>comment</td>
<td>The comment for the queue</td>
</tr>
</tbody>
</table>

UsageNotes

Set up includes the following actions:

- If the specified queue table does not exist, then this procedure runs the CREATE_QUEUE_TABLE procedure in the DBMS_AQADM package to create the queue table.
with the specified storage clause. If this procedure creates the queue table, then it creates a multiple consumer ANYDATA queue that is both a secure queue and a transactional queue.

Also, if the database is Oracle Database 10g release 2 or later, the sort_list setting in CREATE_QUEUE_TABLE is set to commit_time. If the database is a release before Oracle Database 10g release 2, the sort_list setting in CREATE_QUEUE_TABLE is set to enq_time.

- If the specified queue table exists, then the queue uses the properties of the existing queue table.
- If the specified queue name does not exist, then this procedure runs the CREATE_QUEUE procedure in the DBMS_AQADM package to create the queue.
- This procedure starts the queue.
- If a queue user is specified, then this procedure configures this user as a secure queue user of the queue and grants ENQUEUE and DEQUEUE privileges on the queue to the specified queue user.

To configure the queue user as a secure queue user, this procedure creates an Advanced Queuing agent with the same name as the user name, if one does not exist. If an agent with this name exists and is associated with the queue user only, then it is used. SET_UP_QUEUE then runs the ENABLE_DB_ACCESS procedure in the DBMS_AQADM package, specifying the agent and the user.

---

**Note:**

If the agent that SET_UP_QUEUE tries to create exists and is associated with a user other than the user specified by queue_user, then the procedure raises an error. In this case, rename or remove the existing agent, and retry SET_UP_QUEUE.

---

**See Also:**

*Oracle Streams Concepts and Administration* for more information about secure queue users

---

### 212.4.39 SPLIT_STREAMS Procedure

This procedure splits one stream flowing from a capture process off from all of the other streams flowing from the capture process.

This procedure is intended for an Oracle Streams replication environment in which a capture process captures changes that are propagated to two or more destination databases. When one destination of a propagation stops accepting the captured changes, the changes remain in the capture process's queue. The queue can grow and begin to spill LCRs to the hard disk, degrading the performance of the Oracle Streams environment. A destination might stop accepting changes for several reasons. For example, the destination database might be down.
Specifically, this procedure performs the following actions:

1. Creates a new queue at the database running the capture process. The new queue is called the cloned queue because it is a clone of the queue used by the original stream. The new queue will be used by the new, cloned capture process, and it will be the source queue for the new, cloned propagation.

2. Creates a new propagation that propagates LCRs from the source queue created in Step 1 to the existing destination queue. The new propagation is called the cloned propagation because it is a clone of the propagation used by the original stream. The cloned propagation uses the same rule set as the original propagation.

3. Stops the capture process.

4. Queries the acknowledge SCN for the original propagation. The acknowledged SCN is the last SCN acknowledged by the apply process that applies the changes sent by the propagation.

5. Creates a new capture process. The new capture process is called the cloned capture process because it is a clone of the capture process used by the original stream. The procedure sets the start SCN for the cloned capture process to the value of the acknowledged SCN queried in Step 4. The cloned capture process uses the same rule set as the original capture process.

6. Drops the original propagation.

7. Starts the original capture process with the start SCN set to the acknowledged SCN queried in Step 4.

8. If the auto_merge_threshold parameter is set to a positive number, then creates an Oracle Scheduler job to run the MERGE_STREAMS_JOB procedure at set intervals according to its schedule. When the two streams are within the specified merge threshold, the MERGE_STREAMS_JOB procedure runs the MERGE_STREAMS procedure to merge the streams automatically.

After the SPLIT_STREAMS procedure has finished running, the cloned capture process is disabled. When the problem at the destination database is solved, and the destination queue can accept changes, you should start the cloned capture process using the START_CAPTURE procedure in the DBMS_CAPTURE_ADM package.

The DBA_STREAMS_SPLIT_MERGE view contains the name of each cloned component and information about the split and merge operation.

**Note:**

If the original capture process is a downstream capture process, then you must configure the cloned capture process to read the redo log from the source database before you start the cloned capture process.
Syntax

```sql
DBMS_XSTREAM_ADM.SPLIT_STREAMS(
    propagation_name         IN      VARCHAR2,
    cloned_propagation_name  IN      VARCHAR2  DEFAULT NULL,
    cloned_queue_name        IN      VARCHAR2  DEFAULT NULL,
    cloned_capture_name      IN      VARCHAR2  DEFAULT NULL,
    perform_actions          IN      BOOLEAN   DEFAULT TRUE,
    script_name              IN      VARCHAR2  DEFAULT NULL,
    script_directory_object  IN      VARCHAR2  DEFAULT NULL,
    auto_merge_threshold     IN      NUMBER    DEFAULT NULL,
    schedule_name            IN OUT  VARCHAR2,
    merge_job_name           IN OUT  VARCHAR2);
```

Parameters

**Table 212-36  SPLIT_STREAMS Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagation_name</td>
<td>The name of the propagation that cannot send LCRs to its destination queue. The specified propagation is the propagation for the stream that is being split off from the other streams. You must specify an existing propagation name. Do not specify an owner.</td>
</tr>
<tr>
<td>cloned_propagation_name</td>
<td>The name of the new propagation created by this procedure for the stream that is split off. If NULL, then the system generates a propagation name.</td>
</tr>
<tr>
<td>cloned_queue_name</td>
<td>The name of the new queue created by this procedure for the stream that is split off. If NULL, then the system generates a queue name.</td>
</tr>
<tr>
<td>cloned_capture_name</td>
<td>The name of the new capture process created by this procedure for the stream that is split off. If NULL, then the system generates a capture process name.</td>
</tr>
<tr>
<td>perform_actions</td>
<td>If TRUE, then the procedure performs the necessary actions to split the stream directly. If FALSE, then the procedure does not perform the necessary actions to split the stream directly. Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</td>
</tr>
<tr>
<td></td>
<td>* script_name</td>
</tr>
<tr>
<td></td>
<td>* script_directory_object</td>
</tr>
</tbody>
</table>
Table 212-36  (Cont.) SPLIT_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to split the stream. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file. If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to split the stream directly. If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to split the stream directly and does not generate a script. If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.</td>
</tr>
<tr>
<td>script_directory_object</td>
<td>The directory object for the directory on the local computer system into which the generated script is placed. If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script. If NULL and the script_name parameter is non-NULL, then the procedure raises an error. Note: The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</td>
</tr>
<tr>
<td>auto_merge_threshold</td>
<td>If a positive number is specified, then the stream that was split off is automatically merged back into all of the other streams flowing from the capture process by an Oracle Scheduler job. The job runs the MERGE_STREAMS_JOB procedure at set intervals according to its schedule. The value of the capture_message_create_time column for each capture process in the GV$STREAMS_CAPTURE dynamic performance view determines when the streams are merged. Specifically, if the difference, in seconds, between capture_message_create_time of the cloned capture process and the original capture process is less than or equal to the value specified for the auto_merge_threshold parameter, then the two streams are merged automatically. The cloned capture process must be started before the split stream can be merged back with the original stream. If NULL or 0 (zero) is specified, then the split stream is not merged back with the original stream automatically. To merge the split stream with the original stream, run the MERGE_STREAM procedure manually when the capture_message_create_time of the cloned capture process catches up to, or nearly catches up to, the capture_message_create_time of the original capture process. The capture_message_create_time records the time when a captured change was recorded in the redo log.</td>
</tr>
</tbody>
</table>
Table 212-36  **(Cont.)** SPLIT_STREAMS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_name</td>
<td>The Oracle Scheduler schedule name, specified as [schema_name.].schedule_name. For example, strmadmin.merge_schedule. If the schema is not specified, then the current user is the default. If auto_merge_threshold is a non-NULL positive number, then the schedule is used by the job that will automatically merge the streams at the appropriate time. You can specify a schedule name to adhere to naming conventions or to track the schedule more easily. If NULL and auto_merge_threshold is a non-NULL positive number, then the system generates a schedule name. If auto_merge_threshold is NULL or 0 (zero), then this parameter must be NULL. If this procedure creates a schedule, the schedule starts when the procedure completes. You can modify the schedule to control how often the merge job is run. If an existing schedule name is specified, an error is raised.</td>
</tr>
<tr>
<td>merge_job_name</td>
<td>The Oracle Scheduler job name, specified as [schema_name.].merge_job_name. For example, strmadmin.merge_job. If the schema is not specified, then the current user is the default. If auto_merge_threshold is a non-NULL positive number, then the job will automatically merge the streams at the appropriate time. Specify a merge job name to adhere to naming conventions or to track the job more easily. If NULL and auto_merge_threshold is a non-NULL positive number, then the system generates a job name. If auto_merge_threshold is NULL or 0 (zero), then this parameter must be NULL. If an existing job name is specified, an error is raised.</td>
</tr>
</tbody>
</table>

**See Also:**

*Oracle Database Administrator's Guide* for information about Oracle Scheduler

---

### 212.4.40 START_OUTBOUND Procedure

This procedure starts an XStream outbound server. The outbound server streams out the LCRs to an XStream client application.

**Syntax**

```sql
DBMS_XSTREAM_ADM.START_OUTBOUND(
    server_name        IN VARCHAR2);
```
Parameters

Table 212-37  START_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the outbound server being started. A NULL specification is not allowed. Do not specify an owner.</td>
</tr>
</tbody>
</table>

212.4.41 STOP_OUTBOUND Procedure

This procedure stops an XStream outbound server. The outbound server streams out the LCRs to an XStream client application.

Syntax

```
DBMS_XSTREAM_ADM.STOP_OUTBOUND(
    server_name  IN  VARCHAR2,
    force        IN  BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 212-38  STOP_OUTBOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>The name of the outbound server being stopped. A NULL specification is not allowed. Do not specify an owner.</td>
</tr>
<tr>
<td>force</td>
<td>If TRUE, then the procedure stops the outbound server and its capture process as soon as possible. If FALSE, then the procedure stops the outbound server after ensuring that there are no gaps in the set of applied transactions. The behavior of the apply component depends on the setting specified for the force parameter and the setting specified for the commit_serialization apply component parameter.</td>
</tr>
</tbody>
</table>
The DBMS_XSTREAM_AUTH package provides subprograms for granting privileges to and revoking privileges from XStream administrators.

This chapter contains the following topic:

- Overview
- Security Model
- Summary of DBMS_XSTREAM_AUTH Subprograms

See Also:

GRANT_ADMIN_PRIVILEGE Procedure

213.1 DBMS_XSTREAM_AUTH Overview

This package provides subprograms for granting privileges to XStream administrators and revoking privileges from XStream administrators.

213.2 DBMS_XSTREAM_AUTH Security Model

Security on this package can be controlled by either granting EXECUTE on this package to selected users or roles, or by granting EXECUTE_CATALOG_ROLE to selected users or roles.

The user executing the subprograms in the DBMS_XSTREAM_AUTH package must have SYSDBA administrative privilege, and the user must exercise the privilege using AS SYSDBA at connect time.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, connect as an administrative user who can create users, grant privileges, and create tablespaces when using this package.

213.3 Summary of DBMS_XSTREAM_AUTH Subprograms

This table lists the DBMS_XSTREAM_AUTH subprograms and briefly describes them.
### Table 213-1  DBMS_XSTREAM_AUTH Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT_ADMIN_PRIVILEGE Procedure</td>
<td>Either grants the privileges needed by a user to be an XStream administrator directly, or generates a script that grants these privileges.</td>
</tr>
<tr>
<td>GRANT_REMOTE_ADMIN_ACCESS Procedure</td>
<td>Enables a remote XStream administrator to perform administrative actions at the local database by connecting to the grantee using a database link.</td>
</tr>
<tr>
<td>REVOKE_ADMIN_PRIVILEGE Procedure</td>
<td>Either revokes XStream administrator privileges from a user directly, or generates a script that revokes these privileges.</td>
</tr>
<tr>
<td>REVOKE_REMOTE_ADMIN_ACCESS Procedure</td>
<td>Disables a remote XStream administrator from performing administrative actions by connecting to the grantee using a database link.</td>
</tr>
</tbody>
</table>

**Note:**

All subprograms commit unless specified otherwise.

### 213.3.1 GRANT_ADMIN_PRIVILEGE Procedure

This procedure either grants the privileges needed by a user to be an XStream administrator directly, or generates a script that grants these privileges.

**See Also:**

"GRANT_ADMIN_PRIVILEGE Procedure"

**Syntax**

```sql
DBMS_XSTREAM_AUTH.GRANT_ADMIN_PRIVILEGE(  
grantee IN VARCHAR2,  
privilege_type IN VARCHAR2 DEFAULT '*',  
grant_select_privileges IN BOOLEAN DEFAULT FALSE,  
do_grants IN BOOLEAN DEFAULT TRUE,  
file_name IN VARCHAR2 DEFAULT NULL,  
directory_name IN VARCHAR2 DEFAULT NULL  
grant_optional_privileges IN VARCHAR2 DEFAULT NULL,  
container IN VARCHAR2 DEFAULT 'CURRENT');
```
# Parameters

## Table 213-2  GRANT_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user to whom privileges are granted</td>
</tr>
<tr>
<td>privilege_type</td>
<td>Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• CAPTURE</td>
</tr>
<tr>
<td></td>
<td>Specifying CAPTURE grants the minimum privileges required by the user to administer capture processes.</td>
</tr>
<tr>
<td></td>
<td>• APPLY</td>
</tr>
<tr>
<td></td>
<td>Specifying APPLY grants the minimum privileges required by the user to administer outbound servers, inbound servers, and apply processes.</td>
</tr>
<tr>
<td></td>
<td>• *</td>
</tr>
<tr>
<td></td>
<td>Specifying * grants the minimum privileges required by the user to administer capture processes, outbound servers, inbound servers, and apply processes.</td>
</tr>
<tr>
<td>grant_select_privileges</td>
<td>If TRUE, then the procedure grants a set of privileges, including SELECT_CATALOG_ROLE, to the user.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not grant the set of privileges to the user.</td>
</tr>
<tr>
<td></td>
<td>SELECT_CATALOG_ROLE enables the user to select from the data dictionary. Set this parameter to TRUE for the XStream trusted user model.</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to FALSE for the XStream untrusted user model.</td>
</tr>
<tr>
<td>do_grants</td>
<td>If TRUE, then the procedure grants the privileges to the specified grantee directly, and adds the grantee to the DBA_XSTREAM_ADMINISTRATOR data dictionary view with YES for both the LOCAL_PRIVILEGES column and the ACCESS_FROM_REMOTE column. If the user already has an entry in this data dictionary view, then the procedure does not make another entry, and no error is raised. If TRUE and any of the grant statements fails, then the procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not grant the privileges to the specified grantee directly, and does not add the grantee to the DBA_XSTREAM_ADMINISTRATOR data dictionary view.</td>
</tr>
<tr>
<td></td>
<td>You specify FALSE when the procedure is generating a file that you will run later. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file generated by the procedure. The file contains all of the statements that grant the privileges. If a file with the specified file name exists in the specified directory name, then the grant statements are appended to the existing file.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not generate a file.</td>
</tr>
</tbody>
</table>
Table 213-2  (Cont.) GRANT_ADMIN_PRIVILEGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory_name</td>
<td>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have the WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>grant_optional_privileges</td>
<td>A comma-separated list of optional privileges to grant to the grantee, such as the DV_XSTREAM_ADMIN and DV_GOLDENGATE_ADMIN privileges</td>
</tr>
<tr>
<td>container</td>
<td>If CURRENT, then grants privileges to the grantee only in the container where the procedure is invoked. CURRENT can be specified while connected to the root or to a PDB. If ALL, then grants privileges to the grantee in all containers in the CDB and all PDBs created after the procedure is invoked. To specify ALL, the procedure must be invoked in the root. If a container name, then grants privileges to the grantee only in the specified container. To specify root, use CDB$ROOT while connected to the root. To specify a PDB, the procedure must be invoked in the root.</td>
</tr>
</tbody>
</table>

Usage Notes

The user who runs the procedure must be an administrative user who can grant privileges to other users.

Specifically, the procedure grants the following privileges to the specified user:

- The RESTRICTED SESSION system privilege
- EXECUTE on the following packages:
  - DBMS_APPLY_ADM
  - DBMS_AQ
  - DBMS_AQADM
  - DBMS_AQIN
  - DBMS_AQELM
  - DBMS_CAPTURE_ADM
  - DBMS_FLASHBACK
  - DBMS_LOCK
  - DBMS_PROPAGATION_ADM
  - DBMS_RULE_ADM
  - DBMS_STREAMS_ADM
  - DBMS_STREAMS_ADVISOR_ADM

Note: This parameter only applies to CDBs.
Privileges to enqueue messages into and dequeue messages from any queue

Privileges to manage any queue

Privileges to create, alter, and execute any of the following types of objects in the user's own schema and in other schemas:
  - Evaluation contexts
  - Rule sets
  - Rules

In addition, the grantee can grant these privileges to other users.

- `SELECT_CATALOG_ROLE`
- `SELECT` or `READ` privilege on data dictionary views related to XStream and Oracle Streams
- The ability to allow a remote XStream administrator to perform administrative actions through a database link by connecting to the grantee

This ability is enabled by running the `GRANT_REMOTE_ADMIN_ACCESS` procedure in this package.

**Note:**

- To view all of the statements run by the procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.
- This procedure grants only the privileges necessary to configure and administer an XStream environment. You can grant additional privileges to the grantee if necessary.

**See Also:**

- "GRANT_REMOTE_ADMIN_ACCESS Procedure"
- Oracle Database SQL Language Reference for information about the `CREATE DIRECTORY` SQL statement
213.3.2 GRANT_REMOTE_ADMIN_ACCESS Procedure

This procedure enables a remote XStream administrator to perform administrative actions at the local database by connecting to the grantee using a database link.

Syntax

```
DBMS_XSTREAM_AUTH.GRANT_REMOTE_ADMIN_ACCESS(
    grantee  IN  VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user who allows remote access. The procedure adds the grantee to the DBA_XSTREAM_ADMINISTRATOR data dictionary view with YES for the ACCESS_FROM_REMOTE column. If the user already has an entry in this data dictionary view, then the procedure does not make another entry. Instead, it updates the ACCESS_FROM_REMOTE column to YES.</td>
</tr>
</tbody>
</table>

Usage Notes

Typically, you run the procedure and specify a grantee at a local source database if a downstream capture process captures changes originating at the local source database. The XStream administrator at a downstream capture database administers the source database using this connection.

Note:

The GRANT_ADMIN_PRIVILEGE procedure in this package runs this procedure.

See Also:

"GRANT_ADMIN_PRIVILEGE Procedure"

213.3.3 REVOKE_ADMIN_PRIVILEGE Procedure

This procedure either revokes XStream administrator privileges from a user directly, or generates a script that revokes these privileges.

Syntax

```
DBMS_XSTREAM_AUTH.REVOKE_ADMIN_PRIVILEGE(
    grantee    IN      VARCHAR2,
    privilege_type  IN      VARCHAR2  DEFAULT '',
    revoke_select_privileges  IN      BOOLEAN  DEFAULT FALSE,
    do_revokes    IN      BOOLEAN  DEFAULT TRUE,
```

Parameters

Table 213-4  **REVOKE_ADMIN_PRIVILEGE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user from whom privileges are revoked</td>
</tr>
<tr>
<td>privilege_type</td>
<td>Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td>•  CAPTURE</td>
</tr>
<tr>
<td></td>
<td>Specifying CAPTURE revokes the minimum privileges required by the</td>
</tr>
<tr>
<td></td>
<td>user to administer capture processes.</td>
</tr>
<tr>
<td></td>
<td>•  APPLY</td>
</tr>
<tr>
<td></td>
<td>Specifying APPLY revokes the minimum privileges required by the user</td>
</tr>
<tr>
<td></td>
<td>to administer outbound servers, inbound servers, and apply processes.</td>
</tr>
<tr>
<td></td>
<td>•  *</td>
</tr>
<tr>
<td></td>
<td>Specifying * revokes the minimum privileges required by the user to</td>
</tr>
<tr>
<td></td>
<td>administer capture processes, outbound servers, inbound servers, and</td>
</tr>
<tr>
<td></td>
<td>apply processes.</td>
</tr>
<tr>
<td>revoke_select_privil</td>
<td>If TRUE, then the procedure revokes a set of privileges, including SELECT</td>
</tr>
<tr>
<td>eges</td>
<td>_CATALOG_ROLE, to the user.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not revoke the set of privileges to the</td>
</tr>
<tr>
<td></td>
<td>user.</td>
</tr>
<tr>
<td></td>
<td>SELECT_CATALOG_ROLE enables the user to select from the data dictionary.</td>
</tr>
<tr>
<td>do_revokes</td>
<td>If TRUE, then the procedure revokes the privileges from the specified user</td>
</tr>
<tr>
<td></td>
<td>directly, and removes the user from the DBA_XSTREAM_ADMINISTRATOR</td>
</tr>
<tr>
<td></td>
<td>data dictionary view. If the user does not have a record in this data</td>
</tr>
<tr>
<td></td>
<td>dictionary view, then the procedure does not remove a record from the</td>
</tr>
<tr>
<td></td>
<td>view, and no error is raised. If TRUE and any of the revoke statements</td>
</tr>
<tr>
<td></td>
<td>fails, then the procedure raises an error. A revoke statement fails if</td>
</tr>
<tr>
<td></td>
<td>the user is not granted the privilege that is being revoked.</td>
</tr>
<tr>
<td></td>
<td>If FALSE, then the procedure does not revoke the privileges from the</td>
</tr>
<tr>
<td></td>
<td>specified user directly, and does not remove the user from the</td>
</tr>
<tr>
<td></td>
<td>DBA_XSTREAM_ADMINISTRATOR data dictionary view.</td>
</tr>
<tr>
<td></td>
<td>You specify FALSE when the procedure is generating a file that you will run</td>
</tr>
<tr>
<td></td>
<td>later. If you specify FALSE and either the file_name or directory_name</td>
</tr>
<tr>
<td></td>
<td>parameter is NULL, then the procedure does not raise an error.</td>
</tr>
<tr>
<td>file_name</td>
<td>The name of the file generated by this procedure. The file contains all of</td>
</tr>
<tr>
<td></td>
<td>the statements that revoke the privileges. If a file with the specified</td>
</tr>
<tr>
<td></td>
<td>file name exists in the specified directory name, then the revoke</td>
</tr>
<tr>
<td></td>
<td>statements are appended to the existing file.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure does not generate a file.</td>
</tr>
</tbody>
</table>
Table 213-4  (Cont.) **REVOKE_ADMIN_PRIVILEGE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory_name</td>
<td>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have the WRITE privilege on the directory object. If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file. If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</td>
</tr>
<tr>
<td>revoke_optional_privileges</td>
<td>A comma-separated list of optional privileges to revoke from the grantee, such as the DV_XSTREAM_ADMIN and DV_GOLDENGATE_ADMIN privileges</td>
</tr>
<tr>
<td>container</td>
<td>If CURRENT, then revokes privileges from the grantee only in the container where the procedure is invoked. CURRENT can be specified while connected to the root or to a PDB. If ALL, then revokes privileges from the grantee in all containers in the CDB. To specify ALL, the procedure must be invoked in the root. If a container name, then revokes privileges from the grantee only in the specified container. To specify root, use CDB$ROOT while connected to the root. To specify a PDB, the procedure must be invoked in the root. <strong>Note:</strong> This parameter only applies to CDBs.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The user who runs this procedure must be an administrative user who can revoke privileges from other users. Specifically, this procedure revokes the privileges granted by running the **GRANT_ADMIN_PRIVILEGE** procedure in this package.

**Note:**

To view all of the statements run by this procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.

**See Also:**

- "**GRANT_ADMIN_PRIVILEGE Procedure**"
- *Oracle Database SQL Language Reference* for information about the CREATE_DIRECTORY SQL statement
213.3.4 REVOKE_REMOTE_ADMIN_ACCESS Procedure

This procedure disables a remote XStream administrator from performing administrative actions by connecting to the grantee using a database link.

**Note:**

The `REVOKE_ADMIN_PRIVILEGE` procedure in this package runs this procedure.

**See Also:**

"REVOKE_ADMIN_PRIVILEGE Procedure"

**Syntax**

```
DBMS_XSTREAM_AUTH.REVOKE_REMOTE_ADMIN_ACCESS(
    grantee  IN  VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>The user for whom access from a remote XStream administrator is disabled. If a row for the grantee exists in the <code>DBA_XSTREAM_ADMINISTRATOR</code> data dictionary view, then the procedure updates the <code>ACCESS_FROM_REMOTE</code> column for the grantee to <code>NO</code>. If, after this update, both the <code>LOCAL_PRIVILEGES</code> column and the <code>ACCESS_FROM_REMOTE</code> column are <code>NO</code> for the grantee, then the procedure removes the grantee from the view. If no row for the grantee exists in the <code>DBA_XSTREAM_ADMINISTRATOR</code> data dictionary view, then the procedure does not update the view and does not raise an error.</td>
</tr>
</tbody>
</table>
The `DEBUG_EXTPROC` package enables you to start up the extproc agent within a session. This utility package can help you debug external procedures.

This chapter contains the following topics:

- Security Model
- Operational Notes
- Rules and Limits
- Summary of `DEBUG_EXTPROC` Subprograms

### 214.1 DEBUG_EXTPROC Security Model

Your Oracle account must have `EXECUTE` privileges on the package and `CREATE LIBRARY` privileges.

### 214.2 DEBUG_EXTPROC Operational Notes

These operational notes apply to `DEBUG_EXTPROC`.

To install the package, run the script `DBGEXTP.SQL`.

- Install/load this package in the Oracle USER where you want to debug the `extproc` process.
- Ensure that you have execute privileges on package `DEBUG_EXTPROC`

```sql
SELECT SUBSTR(OBJECT_NAME, 1, 20)
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'DEBUG_EXTPROC';
```

- You can install this package as any other user, as long as you have `EXECUTE` privileges on the package.

**Note:**

These notes assumes that you built your shared library with debug symbols to aid in the debugging process. Please check the C compiler manual pages for the appropriate C compiler switches to build the shared library with debug symbols.

Having installed the package, proceed accordingly:

- Start a new Oracle session through SQL*Plus or OCI program by connecting to `ORACLE`. 
- Execute procedure `DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT` to startup the extproc agent in this session; for example, execute `DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT`. Do not exit this session, because that terminates the extproc agent.

- Determine the PID of the extproc agent that was started up for this session.

- Using a debugger (for example, gdb, dbx, or the native system debugger), load the extproc executable and attach to the running process.

- Set a breakpoint on function 'pextproc' and let the debugger continue with its execution.

- Now execute your external procedure in the same session where you first executed `DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT`.

- Your debugger should now break in function 'pextproc'. At this point in time, the shared library referenced by your PL/SQL external function would have been loaded and the function resolved. Now set a breakpoint in your C function and let the debugger continue its execution.

Because PL/SQL loads the shared library at runtime, the debugger you use may or may not automatically be able to track the new symbols from the shared library. You may have to issue some debugger command to load the symbols (for example, 'share' in gdb).

- The debugger should now break in your C function. Its assumed that you had built the shared library with debugging symbols.

- Now proceed with your debugging.

### 214.3 Rules and Limits

`DEBUG_EXTPROC` works only on platforms with debuggers that can attach to a running process.

### 214.4 Summary of DEBUG_EXTPROC Subprograms

The `STARTUP_EXTPROC_AGENT` procedure is the only `DEBUG_EXTPROC` subprogram.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTUP_EXTPROC_AGENT Procedure</td>
<td>Starts up the extproc agent process in the session</td>
</tr>
</tbody>
</table>

### 214.4.1 STARTUP_EXTPROC_AGENT Procedure

This procedure starts up the extproc agent process in the session. This enables you to get the PID of the executing process. This PID is needed to be able to attach to the running process using a debugger.

**Syntax**

```
DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT;
```
The HTF (hypertext functions) and HTP (hypertext procedures) packages generate HTML tags. For example, the HTF.ANCHOR function generates the HTML anchor tag, `<A>`.

**See Also:**

For more information about implementation of this package:

- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

This chapter contains the following topics:

- Deprecated Subprograms
- Operational Notes
- Rules and Limits
- Examples
- Summary of Tags
- Summary of HTF Subprograms

### 215.1 HTF Deprecated Subprograms

The **ESCAPE_URL** function is deprecated with Oracle Database 10g.

**Note:**

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

**Related Topics**

- **ESCAPE_URL Function**
  This deprecated function replaces characters that have special meaning in HTML and HTTP with their escape sequences.
215.2 HTF Operational Notes

These operational notes apply to the HTF package.

For every HTF function that generates one or more HTML tags, there is a corresponding HTP procedure with identical parameters with the following exception:

- The PRINTS Procedure and the PS Procedure do not have HTF function equivalents. Use the ESCAPE_SC Function or the ESCAPE_URL Function if you need a string conversion function. Note that while there is an ESCAPE_SC Procedure that performs the same operation as the PRINTS Procedure and the PS Procedure, there is no procedural equivalent for the ESCAPE_URL Function.

- The FORMAT_CELL Function does not have an HTP equivalent. The function formats column values inside an HTML table using TABLEDATA Function which does have an HTP equivalent in the TABLEDATA Procedure. The advantage of this using the FORMAT_CELL Function is that it allows for better control over the HTML tables.

The function versions do not directly generate output in your Web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls. To print the output of HTF functions, call the functions from within the HTF.PRINT function. It then prints its parameters to the generated Web page.

215.3 Rules and Limits

If you use values of the LONG datatype in functions such as HTF.PRINT, HTF.PRN, HTF.PA or OWA_UTIL.CELLSPRINT, only the first 32 K of the LONG data is used. The LONG data is bound to a VARCHAR2 datatype in the function.

215.4 HTF Examples

The commands in this example generate a simple HTML document.

```sql
CREATE OR REPLACE PROCEDURE hello AS
BEGIN
  HTP.P (HTF.HTMLOPEN); -- generates <HTML>
  HTP.P (HTF.HEADOPEN); -- generates <HEAD>
  HTP.P (HTF.TITLE('Hello')); -- generates <TITLE>Hello</TITLE>
  HTP.P (HTF.HEADCLOSE); -- generates </HEAD>
  HTP.P (HTF.BODYOPEN); -- generates <BODY>
  HTP.P (HTF.HEADER(1, 'Hello')); -- generates <H1>Hello</H1>
  HTP.P (HTF.BODYCLOSE); -- generates </BODY>
  HTP.P (HTF.HTMLCLOSE); -- generates </HTML>
END;
```

215.5 Summary of Tags

This list summarizes the HTML tags generated by the HTF package.

**HTML, HEAD, and BODY Tags**

- **HTMLOPEN Function, HTMLCLOSE Function** - generate `<HTML>` and `</HTML>`
HEADOPEN Function, HEADCLOSE Function - generate `<HEAD>` and `</HEAD>`

BODYOPEN Function, BODYCLOSE Function - generate `<BODY>` and `</BODY>`

Comment Tag

COMMENT Function - generates `<!--` and `-->`

http://www.w3.org.BASE Function - generates `<BASE>`

LINKREL Function - generates `<LINK>` with the `REL` attribute

LINKREV Function - generates `<LINK>` with the `REV` attribute

TITLE Function - generates `<TITLE>`

META Function - generates `<META>`

SCRIPT Function - generates `<SCRIPT>`

STYLE Function - generates `<STYLE>`

ISINDEX Function - generates `<ISINDEX>`

Applet Tags

APPLETOPEN Function, APPLETCLOSE Function - generate `<APPLET>` and `</APPLET>`

PARAM Function - generates `<PARAM>`

List Tags

OLISTOPEN Function, OLISTCLOSE Function - generate `<OL>` and `</OL>`

ULISTOPEN Function, ULISTCLOSE Function - generate `<UL>` and `</UL>`

DLISTOPEN Function, DLISTCLOSE Function - generate `<DL>` and `</DL>`

DLISTTERM Function - generates `<DT>`

DLISTDEF Function - generates `<DD>`

DIREFIXLISTOPEN Function, DIREFIXLISTCLOSE Function - generate `<DIR>` and `</DIR>`

LISTHEADER Function - generates `<LH>`

LISTINGOPEN Function, LISTINGCLOSE Function - generate `<LISTING>` and `</LISTING>`

MENULISTOPEN Function - generate `<MENU>` and `</MENU>`

LISTITEM Function - generates `<LI>`

Form Tags

FORMOPEN Function, FORMCLOSE Function - generate `<FORM>` and `</FORM>`

FORMCHECKBOX Function - generates `<INPUT TYPE="CHECKBOX">`

FORMHIDDEN Function - generates `<INPUT TYPE="HIDDEN">`

FORMIMAGE Function - generates `<INPUT TYPE="IMAGE">`
FORMPASSWORD Function - generates `<INPUT TYPE="PASSWORD">`
FORMRADIO Function - generates `<INPUT TYPE="RADIO">`
FORMSELECTOPEN Function, FORMSELECTCLOSE Function - generate `<SELECT>` and `</SELECT>`
FORMSELECTOPTION Function - generates `<OPTION>`
FORMTEXT Function - generates `<INPUT TYPE="TEXT">`
FORMTEXTAREA Function - generates `<TEXTAREA>`
FORMTEXTAREAOPTION Function, FORMTEXTAREACLOSE Function - generate `<TEXTAREA>` and `</TEXTAREA>`
FORMRESET Function - generates `<INPUT TYPE="RESET">`
FORMSUBMIT Function - generates `<INPUT TYPE="SUBMIT">`

Table Tags
TABLEOPEN Function, TABLECLOSE Function - generate `<TABLE>` and `</TABLE>`
TABLECAPTION Function - generates `<CAPTION>`
TABLEROWOPEN Function, TABLEROWCLOSE Function - generate `<TR>` and `</TR>`
TABLEHEADER Function - generates `<TH>`
TABLEDATA Function - generates `<TD>`

IMG, HR, and A Tags
HR Function, LINE Function - generate `<HR>`
IMG Function, IMG2 Function - generate `<IMG>`
ANCHOR Function, ANCHOR2 Function - generate `<A>`
MAPOPEN Function, MAPCLOSE Function - generate `<MAP>` and `</MAP>`

Paragraph Formatting Tags
HEADER Function - generates heading tags (`<H1>` to `<H6>`) PARA Function, PARAGRAPH Function - generate `<p>`
PRN Functions, PRINT Functions - generate any text that is passed in
PRN Functions, S Function - generate any text that is passed in; special characters in HTML are escaped
PREOPEN Function, PRECLOSE Function - generate `<PRE>` and `</PRE>`
BLOCKQUOTEOPEN Function, BLOCKQUOTECLOSE Function - generate `<BLOCKQUOTE>` and `</BLOCKQUOTE>`
DIV Function - generates `<DIV>`
NL Function, BR Function - generate `<BR>`
NOBR Function - generates `<NOBR>`
WBR Function - generates `<WBR>`
PLAINTEXT Function - generates `<PLAINTEXT>`
ADDRESS Function - generates `<ADDRESS>`
MAILTO Function - generates `<A>` with the MAILTO attribute
AREA Function - generates `<AREA>`
BGSOUND Function - generates `<BGSOUND>`

Character Formatting Tags
BASEFONT Function - generates `<BASEFONT>`
BIG Function - generates `<BIG>`
BOLD Function - generates `<B>`
CENTER Function - generates `<CENTER>` and `</CENTER>`
CENTEROPEN Function, CENTERCLOSE Function - generate `<CENTER>` and `</CENTER>`
CITE Function - generates `<CITE>`
CODE Function - generates `<CODE>`
DFN Function - generates `<DFN>`
EM Function, EMPHASIS Function - generate `<EM>`
FONTOPEN Function, FONTCLOSE Function - generate `<FONT>` and `</FONT>`
ITALIC Function - generates `<I>`
KBD Function, KEYBOARD Function - generate `<KBD>` and `</KBD>`
S Function - generates `<S>`
SAMPLE Function - generates `<SAMP>`
SMALL Function - generates `<SMALL>`
STRIKE Function - generates `<STRIKE>`
STRONG Function - generates `<STRONG>`
SUB Function - generates `<SUB>`
SUP Function - generates `<SUP>`
TELETYPE Function - generates `<TT>`
UNDERLINE Function - generates `<U>`
VARIABLE Function - generates `<VAR>`

Frame Tags
FRAME Function - generates `<FRAME>`
**215.6 Summary of HTF Subprograms**

This table lists the HTF subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS Function</td>
<td>Generates the <code>&lt;ADDRESS&gt;</code> and <code>&lt;/ADDRESS&gt;</code> tags which specify the address, author and signature of a document</td>
</tr>
<tr>
<td>ANCHOR Function</td>
<td>Generates the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags which specify the source or destination of a hypertext link</td>
</tr>
<tr>
<td>ANCHOR2 Function</td>
<td>Generates the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags which specify the source or destination of a hypertext link</td>
</tr>
<tr>
<td>APPELTCLOSE Function</td>
<td>Closes the applet invocation with the <code>&lt;/APPLET&gt;</code> tag</td>
</tr>
<tr>
<td>APPELTOPEN Function</td>
<td>Generates the <code>&lt;APPLET&gt;</code> tag which begins the invocation of a Java applet</td>
</tr>
<tr>
<td>AREA Function</td>
<td>Generates the <code>&lt;AREA&gt;</code> tag, which defines a client-side image map</td>
</tr>
<tr>
<td>BASE Function</td>
<td>Generates the <code>&lt;BASE&gt;</code> tag which records the URL of the document</td>
</tr>
<tr>
<td>BASEFONT Function</td>
<td>Generates the <code>&lt;BASEFONT&gt;</code> tag which specifies the base font size for a Web page</td>
</tr>
<tr>
<td>BGSOUND Function</td>
<td>Generates the <code>&lt;BGSOUND&gt;</code> tag which includes audio for a Web page</td>
</tr>
<tr>
<td>BIG Function</td>
<td>Generates the <code>&lt;BIG&gt;</code> and <code>&lt;/BIG&gt;</code> tags which direct the browser to render the text in a bigger font</td>
</tr>
<tr>
<td>BLOCKQUOTECLOSE Function</td>
<td>Generates the <code>&lt;/BLOCKQUOTE&gt;</code> tag which mark the end of a section of quoted text</td>
</tr>
<tr>
<td>BLOCKQUOTEOPEN Function</td>
<td>Generates the <code>&lt;BLOCKQUOTE&gt;</code> tag, which marks the beginning of a section of quoted text</td>
</tr>
<tr>
<td>BODYCLOSE Function</td>
<td>Generates the <code>&lt;/BODY&gt;</code> tag which marks the end of a body section of an HTML document</td>
</tr>
<tr>
<td>BODYOPEN Function</td>
<td>Generates the <code>&lt;BODY&gt;</code> tag which marks the beginning of the body section of an HTML document</td>
</tr>
<tr>
<td>BOLD Function</td>
<td>Generates the <code>&lt;B&gt;</code> and <code>&lt;/B&gt;</code> tags which direct the browser to display the text in boldface</td>
</tr>
<tr>
<td>BR Function</td>
<td>Generates the <code>&lt;BR&gt;</code> tag which begins a new line of text</td>
</tr>
<tr>
<td>CENTER Function</td>
<td>Generates the <code>&lt;CENTER&gt;</code> and <code>&lt;/CENTER&gt;</code> tags which center a section of text within a Web page</td>
</tr>
<tr>
<td>CENTERCLOSE Function</td>
<td>Generates the <code>&lt;/CENTER&gt;</code> tag which marks the end of a section of text to center</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CENTEROPEN Function</td>
<td>Generates the &lt;CENTER&gt; tag which mark the beginning of a section of text to center</td>
</tr>
<tr>
<td>CITE Function</td>
<td>Generates the &lt;CITE&gt; and &lt;/CITE&gt; tags which direct the browser to render the text as a citation</td>
</tr>
<tr>
<td>CODE Function</td>
<td>Generates the &lt;CODE&gt; and &lt;/CODE&gt; tags which direct the browser to render the text in monospace font or however “code” is defined stylistically</td>
</tr>
<tr>
<td>COMMENT Function</td>
<td>Generates the comment tags &lt;!-- ctext --&gt;</td>
</tr>
<tr>
<td>DFN Function</td>
<td>Generates the &lt;DFN&gt; and &lt;/DFN&gt; tags which direct the browser to mark the text as italics or however “definition” is defined stylistically</td>
</tr>
<tr>
<td>DIRLISTCLOSE Function</td>
<td>Generates the &lt;/DIR&gt; tag which ends a directory list section</td>
</tr>
<tr>
<td>DIRLISTOPEN Function</td>
<td>Generates the &lt;DIR&gt; which starts a directory list section</td>
</tr>
<tr>
<td>DIV Function</td>
<td>Generates the &lt;DIV&gt; tag which creates document divisions</td>
</tr>
<tr>
<td>DLISTCLOSE Function</td>
<td>Generates the &lt;/DL&gt; tag which ends a definition list</td>
</tr>
<tr>
<td>DLISTDEF Function</td>
<td>Generates the &lt;DD&gt; tag, which inserts definitions of terms</td>
</tr>
<tr>
<td>DLISTOPEN Function</td>
<td>Generates the &lt;DL&gt; tag which starts a definition list</td>
</tr>
<tr>
<td>DLISTTERM Function</td>
<td>Generates the &lt;DT&gt; tag which defines a term in a definition list &lt;DL&gt;</td>
</tr>
<tr>
<td>EM Function</td>
<td>Generates the &lt;EM&gt; and &lt;/EM&gt; tags, which define text to be emphasized</td>
</tr>
<tr>
<td>EMPHASIS Function</td>
<td>Generates the &lt;EM&gt; and &lt;/EM&gt; tags, which define text to be emphasized</td>
</tr>
<tr>
<td>ESCAPE_SC Function</td>
<td>Replaces characters that have special meaning in HTML with their escape sequences</td>
</tr>
<tr>
<td>ESCAPE_URL Function</td>
<td>Replaces characters that have special meaning in HTML and HTTP with their escape sequences</td>
</tr>
<tr>
<td>FONTCLOSE Function</td>
<td>Generates the &lt;/FONT&gt; tag which marks the end of a section of text with the specified font characteristics</td>
</tr>
<tr>
<td>FONTOPEN Function</td>
<td>Generates the &lt;FONT&gt; which marks the beginning of section of text with the specified font characteristics</td>
</tr>
<tr>
<td>FORMAT_CELL Function</td>
<td>formats column values inside an HTML table using the TABLEDATA Function</td>
</tr>
<tr>
<td>FORMCHECKBOX Function</td>
<td>Generates the &lt;INPUT&gt; tag with TYPE=&quot;checkbox&quot; which inserts a checkbox element in a form</td>
</tr>
<tr>
<td>FORMCLOSE Function</td>
<td>Generates the &lt;/FORM&gt; tag which marks the end of a form section in an HTML document</td>
</tr>
<tr>
<td>FORMFILE Function</td>
<td>Generates the &lt;INPUT&gt; tag with TYPE=&quot;file&quot; which inserts a file form element, and is used for file uploading for a given page</td>
</tr>
<tr>
<td>FORMHIDDEN Function</td>
<td>Generates the &lt;INPUT&gt; tag with TYPE=&quot;hidden&quot; which inserts a hidden form element</td>
</tr>
</tbody>
</table>
### Table 215-1 (Cont.) HTF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMIMAGE Function</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;image&quot; which creates an image field that the user clicks to submit the form immediately</td>
</tr>
<tr>
<td>FORMOPEN Function</td>
<td>Generates the <code>&lt;FORM&gt;</code> tag which marks the beginning of a form section in an HTML document</td>
</tr>
<tr>
<td>FORMPASSWORD Function</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;password&quot; which creates a single-line text entry field</td>
</tr>
<tr>
<td>FORMRADIO Function</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;radio&quot;, which creates a radio button on the HTML form</td>
</tr>
<tr>
<td>FORMRESET Function</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;reset&quot; which creates a button that, when selected, resets the form fields to their initial values</td>
</tr>
<tr>
<td>FORMSELECTCLOSE Function</td>
<td>Generates the <code>/SELECT</code> tag which marks the end of a Select form element</td>
</tr>
<tr>
<td>FORMSELECTOPEN Function</td>
<td>Generates the <code>/SELECT</code> tag which marks the beginning of a Select form element</td>
</tr>
<tr>
<td>FORMSELECTOPTION Function</td>
<td>Generates the <code>&lt;OPTION&gt;</code> tag which represents one choice in a Select element</td>
</tr>
<tr>
<td>FORMSUBMIT Function</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;submit&quot; which creates a button that, when clicked, submits the form</td>
</tr>
<tr>
<td>FORMTEXT Function</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;text&quot;, which creates a field for a single line of text</td>
</tr>
<tr>
<td>FORMTEXTAREA Function</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> tag, which creates a text field that has no predefined text in the text area</td>
</tr>
<tr>
<td>FORMTEXTAREA2 Function</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> tag, which creates a text field that has no predefined text in the text area with the ability to specify a wrap style</td>
</tr>
<tr>
<td>FORMTEXTAREACLOSE Function</td>
<td>Generates the <code>/TEXTAREA</code> tag which ends a text area form element</td>
</tr>
<tr>
<td>FORMTEXTAREAOPEN Function</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> which marks the beginning of a text area form element</td>
</tr>
<tr>
<td>FORMTEXTAREAOPEN2 Function</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> which marks the beginning of a text area form element with the ability to specify a wrap style</td>
</tr>
<tr>
<td>FRAME Function</td>
<td>Generates the <code>&lt;FRAME&gt;</code> tag which defines the characteristics of a frame created by a <code>&lt;FRAMESET&gt;</code> tag</td>
</tr>
<tr>
<td>FRAMESETCLOSE Function</td>
<td>Generates the <code>/FRAMESET</code> tag which ends a frameset section</td>
</tr>
<tr>
<td>FRAMESETOPEN Function</td>
<td>Generates the <code>/FRAMESET</code> tag which begins a frameset section</td>
</tr>
<tr>
<td>HEADCLOSE Function</td>
<td>Generates the <code>/HEAD</code> tag which marks the end of an HTML document head section</td>
</tr>
<tr>
<td>HEADER Function</td>
<td>Generates opening heading tags (<code>&lt;H1&gt;</code> to <code>&lt;H6&gt;</code>) and their corresponding closing tags (<code>&lt;/H1&gt;</code> to <code>&lt;/H6&gt;</code>)</td>
</tr>
</tbody>
</table>
Table 215-1  (Cont.) HTF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADOPEN Function</td>
<td>Generates the <code>&lt;HEAD&gt;</code> tag which marks the beginning of the HTML document</td>
</tr>
<tr>
<td>HR Function</td>
<td>Generates the <code>&lt;HR&gt;</code> tag, which generates a line in the HTML document</td>
</tr>
<tr>
<td>HTMLCLOSE Function</td>
<td>Generates the <code>&lt;/HTML&gt;</code> tag which marks the end of an HTML document</td>
</tr>
<tr>
<td>HTMLOPEN Function</td>
<td>Generates the <code>&lt;HTML&gt;</code> tag which marks the beginning of an HTML document</td>
</tr>
<tr>
<td>IMG Function</td>
<td>Generates the <code>&lt;IMG&gt;</code> tag which directs the browser to load an image onto the HTML page</td>
</tr>
<tr>
<td>IMG2 Function</td>
<td>Generates the <code>&lt;IMG&gt;</code> tag which directs the browser to load an image onto the HTML page with the option of specifying values for the <code>USEMAP</code> attribute</td>
</tr>
<tr>
<td>ISINDEX Function</td>
<td>Creates a single entry field with a prompting text, such as “enter value,” then sends that value to the URL of the page or program</td>
</tr>
<tr>
<td>ITALIC Function</td>
<td>Generates the <code>&lt;I&gt;</code> and <code>&lt;/I&gt;</code> tags which direct the browser to render the text in italics</td>
</tr>
<tr>
<td>KBD Function</td>
<td>Generates the <code>&lt;KBD&gt;</code> and <code>&lt;/KBD&gt;</code> tags which direct the browser to render the text in monospace font</td>
</tr>
<tr>
<td>KEYBOARD Function</td>
<td>Generates the <code>&lt;KBD&gt;</code> and <code>&lt;/KBD&gt;</code> tags, which direct the browser to render the text in monospace font</td>
</tr>
<tr>
<td>LINE Function</td>
<td>Generates the <code>&lt;HR&gt;</code> tag, which generates a line in the HTML document</td>
</tr>
<tr>
<td>LINKREL Function</td>
<td>Generates the <code>&lt;LINK&gt;</code> tag with the <code>REL</code> attribute which delineates the relationship described by the hypertext link from the anchor to the target</td>
</tr>
<tr>
<td>LINKREV Function</td>
<td>Generates the <code>&lt;LINK&gt;</code> tag with the <code>REV</code> attribute which delineates the relationship described by the hypertext link from the target to the anchor</td>
</tr>
<tr>
<td>LISTHEADER Function</td>
<td>Generates the <code>&lt;LH&gt;</code> and <code>&lt;/LH&gt;</code> tags which print an HTML tag at the beginning of the list</td>
</tr>
<tr>
<td>LISTINGCLOSE Function</td>
<td>Generates the <code>&lt;/LISTING&gt;</code> tags which marks the end of a section of fixed-width text in the body of an HTML page</td>
</tr>
<tr>
<td>LISTINGOPEN Function</td>
<td>Generates the <code>&lt;LISTING&gt;</code> tag which marks the beginning of a section of fixed-width text in the body of an HTML page</td>
</tr>
<tr>
<td>LISTITEM Function</td>
<td>Generates the <code>&lt;LI&gt;</code> tag, which indicates a list item</td>
</tr>
<tr>
<td>MAILTO Function</td>
<td>Generates the <code>&lt;A&gt;</code> tag with the <code>HREF</code> set to ‘mailto’ prepended to the mail address argument</td>
</tr>
<tr>
<td>MAPCLOSE Function</td>
<td>Generates the <code>&lt;/MAP&gt;</code> tag which marks the end of a set of regions in a client-side image map</td>
</tr>
<tr>
<td>MAPOPEN Function</td>
<td>Generates the <code>&lt;MAP&gt;</code> tag which mark the beginning of a set of regions in a client-side image map</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MENULISTCLOSE Function</td>
<td>Generates the &lt;/MENU&gt; tag which ends a list that presents one line for each item</td>
</tr>
<tr>
<td>MENULISTOPEN Function</td>
<td>Generates the &lt;MENU&gt; tag which begins a list that presents one line for each item</td>
</tr>
<tr>
<td>META Function</td>
<td>Generates the &lt;META&gt; tag, which embeds meta-information about the document and also specifies values for HTTP headers</td>
</tr>
<tr>
<td>NL Function</td>
<td>Generates the &lt;BR&gt; tag which begins a new line of text</td>
</tr>
<tr>
<td>NOBR Function</td>
<td>Generates the &lt;NOBR&gt; and &lt;/NOBR&gt; tags which turn off line-breaking in a section of text</td>
</tr>
<tr>
<td>NOFRAMESCLOSE Function</td>
<td>Generates the &lt;/NOFRAMES&gt; tag which marks the end of a no-frames section</td>
</tr>
<tr>
<td>NOFRAMESOPEN Function</td>
<td>Generates the &lt;NOFRAMES&gt; tag which marks the beginning of a no-frames section</td>
</tr>
<tr>
<td>OLISTCLOSE Function</td>
<td>Generates the &lt;/OL&gt; tag which defines the end of an ordered list</td>
</tr>
<tr>
<td>OLISTOPEN Function</td>
<td>Generates the &lt;OL&gt; tag which marks the beginning of an ordered list</td>
</tr>
<tr>
<td>PARA Function</td>
<td>Generates the &lt;P&gt; tag which indicates that the text that comes after the tag is to be formatted as a paragraph</td>
</tr>
<tr>
<td>PARAGRAPH Function</td>
<td>Adds attributes to the &lt;P&gt; tag</td>
</tr>
<tr>
<td>PARAM Function</td>
<td>Generates the &lt;PARAM&gt; tag which specifies parameter values for Java applets</td>
</tr>
<tr>
<td>PLAINTEXT Function</td>
<td>Generates the &lt;PLAINTEXT&gt; and &lt;/PLAINTEXT&gt; tags which direct the browser to render the text they surround in fixed-width type</td>
</tr>
<tr>
<td>PRECLOSE Function</td>
<td>Generates the &lt;/PRE&gt; tag which marks the end of a section of preformatted text in the body of the HTML page</td>
</tr>
<tr>
<td>PREOPEN Function</td>
<td>Generates the &lt;PRE&gt; tag which marks the beginning of a section of preformatted text in the body of the HTML page</td>
</tr>
<tr>
<td>PRINT Functions</td>
<td>Generates the specified parameter as a string terminated with the \n newline character</td>
</tr>
<tr>
<td>PRN Functions</td>
<td>Generates the specified parameter as a string</td>
</tr>
<tr>
<td>S Function</td>
<td>Generates the &lt;S&gt; and &lt;/S&gt; tags which direct the browser to render the text they surround in strikethrough type</td>
</tr>
<tr>
<td>SAMPLE Function</td>
<td>Generates the &lt;SAMP&gt; and &lt;/SAMP&gt; tags which direct the browser to render the text they surround in monospace font or however &quot;sample&quot; is defined stylistically</td>
</tr>
<tr>
<td>SCRIPT Function</td>
<td>Generates the &lt;SCRIPT&gt; and &lt;/SCRIPT&gt; tags which contain a script written in languages such as JavaScript and VBscript</td>
</tr>
<tr>
<td>SMALL Function</td>
<td>Generates the &lt;SMALL&gt; and &lt;/SMALL&gt; tags, which direct the browser to render the text they surround using a small font</td>
</tr>
</tbody>
</table>
Table 215-1  (Cont.) HTF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRIKE Function</td>
<td>Generates the &lt;STRIKE&gt; and &lt;/STRIKE&gt; tags which direct the browser to render the text they surround in strikethrough type</td>
</tr>
<tr>
<td>STRONG Function</td>
<td>Generates the &lt;STRONG&gt; and &lt;/STRONG&gt; tags which direct the browser to render the text they surround in bold or however “strong” is defined stylistically</td>
</tr>
<tr>
<td>STYLE Function</td>
<td>Generates the &lt;STYLE&gt; and &lt;/STYLE&gt; tags which include a style sheet in a Web page</td>
</tr>
<tr>
<td>SUB Function</td>
<td>Generates the &lt;SUB&gt; and &lt;/SUB&gt; tags which direct the browser to render the text they surround as subscript</td>
</tr>
<tr>
<td>SUP Function</td>
<td>Generates the &lt;SUP&gt; and &lt;/SUP&gt; tags which direct the browser to render the text they surround as superscript</td>
</tr>
<tr>
<td>TABLECAPTION Function</td>
<td>Generates the &lt;CAPTION&gt; and &lt;/CAPTION&gt; tags which place a caption in an HTML table</td>
</tr>
<tr>
<td>TABLECLOSE Function</td>
<td>Generates the &lt;/TABLE&gt; tag which marks the end of an HTML table</td>
</tr>
<tr>
<td>TABLEDATA Function</td>
<td>Generates the &lt;TD&gt; and &lt;/TD&gt; tags which insert data into a cell of an HTML table</td>
</tr>
<tr>
<td>TABLEHEADER Function</td>
<td>Generates the &lt;TH&gt; and &lt;/TH&gt; tags which insert a header cell in an HTML table.</td>
</tr>
<tr>
<td>TABLEOPEN Function</td>
<td>Generates the &lt;TABLE&gt; tag which marks the beginning of an HTML table</td>
</tr>
<tr>
<td>TABLEROWCLOSE Function</td>
<td>Generates the &lt;/TR&gt; tag which marks the end of a new row in an HTML table</td>
</tr>
<tr>
<td>TABLEROWOPEN Function</td>
<td>Generates the &lt;TR&gt; tag which marks the beginning of a new row in an HTML table</td>
</tr>
<tr>
<td>TELETYPE Function</td>
<td>Generates the &lt;TT&gt; and &lt;/TT&gt; tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font</td>
</tr>
<tr>
<td>TITLE Function</td>
<td>Generates the &lt;TITLE&gt; and &lt;/TITLE&gt; tags which specify the text to display in the titlebar of the browser window</td>
</tr>
<tr>
<td>ULISTCLOSE Function</td>
<td>Generates the &lt;/UL&gt; tag which marks the end of an unordered list</td>
</tr>
<tr>
<td>ULISTOPEN Function</td>
<td>Generates the &lt;UL&gt; tag which marks the beginning of an unordered list</td>
</tr>
<tr>
<td>UNDERLINE Function</td>
<td>Generates the &lt;U&gt; and &lt;/U&gt; tags, which direct the browser to render the text they surround with an underline</td>
</tr>
<tr>
<td>VARIABLE Function</td>
<td>Generates the &lt;VAR&gt; and &lt;/VAR&gt; tags which direct the browser to render the text they surround in italics or however “variable” is defined stylistically.</td>
</tr>
<tr>
<td>WBR Function</td>
<td>Generates the &lt;WBR&gt; tag, which inserts a soft line break within a section of NOBR text</td>
</tr>
</tbody>
</table>
215.6.1 ADDRESS Function

This function generates the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document.

Syntax

```
HTF.ADDRESS (  
cvalue         IN       VARCHAR2  
cnowrap        IN       VARCHAR2   DEFAULT NULL  
cclear         IN       VARCHAR2   DEFAULT NULL  
cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

Table 215-2 ADDRESS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The string that goes between the &lt;ADDRESS&gt; and &lt;/ADDRESS&gt; tags.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>If the value for this parameter is not NULL, the NOWRAP attribute is included in the tag</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue</ADDRESS>
```

215.6.2 ANCHOR Function

Like the ANCHOR2 function, this function generates the <A> and </A> HTML tags which specify the source or destination of a hypertext link.

The difference between these subprograms is that the ANCHOR2 Function provides a target and therefore can be used for a frame.

Syntax

```
HTF.ANCHOR (  
curl           IN       VARCHAR2,  
ctext          IN       VARCHAR2,  
cname          IN       VARCHAR2   DEFAULT NULL,  
cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;
```
Parameters

Table 215-3 ANCHOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the HREF attribute.</td>
</tr>
<tr>
<td>ctext</td>
<td>The string that goes between the &lt;A&gt; and &lt;/A&gt; tags.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

Usage Notes

This tag accepts several attributes, but either HREF or NAME is required. HREF specifies to where to link. NAME allows this tag to be a target of a hypertext link.

215.6.3 ANCHOR2 Function

Like the ANCHOR function, this function generates the <A> and </A> HTML tags which specify the source or destination of a hypertext link.

The difference between this and the ANCHOR function is that this function provides a target and therefore can be used for a frame.

Syntax

```sql
HTF.ANCHOR2 (
    curl           IN       VARCHAR2,
    ctext          IN       VARCHAR2,
    cname          IN       VARCHAR2   DEFAULT NULL,
    ctarget        in       varchar2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-4 ANCHOR2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the HREF attribute.</td>
</tr>
<tr>
<td>ctext</td>
<td>The string that goes between the &lt;A&gt; and &lt;/A&gt; tags.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute</td>
</tr>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag</td>
</tr>
</tbody>
</table>

Chapter 215
Summary of HTF Subprograms
Examples

This function generates

```html
<A HREF="curl" NAME="cname" TARGET = "ctarget" cattributes>ctext</A>
```

### 215.6.4 APPLETCLOSE Function

This function closes the applet invocation with the `<APPLET>` tag.

You must first invoke the a Java applet using APPLETOPEN Function

**Syntax**

```sql
HTF.APPLETCLOSE
RETURN VARCHAR2;
```

### 215.6.5 APPLETOPEN Function

This function generates the `<APPLET>` tag which begins the invocation of a Java applet.

You close the applet invocation with APPLETCLOSE Function which generates the `</APPLET>` tag.

**Syntax**

```sql
HTF.APPLETOPEN (    ccode          IN       VARCHAR2,    cheight        IN       NUMBER,    cwidth         IN       NUMBER,    cattributes    IN       VARCHAR2   DEFAULT NULL)    RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccode</td>
<td>The value for the CODE attribute which specifies the name of the applet class.</td>
</tr>
<tr>
<td>cheight</td>
<td>The value for the HEIGHT attribute.</td>
</tr>
<tr>
<td>cwidth</td>
<td>The value for the WIDTH attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```html
<APPLET CODE=ccode HEIGHT=cheight WIDTH=cwidth cattributes>
```

so that, for example,

```
HTF.appletopen('testclass.class', 100, 200, 'CODEBASE="/ows-applets"')
```

generates
Usage Notes

• Specify parameters to the Java applet using the PARAM Function function.
• Use the cattributes parameter to specify the CODEBASE attribute since the PL/SQL cartridge does not know where to find the class files. The CODEBASE attribute specifies the virtual path containing the class files.

215.6.6 AREA Function

This function generates the <AREA> tag, which defines a client-side image map. The <AREA> tag defines areas within the image and destinations for the areas.

Syntax

```
HTF.AREA ( 
   ccoords        IN       VARCHAR2 ,
   cshape         IN       VARCHAR2 DEFAULT NULL ,
   chref          IN       VARCHAR2 DEFAULT NULL ,
   cnohref        IN       VARCHAR2 DEFAULT NULL ,
   ctarget        IN       VARCHAR2 DEFAULT NULL ,
   cattributes    IN       VARCHAR2 DEFAULT NULL )
RETURN VARCHAR2 ;
```

Parameters

Table 215-6 AREA Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccoords</td>
<td>The value for the COORDS attribute.</td>
</tr>
<tr>
<td>cshape</td>
<td>The value for the SHAPE attribute.</td>
</tr>
<tr>
<td>chref</td>
<td>The value for the HREF attribute.</td>
</tr>
<tr>
<td>cnohref</td>
<td>If the value for this parameter is not NULL, the NOHREF attribute is added to the tag.</td>
</tr>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget" cattributes>
```

215.6.7 BASE Function

This function generates the <BASE> tag which records the URL of the document.

Syntax

```
HTF.BASE ( 
   ctarget    IN       VARCHAR2 DEFAULT NULL ,
```

Properties

Table 215-7  BASE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute which establishes a window name to which all links in this document are targeted.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<BASE HREF="<current URL>" TARGET="ctarget" cattributes>
```

215.6.8 BASEFONT Function

This function generates the `<BASEFONT>` tag which specifies the base font size for a Web page.

Syntax

```
HTF.BASEFONT (    
    nsize        IN    INTEGER)    
RETURN VARCHAR2;
```

Parameters

Table 215-8  BASEFONT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsize</td>
<td>The value for the SIZE attribute.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<BASEFONT SIZE="nsize">
```

215.6.9 BGSOUND Function

This function generates the `<BGSOUND>` tag which includes audio for a Web page.

Syntax

```
HTF.BGSOUND (    
    csrc           IN       VARCHAR2,    
    cloop          IN       VARCHAR2   DEFAULT NULL,    
    cattributes    IN       VARCHAR2   DEFAULT NULL)    
RETURN VARCHAR2;
```
Parameters

**Table 215-9  BGSOUND Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>cloop</td>
<td>The value for the LOOP attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<BGSOUND SRC="csrc" LOOP="cloop" cattributes>
```

215.6.10 BIG Function

This function generates the `<BIG>` and `</BIG>` tags which direct the browser to render the text in a bigger font.

Syntax

```
HTF.BIG (  
   ctext          IN       VARCHAR2,  
   cattributes    IN       VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

**Table 215-10  BIG Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that goes between the tags.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<BIG cattributes>ctext</BIG>
```

215.6.11 BLOCKQUOTECLOSE Function

This function generates the `</BLOCKQUOTE>` tag which mark the end of a section of quoted text.

You mark the beginning of a section of text by means of the `BLOCKQUOTEOPEN Function`.

Syntax

```
HTF.BLOCKQUOTECLOSE  
RETURN VARCHAR2;
```
Examples

This function generates

</BLOCKQUOTE>

215.6.12 BLOCKQUOTEOPEN Function

This function generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text.

You mark the end of a section of text by means of the BLOCKQUOTECLOSE Function.

Syntax

HTF.BLOCKQUOTEOPEN (  
cnowrap        IN       VARCHAR2   DEFAULT NULL,  
cclear         IN       VARCHAR2   DEFAULT NULL,  
cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;

Parameters

Table 215-11  BLOCKQUOTEOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnowrap</td>
<td>If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>

215.6.13 BODYCLOSE Function

This function generates the </BODY> tag which marks the end of a body section of an HTML document.

You mark the beginning of a body section by means of the BODYOPEN Function.

Syntax

HTF.BODYCLOSE  
RETURN VARCHAR2;

Examples

This function generates

</BODY>
215.6.14 BODYOPEN Function

This function generates the `<BODY>` tag which marks the beginning of the body section of an HTML document.

You mark the end of a body section by means of the **BODYCLOSE Function**.

**Syntax**

```
HTF.BODYOPEN ( 
    cbackground    IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL) 
RETURN VARCHAR2;
```

**Parameters**

**Table 215-12  BODYOPEN Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbackground</td>
<td>The value for the <code>BACKGROUND</code> attribute which specifies a graphic file to use for the background of the document.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```
<BODY background="cbackground" cattributes>
```

so that

```
HTF.BODYOPEN('/img/background.gif') 
RETURN VARCHAR2;
```

generates:

```
<BODY background="/img/background.gif">
```

215.6.15 BOLD Function

This function generates the `<B>` and `</B>` tags which direct the browser to display the text in boldface.

**Syntax**

```
HTF.BOLD ( 
    ctext          IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL) 
RETURN VARCHAR2;
```
Parameters

Table 215-13  BOLD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that goes between the tags.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<B cattributes>ctext</B>
```

215.6.16 BR Function

This function generates the `<BR>` tag which begins a new line of text.

It performs the same operation as the NL Function.

Syntax

```
HTF.BR(
  cclear         IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-14  BR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<BR CLEAR="cclear" cattributes>
```

215.6.17 CENTER Function

This function generates the `<CENTER>` and `</CENTER>` tags which center a section of text within a Web page.

Syntax

```
HTF.CENTER (  
  ctext     IN       VARCHAR2)
RETURN VARCHAR2;
```
Parameters

Table 215-15   CENTER Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that goes between the tags.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```<CENTER>ctext</CENTER>```

215.6.18 CENTERCLOSE Function

This function generates the `<CENTER>` tag which marks the end of a section of text to center.

You mark the beginning of a section of text to center with the CENTEROPEN Function.

Syntax

```HTF.CENTERCLOSE
   RETURN VARCHAR2;```

Examples

This function generates

```</CENTER>```

215.6.19 CENTEROPEN Function

This function generates the `<CENTER>` tag which mark the beginning of a section of text to center.

You mark the beginning of a section of text to center with the CENTERCLOSE Function.

Syntax

```HTF.CENTEROPEN
   RETURN VARCHAR2;```

Examples

This function generates

```<CENTER>```
215.6.20 CITE Function

This function generates the `<CITE>` and `</CITE>` tags which direct the browser to render the text as a citation.

Syntax

```sql
HTF.CITE (  
    ctext          IN       VARCHAR2,  
    cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

Table 215-16  CITE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render as citation.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<CITE cattributes>ctext</CITE>
```

215.6.21 CODE Function

This function generates the `<CODE>` and `</CODE>` tags which direct the browser to render the text in monospace font or however “code” is defined stylistically.

Syntax

```sql
HTF.CODE (  
    ctext          IN       VARCHAR2,  
    cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

Table 215-17  CODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render as code.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<CODE cattributes>ctext</CODE>
```
215.6.22 COMMENT Function

This function generates the comment tags.

**Syntax**

```sql
HTF.COMMENT (     
    ctext          IN       VARCHAR2) 
RETURN VARCHAR2;
```

**Parameters**

**Table 215-18  COMMENT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The comment.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```html
<!-- ctext -->
```

215.6.23 DFN Function

This function generates the `<DFN>` and `</DFN>` tags which direct the browser to mark the text in italics or however “definition” is described stylistically.

**Syntax**

```sql
HTF.DFN (     
    ctext          IN       VARCHAR2) 
RETURN VARCHAR2;
```

**Parameters**

**Table 215-19  DFN Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in italics.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```html
<DFN>ctext</DFN>
```
215.6.24 DIRLISTCLOSE Function

This function generates the </DIR> tag which ends a directory list section. You start a directory list section with the DIRLISTOPEN Function.

Syntax

```
HTF.DIROLISTCLOSE
   RETURN VARCHAR2;
```

Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Function so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This function generates

</DIR>

Related Topics

- **DIRLISTOPEN Function**
  This function generates the <DIR> which starts a directory list section. You end a directory list section with the DIRLISTCLOSE Function.

215.6.25 DIRLISTOPEN Function

This function generates the <DIR> which starts a directory list section. You end a directory list section with the DIRLISTCLOSE Function.

Syntax

```
HTF.DIROLISTOPEN
   RETURN VARCHAR2;
```

Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Function so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This function generates

<DIR>

Related Topics

- **DIRLISTCLOSE Function**
  This function generates the </DIR> tag which ends a directory list section. You start a directory list section with the DIRLISTOPEN Function.
215.6.26 DIV Function

This function generates the &lt;DIV&gt; tag which creates document divisions.

Syntax

```
HTF.DIV (calign IN VARCHAR2 DEFAULT NULL, cattributes IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-20 DIV Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
&lt;DIV ALIGN="calign" cattributes&gt;
```

215.6.27 DLISTCLOSE Function

This function generates the &lt;/DL&gt; tag which ends a definition list. You start a definition list by means of the DLISTOPEN Function.

Syntax

```
HTF.DLISTCLOSE
RETURN VARCHAR2;
```

Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Function and definitions are inserted using the DLISTDEF Function.

Examples

This function generates

```
&lt;/DL&gt;
```

Related Topics

- DLISTOPEN Function
  This function generates the &lt;DL&gt; tag which starts a definition list. You end a definition list by means of the DLISTCLOSE Function.
215.6.28 DLISTDEF Function

This function generates the `<DD>` tag, which inserts definitions of terms. Use this tag for a definition list `<DL>`. Terms are tagged `<DT>` and definitions are tagged `<DD>`.

Syntax

```
HTF.DLISTDEF (  
    ctext          IN       VARCHAR2   DEFAULT NULL,  
    cclear         IN       VARCHAR2   DEFAULT NULL,  
    cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The definition of the term.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<DD CLEAR="cclear" cattributes>ctext
```

215.6.29 DLISTOPEN Function

This function generates the `<DL>` tag which starts a definition list. You end a definition list by means of the DLISTCLOSE Function.

Syntax

```
HTF.DLISTOPEN (  
    cclear         IN       VARCHAR2   DEFAULT NULL,  
    cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Function and definitions are inserted using the DLISTDEF Function.

Examples

This function generates

<DL CLEAR="cclear" cattributes>

Related Topics

- DLISTCLOSE Function
  This function generates the </DL> tag which ends a definition list. You start a definition list by means of the DLISTOPEN Function.

215.6.30 DLISTTERM Function

This function generates the <DT> tag which defines a term in a definition list <DL>.

Syntax

HTF.DLISTTERM (  
ctext          IN       VARCHAR2   DEFAULT NULL,  
cclear         IN       VARCHAR2   DEFAULT NULL,  
cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;

Parameters

Table 215-23  DLISTTERM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The term.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<DT CLEAR="cclear" cattributes>ctext

215.6.31 EM Function

This function generates the <EM> and </EM> tags, which define text to be emphasized.

It performs the same task as the EMPHASIS Function.

Syntax

HTF.EM(  
ctext          IN       VARCHAR2,  
cclear         IN       VARCHAR2   DEFAULT NULL,  
cattributes    IN       VARCHAR2   DEFAULT NULL)  
RETURN VARCHAR2;
215.6.32 EMPHASIS Function

This function generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the EM Function.

Syntax

HTF.EMPHASIS(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;

Parameters

Table 215-25  EMPHASIS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to emphasize.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<EM cattributes>ctext</EM>

215.6.33 ESCAPE_SC Function

This function replaces characters that have special meaning in HTML with their escape sequences.

The following characters are converted:

- & to &amp;
- " to &quot;
• < to &lt;
• > to &gt;

This function performs the same operation as HTP. PRINTS Procedure and HTP. PS Procedure.

Syntax

HTF.ESCAPE_SC(
    ctext          IN       VARCHAR2);

Parameters

Table 215-26  ESCAPE_SC Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text string to convert.</td>
</tr>
</tbody>
</table>

215.6.34 ESCAPE_URL Function

This deprecated function replaces characters that have special meaning in HTML and HTTP with their escape sequences.

Note:

This procedure, deprecated in Release 10g, and provided here only for reasons of backward compatibility, does not comply with the Internet Engineering Task Force (IETF) Request for Comments (RFC) standards of URL encoding. If you need to encode URLs, it is recommended you use the ESCAPE Function in the UTL_URL package.

The following characters are converted:

• & to &amp;
• " to &quot;
• < to &lt;
• > to &gt;
• % to &25

Syntax

HTF.ESCAPE_URL(
    p_url          IN       VARCHAR2);
Parameters

Table 215-27 ESCAPE_URL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>The string to convert.</td>
</tr>
</tbody>
</table>

215.6.35 FONTCLOSE Function

This function generates the </FONT> tag which marks the end of a section of text with the specified font characteristics.

You mark the beginning of the section text by means of the FONTOPEN Function.

Syntax

```sql
HTF.FONTCLOSE
  RETURN VARCHAR2;
```

Examples

This function generates

</FONT>

215.6.36 FONTOPEN Function

This function generates the <FONT> which marks the beginning of section of text with the specified font characteristics.

You mark the end of the section text by means of the FONTCLOSE Function.

Syntax

```sql
HTF.FONTOPEN(
  ccolor         IN       VARCHAR2   DEFAULT NULL,
  cface          IN       VARCHAR2   DEFAULT NULL,
  csize          IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 215-28 FONTOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccolor</td>
<td>The value for the COLOR attribute.</td>
</tr>
<tr>
<td>cface</td>
<td>The value for the FACE attribute</td>
</tr>
<tr>
<td>csize</td>
<td>The value for the SIZE attribute</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples

This function generates

<font color="ccolor" face="cface" size="csize" cattributes>

215.6.37 FORMAT_CELL Function

This function formats column values inside an HTML table using the TABLEDATA Function. It allows for better control over the HTML tables.

Syntax

HTF.FORMAT_CELL(
    columnValue          IN       VARCHAR2
    format_numbers       IN       VARCHAR2   DEFAULT NULL
)
RETURN VARCHAR2;

Parameters

Table 215-29  FORMAT_CELL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnValue</td>
<td>The value that needs to be formatted in an HTML table.</td>
</tr>
<tr>
<td>format_numbers</td>
<td>The format that numeric data is displayed in. If the value of this parameter is not NULL, the number fields are right-justified and rounded to two decimal places.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<TD >columnValue</TD>

Related Topics

- TABLEDATA Function
  This function generates the <TD> and </TD> tags which insert data into a cell of an HTML table.

215.6.38 FORMCHECKBOX Function

This function generates the <INPUT> tag with TYPE="checkbox" which inserts a checkbox element in a form.

A checkbox element is a button that the user toggles on or off.

Syntax

HTF.FORMCHECKBOX(
    cname          IN       VARCHAR2,
    cvalue         IN       VARCHAR2   DEFAULT 'ON',
    cchecked       IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL
)
RETURN VARCHAR2;
Parameters

Table 215-30  FORMCHECKBOX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cchecked</td>
<td>If the value for this parameter is not NULL, the CHECKED attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

215.6.39 FORMCLOSE Function

This function generates the &lt;/FORM&gt; tag which marks the end of a form section in an HTML document.

You mark the beginning of the form section by means of the FORMOPEN Function.

Syntax

```
HTF.FORMCLOSE
RETURN VARCHAR2;
```

Examples

This function generates

```
</FORM>
```

215.6.40 FORMFILE Function

This function generates the &lt;INPUT&gt; tag with TYPE="file" which inserts a file form element. This is used for file uploading for a given page.

Syntax

```
HTF.FORMFILE
  cnam e IN VARCHAR2,
  caccept IN VARCHAR2 DEFAULT NULL,
  cattributes IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```
Parameters

Table 215-31 FORMFILE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>caccept</td>
<td>A comma-delimited list of MIME types for upload.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<INPUT TYPE="file" NAME="cname" ACCEPT="caccept" cattributes>
```

215.6.41 FORMHIDDEN Function

This function generates the `<INPUT>` tag with TYPE="hidden", which inserts a hidden form element.

This element is not seen by the user. It submits additional values to the script.

Syntax

```
HTF.FORMHIDDEN(
    cname          IN       VARCHAR2,
    cvalue         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-32 FORMHIDDEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

215.6.42 FORMIMAGE Function

This function generates the `<INPUT>` tag with TYPE="image" which creates an image field that the user clicks to submit the form immediately.

The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/value pairs. The x coordinate is submitted under the name
of the field with .x appended, and the y coordinate with .y appended. Any VALUE
attribute is ignored.

Syntax

HTF.FORMIMAGE(
    cname          IN       VARCHAR2,
    csrc           IN       VARCHAR2,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;

Parameters

Table 215-33  FORMIMAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute that specifies the image file.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

215.6.43 FORMOPEN Function

This function generates the `<FORM>` tag which marks the beginning of a form section in
an HTML document.

You mark the end of the form section by means of the FORMCLOSE Function.

Syntax

HTF.FORMOPEN(
    curl           IN       VARCHAR2,
    cmethod        IN       VARCHAR2   DEFAULT 'POST',
    ctarget        IN       VARCHAR2   DEFAULT NULL,
    cencctype       IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;

Parameters

Table 215-34  FORMOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| curl           | The URL of the Web Request Broker or CGI script where the contents of
                | the form is sent. This parameter is required.                |
| cmethod        | The value for the METHOD attribute. The value can be "GET" or "POST".
                |
Table 215-34  (Cont.) FORMOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute.</td>
</tr>
<tr>
<td>c enctype</td>
<td>The value for the ENCTYPE attribute.</td>
</tr>
<tr>
<td>c attributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarget" ENCTYPE="c enctype" c attributes>
```

215.6.44 FORMPASSWORD Function

This function generates the `<INPUT>` tag with `TYPE="password"` which creates a single-line text entry field.

When the user enters text in the field, each character is represented by one asterisk. This is used for entering passwords.

Syntax

```sql
HTF.FORMPASSWORD(
  cname          IN       VARCHAR2,
  csize          IN       VARCHAR2,
  cmaxlength     IN       VARCHAR2 DEFAULT NULL,
  cvalue         IN       VARCHAR2 DEFAULT NULL,
  c attributes    IN       VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-35  FORMPASSWORD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>csize</td>
<td>The value for the SIZE attribute.</td>
</tr>
<tr>
<td>cmaxlength</td>
<td>The value for the MAXLENGTH attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>c attributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength" VALUE="cvalue" c attributes>
```
215.6.45 FORMRADIO Function

This function generates the `<INPUT>` tag with `TYPE="radio"`, which creates a radio button on the HTML form. Within a set of radio buttons, the user selects only one. Each radio button in the same set has the same name, but different values. The selected radio button generates a name/value pair.

**Syntax**

```sql
HTF.FORMRADIO(
    cname          IN       VARCHAR2,
    cvalue         IN       VARCHAR2,
    cchecked       IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

**Table 215-36  FORMRADIO Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the <code>NAME</code> attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the <code>VALUE</code> attribute.</td>
</tr>
<tr>
<td>cchecked</td>
<td>If the value for this parameter is not <code>NULL</code>, the <code>CHECKED</code> attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```html
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

215.6.46 FORMRESET Function

This function generates the `<INPUT>` tag with `TYPE="reset"` which creates a button that, when selected, resets the form fields to their initial values.

**Syntax**

```sql
HTF.FORMRESET(
    cvalue         IN       VARCHAR2   DEFAULT 'Reset',
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

**Table 215-37  FORMRESET Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The value for the <code>VALUE</code> attribute.</td>
</tr>
</tbody>
</table>
Table 215-37  (Cont.) FORMRESET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

215.6.47 FORMSELECTCLOSE Function

This function generates the `</SELECT>` tag which marks the end of a Select form element.

A Select form element is a listbox where the user selects one or more values. You mark the beginning of Select form element by means of the FORMSELECTOPEN Function. The values are inserted using FORMSELECTOPTION Function.

Syntax

```sql
HTF.FORMSELECTCLOSE
    RETURN VARCHAR2;
```

Examples

This function generates

```html
</SELECT>
```

as shown under Examples of the FORMSELECTOPEN Function.

215.6.48 FORMSELECTOPEN Function

This function generates the `<SELECT>` tags which begins a Select form element.

A Select form element is a listbox where the user selects one or more values. You mark the end of Select form element by means of the FORMSELECTCLOSE Function. The values are inserted using FORMSELECTOPTION Function.

Syntax

```sql
HTF.FORMSELECTOPEN(
    cname        IN    VARCHAR2,
    cprompt      IN    VARCHAR2   DEFAULT NULL,
    nsize        IN    INTEGER    DEFAULT NULL,
    cattributes  IN    VARCHAR2   DEFAULT NULL)
    RETURN VARCHAR2;
```
Parameters

Table 215-38  FORMSELECTOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cprompt</td>
<td>The string preceding the list box.</td>
</tr>
<tr>
<td>nsize</td>
<td>The value for the SIZE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```sql
cprompt <SELECT NAME="cname" SIZE="nsize" cattributes>
</SELECT>
```

so that

```sql
HTF.FORMSELECTOPEN('greatest_player');
    'Pick the greatest player:');
HTF.FORMSELECTOPTION('Messier');
HTF.FORMSELECTOPTION('Howe');
HTF.FORMSELECTOPTION('Gretzky');.
HTF.FORMSELECTCLOSE;
```

generates

```html
Pick the greatest player:
<SELECT NAME="greatest_player">
<OPTION>Messier
<OPTION>Howe
<OPTION>Gretzky
</SELECT>
```

215.6.49 FORMSELECTOPTION Function

This function generates the `<OPTION>` tag which represents one choice in a Select element.

Syntax

```sql
HTF.FORMSELECTOPTION(
    cvalue IN VARCHAR2,
    cselected IN VARCHAR2 DEFAULT NULL,
    cattributes IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```
Parameters

Table 215-39 FORMSELECTOPTION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The text for the option.</td>
</tr>
</tbody>
</table>

- If the value for this parameter is not NULL, the `SELECTED` attribute is added to the tag.

| cattributes | The other attributes to be included as-is in the tag. |

Examples

This function generates `<OPTION SELECTED cattributes>cvalue` as shown under the Examples section of the FORMSELECTOPEN Function.

215.6.50 FORMSUBMIT Function

This function generates the `<INPUT>` tag with `TYPE="submit"` which creates a button that, when clicked, submits the form. If the button has a `NAME` attribute, the button contributes a name/value pair to the submitted data.

Syntax

```
HTF.FORMSUBMIT(
    cname          IN       VARCHAR2   DEFAULT NULL,
    cvalue         IN       VARCHAR2   DEFAULT 'Submit',
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-40 FORMSUBMIT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the <code>NAME</code> attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the <code>VALUE</code> attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```
215.6.51 FORMTEXT Function

This function generates the `<INPUT>` tag with `TYPE="text"`, which creates a field for a single line of text.

**Syntax**

```
HTF.FORMTEXT(
    cname          IN       VARCHAR2,
    csize          IN       VARCHAR2   DEFAULT NULL,
    cmaxlength     IN       VARCHAR2   DEFAULT NULL,
    cvalue         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

**Table 215-41  FORMTEXT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>csize</td>
<td>The value for the SIZE attribute.</td>
</tr>
<tr>
<td>cmaxlength</td>
<td>The value for the MAXLENGTH attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

215.6.52 FORMTEXTAREA Function

This function generates the `<TEXTAREA>` tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text.

The same operation is performed by the FORMTEXTAREA2 Function which in addition has the `cwrap` parameter that lets you specify a wrap style.

**Syntax**

```
HTF.FORMTEXTAREA(
    cname          IN       VARCHAR2,
    nrows          IN       INTEGER,
    ncolumns       IN       INTEGER,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```
Parameters

### Table 215-42 FORMTEXTAREA Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the <strong>NAME</strong> attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the <strong>ROWS</strong> attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the <strong>COLS</strong> attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the <strong>ALIGN</strong> attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes>
```

### 215.6.53 FORMTEXTAREA2 Function

This function generates the `<TEXTAREA>` tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text.

The same operation is performed by the FORMTEXTAREA Function except that in that case you cannot specify a wrap style.

**Syntax**

```
HTF.FORMTEXTAREA2(
    cname          IN       VARCHAR2,
    nrows          IN       INTEGER,
    ncolumns       IN       INTEGER,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cwrap          IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

### Table 215-43 FORMTEXTAREA2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the <strong>NAME</strong> attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the <strong>ROWS</strong> attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the <strong>COLS</strong> attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the <strong>ALIGN</strong> attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the <strong>WRAP</strong> attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples

This function generates

\[
\text{<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP="cwrap" cattributes>\text{\ldots}<TEXTAREA>}
\]

215.6.54 FORMTEXTAREACLOSE Function

This function generates the \text{\ldots}<TEXTAREA> tag which ends a text area form element.

You open a text area element by means of either FORMTEXTAREAOPEN Function or FORMTEXTAREAOPEN2 Function.

Syntax

\[
\text{HTF.FORMTEXTAREACLOSE (RETURN VARCHAR2;)}
\]

Examples

This function generates

\[
\text{\ldots}<\text{TEXTAREA>}
\]

215.6.55 FORMTEXTAREAOPEN Function

This function generates the \text{\ldots}<TEXTAREA> which marks the beginning of a text area form element.

The same operation is performed by the FORMTEXTAREAOPEN2 Function which in addition has the \text{\ldots}cwrap parameter that lets you specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Function.

Syntax

\[
\text{HTF.FORMTEXTAREAOPEN (cname \quad IN \quad VARCHAR2,}
\text{\quad nrows \quad IN \quad INTEGER,}
\text{\quad ncolumns \quad IN \quad INTEGER,}
\text{\quad calign \quad IN \quad VARCHAR2 \quad DEFAULT NULL,}
\text{\quad cattributes \quad IN \quad VARCHAR2 \quad DEFAULT NULL)}
\text{RETURN VARCHAR2;}
\]

Parameters

Table 215-44 FORMTEXTAREAOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the \text{\ldots}NAME attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the \text{\ldots}ROWS attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the \text{\ldots}COLS attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the \text{\ldots}ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples

This function generates

\[<\text{TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes}>\]

215.6.56 FORMTEXTAREAOPEN2 Function

This function generates the `TEXTAREA` which marks the beginning of a text area form element.

The same operation is performed by the FORMTEXTAREAOPEN Function except that in that case you cannot specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Function.

Syntax

```
HTF.FORMTEXTAREAOPEN2(
    cname          IN       VARCHAR2,
    nrows          IN       INTEGER,
    ncolumns       IN       INTEGER,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cwrap          IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-45 FORMTEXTAREAOPEN2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the ROWS attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the COLS attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the WRAP attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

\[<\text{TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP = "cwrap" cattributes}>\]
215.6.57 FRAME Function

This function generates the `<FRAME>` tag which begins the characteristics of a frame created by a `<FRAMESET>` tag.

Syntax

```sql
HTF.FRAME(
  csrc           IN       VARCHAR2,
  cname          IN       VARCHAR2   DEFAULT NULL,
  cmarginwidth   IN       VARCHAR2   DEFAULT NULL,
  cmarginheight  IN       VARCHAR2   DEFAULT NULL,
  cscrolling     IN       VARCHAR2   DEFAULT NULL,
  cnoresize      IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-46  FRAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csrc</td>
<td>The URL to display in the frame.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cmarginwidth</td>
<td>The value for the MARGINWIDTH attribute.</td>
</tr>
<tr>
<td>cmarginheight</td>
<td>The value for the MARGINHEIGHT attribute.</td>
</tr>
<tr>
<td>cscrolling</td>
<td>The value for the SCROLLING attribute.</td>
</tr>
<tr>
<td>cnoresize</td>
<td>If the value for this parameter is not NULL, the NORESIZE attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<FRA ME SRC="csrc" NAME="cname" MARGINWIDTH="cmarginwidth" MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE cattributes>
```

215.6.58 FRAMESETCLOSE Function

This function generates the `</FRAMESET>` tag which ends a frameset section.

You mark the beginning of a frameset section by means of the FRAMESETOPEN Function.

Syntax

```sql
HTF.FRAMESETCLOSE
RETURN VARCHAR2;
```

Examples

This function generates

`</FRAMESET>`
215.6.59 FRAMESETOPEN Function

This function generates the `<FRAMESET>` tag which define a frameset section.

You mark the end of a frameset section by means of the FRAMESETCLOSE Function.

Syntax

```sql
HTF.FRAMESETOPEN(
    crows          IN       VARCHAR2   DEFAULT NULL,
    ccols          IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-47  FRAMESETOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crows</td>
<td>The value for the ROWS attribute.</td>
</tr>
<tr>
<td>ccols</td>
<td>The value for the COLS attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

215.6.60 HEADCLOSE Function

This function generates the `</HEAD>` tag which marks the end of an HTML document head section.

You mark the beginning of an HTML document head section by means of the HEADOPEN Function.

Syntax

```sql
HTF.HEADCLOSE
RETURN VARCHAR2;
```

Examples

This function generates

```html
</HEAD>
```
215.6.61 HEADER Function

This function generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (<H1> to </H6>).

Syntax

```
HTF.HEADER(
    nsize          IN       INTEGER,
    cheader        IN       VARCHAR2,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cnowrap        IN       VARCHAR2   DEFAULT NULL,
    cclear         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-48  HEADER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsize</td>
<td>The heading level. This is an integer between 1 and 6.</td>
</tr>
<tr>
<td>cheader</td>
<td>The text to display in the heading.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>The value for the NOWRAP attribute.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

```
HTF.header (1,'Overview')
RETURN VARCHAR2;
```

produces:

```
<H1>Overview</H1>
```

215.6.62 HEADOPEN Function

This function generates the <HEAD> tag which marks the beginning of the HTML document head section.

You mark the end of an HTML document head section by means of the HEADCLOSE Function.

Syntax

```
HTF.HEADOPEN
RETURN VARCHAR2;
```

Examples

This function generates
215.6.63 HR Function

This function generates the `<hr>` tag, which generates a line in the HTML document. This subprogram performs the same operation as the LINE Function.

**Syntax**

```sql
HTF.HR(
    cclear         IN       VARCHAR2   DEFAULT NULL,
    csrc           IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute which specifies a custom image as the source of the line.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```html
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

215.6.64 HTMLCLOSE Function

This function generates the `</html>` tag which marks the end of an HTML document. You use the HTMLOPEN Function to mark the beginning of an HTML document.

**Syntax**

```sql
HTF.HTMLCLOSE
RETURN VARCHAR2;
```

**Examples**

This function generates

```html
</HTML>
```

215.6.65 HTMLOPEN Function

This function generates the `<html>` tag which marks the beginning of an HTML document. You use the HTMLCLOSE Function to mark the end of the an HTML document.
Syntax

```sql
HTF.HTMLOPEN
   RETURN VARCHAR2;
```

Examples

This function generates

```html
<HTML>
```

215.6.66 IMG Function

This function generates the `<IMG>` tag which directs the browser to load an image onto the HTML page.

The **IMG2 Function** performs the same operation but additionally uses the `cismap` parameter.

Syntax

```sql
HTF.IMG(
  curl           IN       VARCHAR2   DEFAULT NULL,
  calign         IN       VARCHAR2   DEFAULT NULL,
  calt           IN       VARCHAR2   DEFAULT NULL,
  cismap         IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
   RETURN VARCHAR2;
```

Parameters

**Table 215-50  IMG Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>calt</td>
<td>The value for the ALT attribute which specifies alternative text to display if the browser does not support images.</td>
</tr>
<tr>
<td>cismap</td>
<td>If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```
215.6.67 IMG2 Function

This function generates the `<IMG>` tag, which directs the browser to load an image onto the HTML page.

The **IMG Function** performs the same operation but does not use the `cusemap` parameter.

**Syntax**

```sql
HTF.IMG2(
curl           IN       VARCHAR2   DEFAULT NULL,
calign         IN       VARCHAR2   DEFAULT NULL,
calt           IN       VARCHAR2   DEFAULT NULL,
cismap         IN       VARCHAR2   DEFAULT NULL,
cusemap        IN       VARCHAR2   DEFAULT NULL,
cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>calt</td>
<td>The value for the ALT attribute which specifies alternative text to display if the browser does not support images.</td>
</tr>
<tr>
<td>cismap</td>
<td>If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.</td>
</tr>
<tr>
<td>cusemap</td>
<td>The value for the USEMAP attribute which specifies a client-side image map.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```html
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap" cattributes>
```

215.6.68 ISINDEX Function

This function creates a single entry field with a prompting text, such as "enter value," then sends that value to the URL of the page or program.

**Syntax**

```sql
HTF.ISINDEX(
cprompt        IN       VARCHAR2    DEFAULT NULL,
curl           IN       VARCHAR2    DEFAULT NULL)
RETURN VARCHAR2;
```
Parameters

Table 215-52  ISINDEX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cprompt</td>
<td>The value for the PROMPT attribute.</td>
</tr>
<tr>
<td>curl</td>
<td>The value for the HREF attribute.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<ISINDEX PROMPT="cprompt" HREF="curl">
```

215.6.69 ITALIC Function

This function generates the `<I>` and `</I>` tags which direct the browser to render the text in italics.

Syntax

```sql
HTF.ITALIC(
    ctext          IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-53  ITALIC Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in italics.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<I cattributes>ctext</I>
```

215.6.70 KBD Function

This function generates the `<KBD>` and `</KBD>` tags which direct the browser to render the text in monospace font.

This subprogram performs the same operation as the KEYBOARD Function.

Syntax

```sql
HTF.KBD(
    ctext          IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```
Parameters

Table 215-54  KBD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in monospace.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<KBD cattributes>ctext</KBD>

215.6.71 KEYBOARD Function

This function generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font.

This subprogram performs the same operation as the KBD Function.

Syntax

HTF.KEYBOARD(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;

Parameters

Table 215-55  KEYBOARD Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in monospace.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<KBD cattributes>ctext</KBD>

215.6.72 LINE Function

This function generates the <HR> tag, which generates a line in the HTML document.

This subprogram performs the same operation as the HR Function.

Syntax

HTF.LINE(
  cclear         IN       VARCHAR2   DEFAULT NULL,
  csrc           IN       VARCHAR2   DEFAULT NULL,  
RETURN VARCHAR2;
```sql
SELECT
  attributes
FROM
  VARCHAR2
  DEFAULT NULL);
RETURN VARCHAR2;

Parameters

Table 215-56  LINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>csrd</td>
<td>The value for the SRC attribute which specifies a custom image as the source of the line.</td>
</tr>
<tr>
<td>cattribut</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<HR CLEAR="cclear" SRC="csrd" cattribut>
```

215.6.73 LINKREL Function

This function generates the `<LINK>` tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present.

This is the opposite of LINKREV Function. This tag indicates a relationship between documents but does not create a link. To create a link, use the ANCHOR Function.

Syntax

```sql
HTF.LINKREL(
  crel          IN       VARCHAR2,
  curl          IN       VARCHAR2,
  ctitle        IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;

Parameters

Table 215-57  LINKREL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crel</td>
<td>The value for the REL attribute.</td>
</tr>
<tr>
<td>curl</td>
<td>The value for the URL attribute.</td>
</tr>
<tr>
<td>ctitle</td>
<td>The value for the TITLE attribute.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<Link REL="crel" HREF="curl" TITLE="ctitle">
```
215.6.74 LINKREV Function

This function generates the `<LINK>` tag with the `REV` attribute which delineates the relationship described by the hypertext link from the target to the anchor.

This is the opposite of the LINKREL Function. This tag indicates a relationship between documents, but does not create a link. To create a link, use the ANCHOR Function.

Syntax

```
HTF.LINKREV(
    crev           IN       VARCHAR2,
    curl           IN       VARCHAR2,
    ctitle         IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-58   LINKREV Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crev</td>
<td>The value for the <code>REV</code> attribute.</td>
</tr>
<tr>
<td>curl</td>
<td>The value for the <code>URL</code> attribute.</td>
</tr>
<tr>
<td>ctitle</td>
<td>The value for the <code>TITLE</code> attribute.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<Link REV="crev" HREF="curl" TITLE="ctitle">
```

215.6.75 LISTHEADER Function

This function generates the `<LH>` and `</LH>` tags which print an HTML tag at the beginning of the list.

Syntax

```
HTF.LISTHEADER(
    ctext          IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-59   LISTHEADER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to place between <code>&lt;LH&gt;</code> and <code>&lt;/LH&gt;</code>.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples
This function generates
</LH cattributes>ctext</LH>

215.6.76 LISTINGCLOSE Function

This function generates the </LISTING> tags which marks the end of a section of fixed-width text in the body of an HTML page.

To mark the beginning of a section of fixed-width text in the body of an HTML page, use the LISTINGOPEN Function.

Syntax

```
HTF.LISTINGCLOSE
   RETURN VARCHAR2;
```

Examples
This function generates
</LISTING>

215.6.77 LISTINGOPEN Function

This function generates the <LISTING> tag which marks the beginning of a section of fixed-width text in the body of an HTML page.

To mark the end of a section of fixed-width text in the body of an HTML page, use the LISTINGCLOSE Function.

Syntax

```
HTF.LISTINGOPEN
   RETURN VARCHAR2;
```

Examples
This function generates
<LISTING>

215.6.78 LISTITEM Function

This function generates the <LI> tag, which indicates a list item.

Syntax

```
HTF.LISTITEM(
   ctext          IN       VARCHAR2   DEFAULT NULL,
   cclear         IN       VARCHAR2   DEFAULT NULL,
   cdingbat       IN       VARCHAR2   DEFAULT NULL,
   csrc           IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
   RETURN VARCHAR2;
```

Parameters

Table 215-60  LISTITEM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text for the list item.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cdingbat</td>
<td>The value for the DINGBAT attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

\(<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext\>

215.6.79 MAILTO Function

This function generates the \(<A>\) tag with the HREF set to 'mailto' prepended to the mail address argument.

Syntax

```
HTF.MAILTO(
    caddress       IN       VARCHAR2,
    ctext          IN       VARCHAR2,
    cname          IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-61  MAILTO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caddress</td>
<td>The email address of the recipient.</td>
</tr>
<tr>
<td>ctext</td>
<td>The clickable portion of the link.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

\(<A HREF="mailto:caddress" NAME="cname" cattributes>ctext</A>\>

so that

HTF.mailto('pres@white_house.gov','Send Email to the President');
This function generates the `</MAP>` tag which marks the end of a set of regions in a client-side image map. To mark the beginning of a set of regions in a client-side image map, use the MAPOPEN Function.

Syntax

```
HTF.MAPCLOSE
    RETURN VARCHAR2;
```

Examples

This function generates

```
</MAP>
```

### 215.6.81 MAPOPEN Function

This function generates the `<MAP>` tag which mark the beginning of a set of regions in a client-side image map. To mark the end of a set of regions in a client-side image map, use the MAPCLOSE Function.

Syntax

```
HTF.MAPOPEN(
    cname          IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
    RETURN VARCHAR2;
```

**Parameters**

**Table 215-62  MAPOPEN Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<MAP NAME="cname" cattributes>
```
215.6.82 MENULISTCLOSE Function

This function generates the `<MENU>` tag which ends a list that presents one line for each item.

To begin a list of this kind, use the MENULISTOPEN Function. The items in the list appear more compact than an unordered list. The LISTITEM Function defines the list items in a menu list.

**Syntax**

```sql
HTF.MENULISTCLOSE
    RETURN VARCHAR2;
```

**Examples**

This function generates

`</MENU>`

215.6.83 MENULISTOPEN Function

This function generates the `<MENU>` tag which begins a list that presents one line for each item.

To end a list of this kind, use the MENULISTCLOSE Function. The items in the list appear more compact than an unordered list. The LISTITEM Function defines the list items in a menu list.

**Syntax**

```sql
HTF.MENULISTOPEN
    RETURN VARCHAR2;
```

**Examples**

This function generates

`<MENU>`

215.6.84 META Function

This function generates the `<META>` tag, which embeds meta-information about the document and also specifies values for HTTP headers. For example, you can specify the expiration date, keywords, and author name.

**Syntax**

```sql
HTF.META(
    chttp_equiv    IN       VARCHAR2,
    cname          IN       VARCHAR2,
    ccontent       IN       VARCHAR2)
    RETURN VARCHAR2;
```
Parameters

Table 215-63  META Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chttp_equiv</td>
<td>The value for the CHTTP_EQUIV attribute.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>ccontent</td>
<td>The value for the CONTENT attribute.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<META HTTP-EQUIV="chttp_equiv" NAME="cname" CONTENT="ccontent">
```

so that

```
HTF.meta ('Refresh', NULL, 120);
```

generates

```
<META HTTP-EQUIV="Refresh" CONTENT=120>
```

On some Web browsers, this causes the current URL to be reloaded automatically every 120 seconds.

215.6.85 NL Function

This function generates the `<BR>` tag which begins a new line of text.

It performs the same operation as the BR Function.

Syntax

```
HTF.NL(
    cclear         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-64  NL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<BR CLEAR="cclear" cattributes>
```
215.6.86 NOBR Function

This function generates the `<NOBR>` and `</NOBR>` tags which turn off line-breaking in a section of text.

**Syntax**

```sql
HTF.NOBR(
    ctext       IN       VARCHAR2
) RETURN VARCHAR2;
```

**Parameters**

**Table 215-65  NOBR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that is to be rendered on one line.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```
<NOBR>ctext</NOBR>
```

215.6.87 NOFRAMESCLOSE Function

This function generates the `</NOFRAMES>` tag which marks the end of a no-frames section.

To mark the beginning of a no-frames section, use the FRAMESETOPEN Function. See also FRAME Function, FRAMESETOPEN Function and FRAMESETCLOSE Function.

**Syntax**

```sql
HTF.NOFRAMESCLOSE
RETURN VARCHAR2;
```

**Examples**

This function generates

```
</NOFRAMES>
```

215.6.88 NOFRAMESOPEN Function

This function generates the `<NOFRAMES>` tag which mark the beginning of a no-frames section.

To mark the end of a no-frames section, use the FRAMESETCLOSE Function. See also FRAME Function, FRAMESETOPEN Function and FRAMESETCLOSE Function.

**Syntax**

```sql
HTF.NOFRAMESOPEN
RETURN VARCHAR2;
```
Examples
This function generates

<NOFRAMES>

215.6.89 OLISTCLOSE Function

This function generates the </ol> tag which defines the end of an ordered list. An ordered list presents a list of numbered items.

To mark the beginning of a list of this kind, use the OLISTOPEN Function. Numbered items are added using LISTITEM Function.

Syntax
HTF.OLISTCLOSE
   RETURN VARCHAR2;

Examples
This function generates
</ol>

215.6.90 OLISTOPEN Function

This function generates the <ol> tag which marks the beginning of an ordered list. An ordered list presents a list of numbered items.

To mark the end of a list of this kind, use the OLISTCLOSE Function. Numbered items are added using LISTITEM Function.

Syntax
HTF.OLISTOPEN(
   cclear IN VARCHAR2 DEFAULT NULL,
   cwrap  IN VARCHAR2 DEFAULT NULL,
   cattributes IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the WRAP attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples
This function generates

<ol CLEAR="cclear" WRAP="cwrap" cattributes>
215.6.91 PARA Function

This function generates the `<P>` tag which indicates that the text that comes after the tag is to be formatted as a paragraph.

You can add attributes to the tag by means of the PARAGRAPH Function.

Syntax

```
HTF.PARA
  RETURN VARCHAR2;
```

Examples

This function generates

```
<P>
```

215.6.92 PARAGRAPH Function

You can use this function to add attributes to the `<P>` tag created by the PARA Function.

Syntax

```
HTF.PARAGRAPH(
  calign         IN       VARCHAR2   DEFAULT NULL,
  cnowrap        IN       VARCHAR2   DEFAULT NULL,
  cclear         IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 215-67 PARAGRAPH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

Related Topics

- PARA Function
  This function generates the `<P>` tag which indicates that the text that comes after the tag is to be formatted as a paragraph.
215.6.93 PARAM Function

This function generates the `<PARAM>` tag which specifies parameter values for Java applets.

The values can reference HTML variables. To invoke a Java applet from a Web page, use `APPLETOPEN Function` to begin the invocation. Use one `PARAM Function` for each desired name-value pair, and use `APPLETCLOSE Function` to end the applet invocation.

**Syntax**

```sql
HTF.PARAM(
    cname          IN       VARCHAR2,
    cvalue         IN       VARCHAR2
)
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```html
<PARAM NAME=cname VALUE="cvalue">
```

215.6.94 PLAINTEXT Function

This function generates the `<PLAINTEXT>` and `</PLAINTEXT>` tags which direct the browser to render the text they surround in fixed-width type.

**Syntax**

```sql
HTF.PLAINTEXT(
    ctext          IN       VARCHAR2,
    cattributes    IN       VARCHAR2 DEFAULT NULL
)
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in fixed-width font.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples

This function generates

<PLAINTEXT cattributes>ctext</PLAINTEXT>

215.6.95 PRECLOSE Function

This function generates the </PRE> tag which marks the end of a section of preformatted text in the body of the HTML page.

To mark the beginning of a section of preformatted text in the body of the HTML page, use the PREOPEN Function.

Syntax

HTF.PRECLOSE
  RETURN VARCHAR2;

Examples

This function generates

</PRE>

215.6.96 PREOPEN Function

This function generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page.

To mark the end of a section of preformatted text in the body of the HTML page, use the PRECLOSE Function.

Syntax

HTF.PREOPEN(
  cclear         IN       VARCHAR2   DEFAULT NULL,
  cwidth         IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
  RETURN VARCHAR2;

Parameters

Table 215-70  PREOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cwidth</td>
<td>The value for the WIDTH attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
215.6.97 PRINT Functions

These functions generate the specified parameter as a string terminated with the \n newline character.

The PRN Functions performs the same operation but does not terminate with a newline character.

Syntax

HTF.PRINT (  
    cbuf      IN       VARCHAR2)  
RETURN VARCHAR2;

HTF.PRINT (  
   dbuf      IN       DATE)  
RETURN VARCHAR2;

HTF.PRINT (  
    nbuf      IN       NUMBER)  
RETURN VARCHAR2;

Parameters

Table 215-71  PRINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbuf</td>
<td>The string to generate terminated by a newline.</td>
</tr>
<tr>
<td>dbuf</td>
<td>The string to generate terminated by a newline.</td>
</tr>
<tr>
<td>nbuf</td>
<td>The string to generate terminated by a newline.</td>
</tr>
</tbody>
</table>

Usage Notes

• The \n character is not the same as <BR>. The \n character formats the HTML source but it does not affect how the browser renders the HTML source. Use <BR> to control how the browser renders the HTML source.

• These functions do not have function equivalents.

215.6.98 PRN Functions

These functions generate the specified parameter as a string.

Unlike the PRINT Functions the string is not terminated with the \n newline character.

Syntax

HTF.PRIN (  
    cbuf      IN       VARCHAR2)  
RETURN VARCHAR2;

HTF.PRIN (  
   dbuf      IN       DATE)  
RETURN VARCHAR2;
HTF.PRN (  
nbuf      IN       NUMBER)  
RETURN VARCHAR2;

Parameters

Table 215-72  PRN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbuf</td>
<td>The string to generate (not terminated by a newline).</td>
</tr>
<tr>
<td>dbuf</td>
<td>The string to generate (not terminated by a newline).</td>
</tr>
<tr>
<td>nbuf</td>
<td>The string to generate (not terminated by a newline).</td>
</tr>
</tbody>
</table>

Usage Notes

These functions do not have function equivalents.

215.6.99 S Function

This function generates the <S> and </S> tags which direct the browser to render the text they surround in strikethrough type.

This performs the same operation as STRIKE Function.

Syntax

HTF.S (  
ctext          IN       VARCHAR2,  
cattributes    IN       VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;

Parameters

Table 215-73  S Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in strikethrough type.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

<S cattributes>ctext</S>
215.6.100 SAMPLE Function

This function generates the `<SAMP>` and `</SAMP>` tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically.

Syntax

```
HTF.SAMPLE (  
ctext          IN       VARCHAR2,  
cattributes    IN       VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

Table 215-74  SAMPLE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in monospace font.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<SAMP cattributes>ctext</SAMP>
```

215.6.101 SCRIPT Function

This function generates the `<SCRIPT>` and `</SCRIPT>` tags which contain a script written in languages such as JavaScript and VBscript.

Syntax

```
HTF.SCRIPT (  
cscript        IN       VARCHAR2,  
clanguage      IN       VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

Table 215-75  SCRIPT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cscript</td>
<td>The text of the script. This is the text that makes up the script itself, not the name of a file containing the script.</td>
</tr>
<tr>
<td>clanguage</td>
<td>The language in which the script is written. If this parameter is omitted, the user's browser determines the scripting language.</td>
</tr>
</tbody>
</table>

Examples

This function generates
so that

HTF.script ('Erupting_Volcano', 'Javascript');

generates

<SCRIPT LANGUAGE=Javascript>"script text here"</SCRIPT>

This causes the browser to run the script enclosed in the tags.

### 215.6.102 SMALL Function

This function generates the `<SMALL>` and `</SMALL>` tags, which direct the browser to render the text they surround using a small font.

**Syntax**

```sql
HTF.SMALL (  
  ctext          IN       VARCHAR2,  
  cattributes    IN       VARCHAR2   DEFAULT NULL) 
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in small font.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```sql
<SMALL cattributes>ctext</SMALL>
```

### 215.6.103 STRIKE Function

This function generates the `<STRIKE>` and `</STRIKE>` tags which direct the browser to render the text they surround in strikethrough type.

This performs the same operation as `S Function`.

**Syntax**

```sql
STRIKE (  
  ctext          IN       VARCHAR2,  
  cattributes    IN       VARCHAR2   DEFAULT NULL) 
RETURN VARCHAR2;
```
Parameters

Table 215-77   STRIKE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in strikethrough type.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<STRIKE cattributes>ctext</STRIKE>
```

215.6.104 STRONG Function

This function generates the `<STRONG>` and `</STRONG>` tags which direct the browser to render the text they surround in bold, or however "strong" is defined.

Syntax

```sql
HTF.STRONG(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-78  STRONG Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be emphasized.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<STRONG cattributes>ctext</STRONG>
```

215.6.105 STYLE Function

This function generates the `<STYLE>` and `</STYLE>` tags which include a style sheet in a Web page.

You can get more information about style sheets at [http://www.w3.org](http://www.w3.org). This feature is not compatible with browsers that support only HTML versions 2.0 or earlier. Such browsers will ignore this tag.
Syntax

```sql
HTF.STYLE(
    cstyle          IN       VARCHAR2
) RETURN VARCHAR2;
```

Parameters

**Table 215-79  STYLE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cstyle</td>
<td>The style information to include.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<style>cstyle</style>
```

### 215.6.106 SUB Function

This function generates the `<SUB>` and `</SUB>` tags which direct the browser to render the text they surround as subscript.

Syntax

```sql
HTF.SUB(
    ctext          IN       VARCHAR2,
    calign         IN       VARCHAR2 DEFAULT NULL,
    cattributes    IN       VARCHAR2 DEFAULT NULL
) RETURN VARCHAR2;
```

Parameters

**Table 215-80  SUB Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in subscript.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<sub align="calign" cattributes>ctext</sub>
```
215.6.107 SUP Function

This function generates the `<SUP>` and `</SUP>` tags which direct the browser to render the text they surround as superscript.

**Syntax**

```
HTF.SUP(
    ctext          IN       VARCHAR2,
    calign         in       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

**Table 215-81  SUP Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in superscript.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

215.6.108 TABLECAPTION Function

This function generates the `<CAPTION>` and `</CAPTION>` tags which place a caption in an HTML table.

**Syntax**

```
HTF.TABLECAPTION(
    ccaption       IN       VARCHAR2,
    calign         in       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

**Table 215-82  TABLECAPTION Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text for the caption.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
215.6.109 TABLECLOSE Function

This function generates the </TABLE> tag which marks the end of an HTML table.

To define the beginning of an HTML table, use the TABLEOPEN Function.

Syntax

```sql
HTF.TABLECLOSE
RETURN VARCHAR2;
```

Examples

This function generates

```html
</TABLE>
```

215.6.110 TABLEDATA Function

This function generates the <TD> and </TD> tags which insert data into a cell of an HTML table.

Syntax

```sql
HTF.TABLEDATA(
   cvalue         IN       VARCHAR2   DEFAULT NULL,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cdp            IN       VARCHAR2   DEFAULT NULL,
   cnowrap        IN       VARCHAR2   DEFAULT NULL,
   crowspan       IN       VARCHAR2   DEFAULT NULL,
   ccolspan       IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-83  TABLEDATA Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The data for the cell in the table.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cdp</td>
<td>The value for the DP attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>crowspan</td>
<td>The value for the COLSPAN attribute.</td>
</tr>
<tr>
<td>ccolspan</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples

This function generates

<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TD>

### 215.6.111 TABLEHEADER Function

This function generates the `<TH>` and `</TH>` tags which insert a header cell in an HTML table.

The `<TH>` tag is similar to the `<TD>` tag except that the text in this case the rows are usually rendered in bold type.

**Syntax**

```
HTF.TABLEHEADER(    
  cvalue        IN  VARCHAR2 DEFAULT NULL, 
  calign        IN  VARCHAR2 DEFAULT NULL, 
  cdp           IN  VARCHAR2 DEFAULT NULL, 
  cnnowrap      IN  VARCHAR2 DEFAULT NULL, 
  crowspan      IN  VARCHAR2 DEFAULT NULL, 
  ccolspan      IN  VARCHAR2 DEFAULT NULL, 
  cattributes   IN  VARCHAR2 DEFAULT NULL) 
RETURN VARCHAR2;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The data for the cell in the table.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cdp</td>
<td>The value for the DP attribute.</td>
</tr>
<tr>
<td>cnnowrap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to</td>
</tr>
<tr>
<td></td>
<td>the tag.</td>
</tr>
<tr>
<td>crowspan</td>
<td>The value for the ROWSPAN attribute.</td>
</tr>
<tr>
<td>ccolspan</td>
<td>The value for the COLSPAN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TH>
```
### 215.6.112 TABLEOPEN Function

This function generates the `<TABLE>` tag which marks the beginning of an HTML table.

To define the end of an HTML table, use the TABLECLOSE Function.

**Syntax**

```
HTF.TABLEOPEN(
    cborder IN VARCHAR2 DEFAULT NULL,
    calign IN VARCHAR2 DEFAULT NULL,
    cnowrap IN VARCHAR2 DEFAULT NULL,
    cclear IN VARCHAR2 DEFAULT NULL,
    cattributes IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

Table 215-85  **TABLEOPEN Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border</td>
<td>The value for the BORDER attribute.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```
<TABLE "cborder" NOWRAP ALIGN="calign" CLEAR="cclear" cattributes>
```

### 215.6.113 TABLEROWCLOSE Function

This function generates the `<TR>` tag which marks the end of a new row in an HTML table.

To mark the beginning of a new row, use the TABLEROWOPEN Function.

**Syntax**

```
HTF.TABLEROWCLOSE
RETURN VARCHAR2;
```

**Examples**

This function generates

```
</TABLE>
```
215.6.114 TABLEROWOPEN Function

This function generates the `<TR>` tag which marks the beginning of a new row in an HTML table.

To mark the end of a new row, use the TABLEROWCLOSE Function.

**Syntax**

```sql
HTF.TABLEROWOPEN(
    calign  IN       VARCHAR2   DEFAULT NULL,
    cvalign IN       VARCHAR2   DEFAULT NULL,
    cdp     IN       VARCHAR2   DEFAULT NULL,
    cnoswap IN       VARCHAR2   DEFAULT NULL,
    cattributes IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

**Table 215-86  TABLEROWOPEN Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cvalign</td>
<td>The value for the VALIGN attribute.</td>
</tr>
<tr>
<td>cdp</td>
<td>The value for the DP attribute.</td>
</tr>
<tr>
<td>cnoswap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This function generates

```
<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>
```

215.6.115 TELETYPETYPE Function

This function generates the `<TT>` and `</TT>` tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font.

**Syntax**

```sql
HTF.TELETYPETYPE(
    ctext    IN       VARCHAR2,
    cattributes IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```
Parameters

### Table 215-87  TELETYP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in a fixed width typewriter font.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<TT cattributes>ctext</TT>
```

### 215.6.116 TITLE Function

This function generates the `<TITLE>` and `</TITLE>` tags which specify the text to display in the titlebar of the browser window.

**Syntax**

```sql
HTF.TITLE(  
    ctitle          IN       VARCHAR2)  
RETURN VARCHAR2;
```

Parameters

### Table 215-88  TITLE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctitle</td>
<td>The text to display in the titlebar of the browser window.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```html
<TITLE>ctitle</TITLE>
```

### 215.6.117 ULISTCLOSE Function

This function generates the `</UL>` tag which marks the end of an unordered list. An unordered list presents items with bullets.

To mark the beginning of an unordered list, use the ULISTOPEN Function. Add list items with LISTITEM Function.

**Syntax**

```sql
HTF.ULISTCLOSE  
RETURN VARCHAR2;
```
Examples
This function generates
</UL>

215.6.118 ULISTOPEN Function

This function generates the <UL> tag which marks the beginning of an unordered list. An unordered list presents items with bullets.

To mark the end of an unordered list, use the ULISTCLOSE Function. Add list items with LISTITEM Function.

Syntax

```
HTF.ULISTOPEN(
  cclear         IN       VARCHAR2   DEFAULT NULL,
  cwrap          IN       VARCHAR2   DEFAULT NULL,
  cdingbat       IN       VARCHAR2   DEFAULT NULL,
  csrc           IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-89  ULISTOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the WRAP attribute.</td>
</tr>
<tr>
<td>cdingbat</td>
<td>The value for the DINGBAT attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

215.6.119 UNDERLINE Function

This function generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline.

Syntax

```
HTF.UNDERLINE(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```
Parameters

Table 215-90  UNDERLINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render with an underline.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

\(<U cattributes>c\text{text}</U>\)

215.6.120 VARIABLE Function

This function generates the \(<VAR>\) and \(</VAR>\) tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.

Syntax

```
HTF.VARIABLE(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 215-91  VARIABLE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in italics.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This function generates

\(<VAR cattributes>c\text{text}</VAR>\)

215.6.121 WBR Function

This function generates the \(<WBR>\) tag, which inserts a soft line break within a section of NOBR text.

Syntax

```
HTF.WBR
RETURN VARCHAR2;
```

Examples

This function generates
Chapter 215
Summary of HTF Subprograms
The HTP (hypertext procedures) and HTF (hypertext functions) packages generate HTML tags. For example, the HTP.ANCHOR procedure generates the HTML anchor tag, `<A>`.

This chapter contains the following topics:

- Operational Notes
- Rules and Limits
- Examples
- Summary of Tags
- Summary of HTP Subprograms

**See Also:**

For more information about implementation of this package:

- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

## 216.1 HTP Operational Notes

Certain operational notes apply to the HTP package.

For every HTP procedure that generates one or more HTML tags, there is a corresponding HTF function with identical parameters with the following exception:

- The PRINTS Procedure and the PS Procedure do not have HTF function equivalents. Use the ESCAPE_SC Function or the ESCAPE_URL Function if you need a string conversion function. Note that while there is a ESCAPE_SC Procedure that performs the same operation as the PRINTS Procedure and the PS Procedure, there is no procedural equivalent for the ESCAPE_URL Function.

- The FORMAT_CELL Function does not have an HTP equivalent. The function formats column values inside an HTML table using TABLEDATA Function which does have an HTP equivalent in the TABLEDATA Procedure. The advantage of this using the FORMAT_CELL Function is that it allows for better control over the HTML tables.

The function versions do not directly generate output in your Web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls. To print the output of HTF functions, call the functions from within the HTP.PRINT procedure. It then prints its parameters to the generated Web page.
216.2 HTP Rules and Limits

If you use values of the LONG datatype in procedures such as HTP.PRINT, HTP.PRN, HTP.PRINTS, HTP.PA or OWA_UTIL.CELLSPRINT, only the first 32 K of the LONG data is used. The LONG data is bound to a VARCHAR2 datatype in the procedure.

216.3 HTP Examples

These commands generate a simple HTML document.

```sql
CREATE OR REPLACE PROCEDURE hello AS
BEGIN
    HTP.HTMLOPEN;           -- generates <HTML>
    HTP.HEADOPEN;           -- generates <HEAD>
    HTP.TITLE('Hello');     -- generates <TITLE>Hello</TITLE>
    HTP.HEADCLOSE;          -- generates </HEAD>
    HTP.BODYOPEN;           -- generates <BODY>
    HTP.HEADER(1, 'Hello'); -- generates <H1>Hello</H1>
    HTP.BODYCLOSE;          -- generates </BODY>
    HTP.HTMLCLOSE;          -- generates </HTML>
END;
```

216.4 HTP Summary of Tags

HTP tags can be grouped by type, to make understanding them easier.

**HTML, HEAD, and BODY Tags**

- **HTMLOPEN Procedure, HTMLCLOSE Procedure** - generate `<HTML>` and `</HTML>`
- **HEADOPEN Procedure, HEADCLOSE Procedure** - generate `<HEAD>` and `</HEAD>`
- **BODYOPEN Procedure, BODYCLOSE Procedure** - generate `<BODY>` and `</BODY>`

**Comment Tag**

- **COMMENT Procedure** - generates `<!--` and `-->`

**Tags in the `<HEAD>` Area**

- **BASE Procedure** - generates `<BASE>`
- **LINKREL Procedure** - generates `<LINK>` with the REL attribute
- **LINKREV Procedure** - generates `<LINK>` with the REV attribute
- **TITLE Procedure** - generates `<TITLE>`
- **META Procedure** - generates `<META>`
- **SCRIPT Procedure** - generates `<SCRIPT>`
- **STYLE Procedure** - generates `<STYLE>`
- **ISINDEX Procedure** - generates `<ISINDEX>`
Applet Tags
APPLETOPEN Procedure, APPLETCLOSE Procedure - generate <APPLET> and </APPLET>
PARAM Procedure - generates <PARAM>

List Tags
OLISTOPEN Procedure, OLISTCLOSE Procedure - generate <OL> and </OL>
ULISTOPEN Procedure, ULISTCLOSE Procedure - generate <UL> and </UL>
DLISTOPEN Procedure, DLISTCLOSE Procedure - generate <DL> and </DL>
DLISTTERM Procedure - generates <DT>
DLISTDEF Procedure - generates <DD>
DIRLISTOPEN Procedure, DIRLISTCLOSE Procedure - generate <DIR> and </DIR>
LISTHEADER Procedure - generates <LH>
LISTINGOPEN Procedure, LISTINGCLOSE Procedure - generate <LISTING> and </LISTING>
MENULISTOPEN Procedure - generate <MENU> and </MENU>
LISTITEM Procedure - generates <LI>

Form Tags
FORMOPEN Procedure, FORMCLOSE Procedure - generate <FORM> and </FORM>
FORMCHECKBOX Procedure - generates <INPUT TYPE="CHECKBOX">
FORMHIDDEN Procedure - generates <INPUT TYPE="HIDDEN">
FORMIMAGE Procedure - generates <INPUT TYPE="IMAGE">
FORMPASSWORD Procedure - generates <INPUT TYPE="PASSWORD">
FORMRADIO Procedure - generates <INPUT TYPE="RADIO">
FORMSELECTOPEN Procedure, FORMSELECTCLOSE Procedure - generate <SELECT> and </SELECT>
FORMSELECTOPTION Procedure - generates <OPTION>
FORMTEXT Procedure - generates <INPUT TYPE="TEXT">
FORMTEXTAREA Procedure - generate <TEXTAREA>
FORMTEXTAREAOPEN Procedure, FORMTEXTAREACLOSE Procedure - generate <TEXTAREA> and </TEXTAREA>
FORMRESET Procedure - generates <INPUT TYPE="RESET">
FORMSUBMIT Procedure - generates <INPUT TYPE="SUBMIT">
Table Tags

**TABLEOPEN Procedure, TABLECLOSE Procedure** - generate `<TABLE>` and `<TABLE>`

**TABLECAPTION Procedure** - generates `<CAPTION>`

**TABLEROWOPEN Procedure, TABLEROWCLOSE Procedure** - generate `<TR>` and `<TR>`

**TABLEHEADER Procedure** - generates `<TH>`

**TABLEDATA Procedure** - generates `<TD>`

**IMG, HR, and A Tags**

**HR Procedure, LINE Procedure** - generate `<HR>`

**IMG Procedure, IMG2 Procedure** - generate `<IMG>`

**ANCHOR Procedure, ANCHOR2 Procedure** - generate `<A>`

**MAPOPEN Procedure, MAPCLOSE Procedure** - generate `<MAP>` and `<MAP>`

**Paragraph Formatting Tags**

**HEADER Procedure** - generates heading tags (`<H1>` to `<H6>`)

**PARA Procedure, PARAGRAPH Procedure** - generate `<P>`

**PRN Procedures, PRINT Procedures** - generate any text that is passed in

**PRINTS Procedure, PS Procedure** - generate any text that is passed in; special characters in HTML are escaped

**PREOPEN Procedure, PRECLOSE Procedure** - generate `<PRE>` and `<PRE>`

**BLOCKQUOTEOPEN Procedure, BLOCKQUOTECLOSE Procedure** - generate `<BLOCKQUOTE>` and `<BLOCKQUOTE>`

**DIV Procedure** - generates `<DIV>`

**NL Procedure, BR Procedure** - generate `<BR>`

**NOBR Procedure** - generates `<NOBR>`

**WBR Procedure** - generates `<WBR>`

**PLAINTEXT Procedure** - generates `<PLAINTEXT>`

**ADDRESS Procedure** - generates `<ADDRESS>`

**MAILTO Procedure** - generates `<A>` with the MAILTO attribute

**AREA Procedure** - generates `<AREA>`

**BGSOUND Procedure** - generates `<BGSOUND>`

**Character Formatting Tags**

**BASEFONT Procedure** - generates `<BASEFONT>`
BIG Procedure - generates <BIG>
BOLD Procedure - generates <B>
CENTER Procedure - generates <CENTER> and </CENTER>
CENTEROPEN Procedure, CENTERCLOSE Procedure - generate <CENTER> and </CENTER>
CITE Procedure - generates <CITE>
CODE Procedure - generates <CODE>
DFN Procedure - generates <DFN>
EM Procedure, EMPHASIS Procedure - generate <EM>
FONTOPEN Procedure, FONTCLOSE Procedure - generate <FONT> and </FONT>
ITALIC Procedure - generates <I>
KBD Procedure, KEYBOARD Procedure - generate <KBD> and </KBD>
S Procedure - generates <S>
SAMPLE Procedure - generates <SAMP>
SMALL Procedure - generates <SMALL>
STRIKE Procedure - generates <STRIKE>
STRONG Procedure - generates <STRONG>
SUB Procedure - generates <SUB>
SUP Procedure - generates <SUP>
TELETYPE Procedure - generates <TT>
UNDERLINE Procedure - generates <U>
VARIABLE Procedure - generates <VAR>

Frame Tags
FRAME Procedure - generates <FRAME>
FRAMESETOPEN Procedure, FRAMESETCLOSE Procedure - generate <FRAMESET> and </FRAMESET>
NOFRAMESOPEN Procedure, NOFRAMESCLOSE Procedure - generate <NOFRAMES> and </NOFRAMES>

216.5 Summary of HTP Subprograms

This table lists the HTP subprograms and briefly describes them.
<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS Procedure</td>
<td>Generates the <code>&lt;ADDRESS&gt;</code> and <code>&lt;/ADDRESS&gt;</code> tags which specify the address, author and signature of a document</td>
</tr>
<tr>
<td>ANCHOR Procedure</td>
<td>Generates the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags which specify the source or destination of a hypertext link</td>
</tr>
<tr>
<td>ANCHOR2 Procedure</td>
<td>Generates the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags which specify the source or destination of a hypertext link</td>
</tr>
<tr>
<td>APPLETCLOSE Procedure</td>
<td>Closes the applet invocation with the <code>&lt;/APPLET&gt;</code> tag</td>
</tr>
<tr>
<td>APPLETOPEN Procedure</td>
<td>Generates the <code>&lt;APPLET&gt;</code> tag which begins the invocation of a Java applet</td>
</tr>
<tr>
<td>AREA Procedure</td>
<td>Generates the <code>&lt;AREA&gt;</code> tag, which defines a client-side image map</td>
</tr>
<tr>
<td>BASE Procedure</td>
<td>Generates the <code>&lt;BASE&gt;</code> tag which records the URL of the document</td>
</tr>
<tr>
<td>BASEFONT Procedure</td>
<td>Generates the <code>&lt;BASEFONT&gt;</code> tag which specifies the base font size for a Web page</td>
</tr>
<tr>
<td>BGSOUND Procedure</td>
<td>Generates the <code>&lt;BGSOUND&gt;</code> tag which includes audio for a Web page</td>
</tr>
<tr>
<td>BIG Procedure</td>
<td>Generates the <code>&lt;BIG&gt;</code> and <code>&lt;/BIG&gt;</code> tags which direct the browser to render the text in a bigger font</td>
</tr>
<tr>
<td>BLOCKQUOTECLOSE Procedure</td>
<td>Generates the <code>&lt;/BLOCKQUOTE&gt;</code> tag which mark the end of a section of quoted text</td>
</tr>
<tr>
<td>BLOCKQUOTEOPEN Procedure</td>
<td>Generates the <code>&lt;BLOCKQUOTE&gt;</code> tag, which marks the beginning of a section of quoted text</td>
</tr>
<tr>
<td>BODYCLOSE Procedure</td>
<td>Generates the <code>&lt;/BODY&gt;</code> tag which marks the end of a body section of an HTML document</td>
</tr>
<tr>
<td>BODYOPEN Procedure</td>
<td>Generates the <code>&lt;BODY&gt;</code> tag which marks the beginning of the body section of an HTML document</td>
</tr>
<tr>
<td>BOLD Procedure</td>
<td>Generates the <code>&lt;B&gt;</code> and <code>&lt;/B&gt;</code> tags which direct the browser to display the text in boldface</td>
</tr>
<tr>
<td>BR Procedure</td>
<td>Generates the <code>&lt;BR&gt;</code> tag which begins a new line of text</td>
</tr>
<tr>
<td>CENTER Procedure</td>
<td>Generates the <code>&lt;CENTER&gt;</code> and <code>&lt;/CENTER&gt;</code> tags which center a section of text within a Web page</td>
</tr>
<tr>
<td>CENTERCLOSE Procedure</td>
<td>Generates the <code>&lt;/CENTER&gt;</code> tag which marks the end of a section of text to center</td>
</tr>
<tr>
<td>CENTEROPEN Procedure</td>
<td>Generates the <code>&lt;CENTER&gt;</code> tag which mark the beginning of a section of text to center</td>
</tr>
<tr>
<td>CITE Procedure</td>
<td>Generates the <code>&lt;CITE&gt;</code> and <code>&lt;/CITE&gt;</code> tags which direct the browser to render the text as a citation</td>
</tr>
<tr>
<td>CODE Procedure</td>
<td>Generates the <code>&lt;CODE&gt;</code> and <code>&lt;/CODE&gt;</code> tags which direct the browser to render the text in monospace font or however &quot;code&quot; is defined stylistically</td>
</tr>
<tr>
<td>COMMENT Procedure</td>
<td>Generates procedure generates the comment tags <code>&lt;!-- ctext --&gt;</code></td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DFN Procedure</td>
<td>Generates the <code>&lt;DFN&gt;</code> and <code>&lt;/DFN&gt;</code> tags which direct the browser to mark the text as italics or however &quot;definition&quot; is defined stylistically</td>
</tr>
<tr>
<td>DIRLISTCLOSE Procedure</td>
<td>Generates the <code>&lt;/DIR&gt;</code> tag which ends a directory list section</td>
</tr>
<tr>
<td>DIRLISTOPEN Procedure</td>
<td>Generates the <code>&lt;DIR&gt;</code> tag which starts a directory list section</td>
</tr>
<tr>
<td>DIV Procedure</td>
<td>Generates the <code>&lt;DIV&gt;</code> tag which creates document divisions</td>
</tr>
<tr>
<td>DLISTCLOSE Procedure</td>
<td>Generates the <code>&lt;DL&gt;</code> tag which ends a definition list</td>
</tr>
<tr>
<td>DLISTDEF Procedure</td>
<td>Generates the <code>&lt;DD&gt;</code> tag, which inserts definitions of terms</td>
</tr>
<tr>
<td>DLISTOPEN Procedure</td>
<td>Generates the <code>&lt;DL&gt;</code> tag which starts a definition list</td>
</tr>
<tr>
<td>DLISTTERM Procedure</td>
<td>Generates the <code>&lt;DT&gt;</code> tag which defines a term in a definition list <code>&lt;DL&gt;</code></td>
</tr>
<tr>
<td>EM Procedure</td>
<td>Generates the <code>&lt;EM&gt;</code> and <code>&lt;/EM&gt;</code> tags, which define text to be emphasized</td>
</tr>
<tr>
<td>EMPHASIS Procedure</td>
<td>Generates the <code>&lt;EM&gt;</code> and <code>&lt;/EM&gt;</code> tags, which define text to be emphasized</td>
</tr>
<tr>
<td>ESCAPE_SC Procedure</td>
<td>Replaces characters that have special meaning in HTML with their escape sequences</td>
</tr>
<tr>
<td>FONTCLOSE Procedure</td>
<td>Generates the <code>&lt;/FONT&gt;</code> tag which marks the end of a section of text with the specified font characteristics</td>
</tr>
<tr>
<td>FONTOPEN Procedure</td>
<td>Generates the <code>&lt;FONT&gt;</code> tag which marks the beginning of section of text with the specified font characteristics</td>
</tr>
<tr>
<td>FORMCHECKBOX Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;checkbox&quot; which inserts a checkbox element in a form</td>
</tr>
<tr>
<td>FORMCLOSE Procedure</td>
<td>Generates the <code>&lt;/FORM&gt;</code> tag which marks the end of a form section in an HTML document</td>
</tr>
<tr>
<td>FORMOPEN Procedure</td>
<td>Generates the <code>&lt;FORM&gt;</code> tag which marks the beginning of a form section in an HTML document</td>
</tr>
<tr>
<td>FORMFILE Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;file&quot; which inserts a file form element, and is used for file uploading for a given page</td>
</tr>
<tr>
<td>FORMHIDDEN Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;hidden&quot;which inserts a hidden form element</td>
</tr>
<tr>
<td>FORMIMAGE Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;image&quot; which creates an image field that the user clicks to submit the form immediately</td>
</tr>
<tr>
<td>FORMPASSWORD Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;password&quot; which creates a single-line text entry field</td>
</tr>
<tr>
<td>FORMRADIO Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;radio&quot;, which creates a radio button on the HTML form</td>
</tr>
<tr>
<td>FORMRESET Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;reset&quot; which creates a button that, when selected, resets the form fields to their initial values</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FORMSELECTCLOSE Procedure</td>
<td>Generates the <code>&lt;SELECT&gt;</code> tag which marks the end of a Select form element</td>
</tr>
<tr>
<td>FORMSELECTOPEN Procedure</td>
<td>Generates the <code>&lt;SELECT&gt;</code> tag which marks the beginning of a Select form element</td>
</tr>
<tr>
<td>FORMSELECTOPTION Procedure</td>
<td>Generates the <code>&lt;OPTION&gt;</code> tag which represents one choice in a Select element</td>
</tr>
<tr>
<td>FORMSUBMIT Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;submit&quot; which creates a button that, when clicked, submits the form</td>
</tr>
<tr>
<td>FORMTEXT Procedure</td>
<td>Generates the <code>&lt;INPUT&gt;</code> tag with TYPE=&quot;text&quot;, which creates a field for a single line of text</td>
</tr>
<tr>
<td>FORMTEXTAREA Procedure</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> tag, which creates a text field that has no predefined text in the text area</td>
</tr>
<tr>
<td>FORMTEXTAREA2 Procedure</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> tag, which creates a text field that has no predefined text in the text area with the ability to specify a wrap style</td>
</tr>
<tr>
<td>FORMTEXTAREACLOSE Procedure</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> tag which ends a text area form element</td>
</tr>
<tr>
<td>FORMTEXTAREAOPEN Procedure</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> which marks the beginning of a text area form element</td>
</tr>
<tr>
<td>FORMTEXTAREAOPEN2 Procedure</td>
<td>Generates the <code>&lt;TEXTAREA&gt;</code> which marks the beginning of a text area form element with the ability to specify a wrap style</td>
</tr>
<tr>
<td>FRAME Procedure</td>
<td>Generates the <code>&lt;FRAME&gt;</code> tag which begins the characteristics of a frame created by a <code>&lt;FRAMESET&gt;</code> tag</td>
</tr>
<tr>
<td>FRAMESETCLOSE Procedure</td>
<td>Generates the <code>&lt;FRAMESET&gt;</code> tag which ends a frameset section</td>
</tr>
<tr>
<td>FRAMESETOPEN Procedure</td>
<td>Generates the <code>&lt;FRAMESET&gt;</code> tag which begins a frameset section</td>
</tr>
<tr>
<td>HEADCLOSE Procedure</td>
<td>Generates the <code>&lt;HEAD&gt;</code> tag which marks the end of an HTML document head section</td>
</tr>
<tr>
<td>HEADER Procedure</td>
<td>Generates opening heading tags (&lt;H1&gt; to &lt;H6&gt;) and their corresponding closing tags (&lt;H1&gt; to &lt;H6&gt;)</td>
</tr>
<tr>
<td>HEADOPEN Procedure</td>
<td>Generates the <code>&lt;HEAD&gt;</code> tag which marks the beginning of the HTML document head section</td>
</tr>
<tr>
<td>HR Procedure</td>
<td>Generates the <code>&lt;HR&gt;</code> tag, which generates a line in the HTML document</td>
</tr>
<tr>
<td>HTMLCLOSE Procedure</td>
<td>Generates the <code>&lt;HTML&gt;</code> tag which marks the end of an HTML document</td>
</tr>
<tr>
<td>HTMLOPEN Procedure</td>
<td>Generates the <code>&lt;HTML&gt;</code> tag which marks the beginning of an HTML document</td>
</tr>
<tr>
<td>IMG Procedure</td>
<td>Generates the <code>&lt;IMG&gt;</code> tag which directs the browser to load an image onto the HTML page</td>
</tr>
<tr>
<td>IMG2 Procedure</td>
<td>Generates the <code>&lt;IMG&gt;</code> tag which directs the browser to load an image onto the HTML page with the option of specifying values for the USEMAP attribute</td>
</tr>
</tbody>
</table>
Table 216-1 (Cont.) HTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISINDEX Procedure</td>
<td>Creates a single entry field with a prompting text, such as &quot;enter value,&quot;</td>
</tr>
<tr>
<td></td>
<td>then sends that value to the URL of the page or program</td>
</tr>
<tr>
<td>ITALIC Procedure</td>
<td>Generates the <code>&lt;I&gt;</code> and <code>&lt;/I&gt;</code> tags which direct the browser to render the</td>
</tr>
<tr>
<td></td>
<td>text in italics</td>
</tr>
<tr>
<td>KBD Procedure</td>
<td>Generates the <code>&lt;KBD&gt;</code> and <code>&lt;/KBD&gt;</code> tags which direct the browser to render</td>
</tr>
<tr>
<td></td>
<td>the text in monospace font</td>
</tr>
<tr>
<td>KEYBOARD Procedure</td>
<td>Generates the <code>&lt;KBD&gt;</code> and <code>&lt;/KBD&gt;</code> tags, which direct the browser to render</td>
</tr>
<tr>
<td></td>
<td>the text in monospace font</td>
</tr>
<tr>
<td>LINE Procedure</td>
<td>Generates the <code>&lt;HR&gt;</code> tag, which generates a line in the HTML document</td>
</tr>
<tr>
<td>LINKREL Procedure</td>
<td>Generates the <code>&lt;LINK&gt;</code> tag with the <code>REL</code> attribute which delineates the</td>
</tr>
<tr>
<td></td>
<td>relationship described by the hypertext link from the anchor to the target</td>
</tr>
<tr>
<td>LINKREV Procedure</td>
<td>Generates the <code>&lt;LINK&gt;</code> tag with the <code>REV</code> attribute which delineates the</td>
</tr>
<tr>
<td></td>
<td>relationship described by the hypertext link from the target to the anchor</td>
</tr>
<tr>
<td>LISTHEADER Procedure</td>
<td>Generates the <code>&lt;LH&gt;</code> and <code>&lt;/LH&gt;</code> tags which print an HTML tag at the</td>
</tr>
<tr>
<td></td>
<td>beginning of the list</td>
</tr>
<tr>
<td>LISTINGCLOSE Procedure</td>
<td>Generates the <code>&lt;/LISTING&gt;</code> tags which marks the end of a section of fixed-</td>
</tr>
<tr>
<td></td>
<td>width text in the body of an HTML page</td>
</tr>
<tr>
<td>LISTINGOPEN Procedure</td>
<td>Generates the <code>&lt;LISTING&gt;</code> tag which marks the beginning of a section of</td>
</tr>
<tr>
<td></td>
<td>fixed-width text in the body of an HTML page</td>
</tr>
<tr>
<td>LISTITEM Procedure</td>
<td>Generates the <code>&lt;LI&gt;</code> tag, which indicates a list item</td>
</tr>
<tr>
<td>MAILTO Procedure</td>
<td>Generates the <code>&lt;A&gt;</code> tag with the <code>HREF</code> set to ‘mailto’ prepended to the</td>
</tr>
<tr>
<td></td>
<td>mail address argument</td>
</tr>
<tr>
<td>MAPCLOSE Procedure</td>
<td>Generates the <code>&lt;/MAP&gt;</code> tag which marks the end of a set of regions in a</td>
</tr>
<tr>
<td></td>
<td>client-side image map</td>
</tr>
<tr>
<td>MAPOPEN Procedure</td>
<td>Generates the <code>&lt;MAP&gt;</code> tag which marks the beginning of a set of regions in</td>
</tr>
<tr>
<td></td>
<td>a client-side image map</td>
</tr>
<tr>
<td>MENULISTCLOSE Procedure</td>
<td>Generates the <code>&lt;MENU&gt;</code> tag which ends a list that presents one line for each</td>
</tr>
<tr>
<td></td>
<td>item</td>
</tr>
<tr>
<td>MENULISTOPEN Procedure</td>
<td>Generates the <code>&lt;MENU&gt;</code> tag which begins a list that presents one line for</td>
</tr>
<tr>
<td></td>
<td>each item</td>
</tr>
<tr>
<td>META Procedure</td>
<td>Generates the <code>&lt;META&gt;</code> tag, which embeds meta-information about the document</td>
</tr>
<tr>
<td></td>
<td>and also specifies values for HTTP headers</td>
</tr>
<tr>
<td>NL Procedure</td>
<td>Generates the <code>&lt;BR&gt;</code> tag which begins a new line of text</td>
</tr>
<tr>
<td>NOBR Procedure</td>
<td>Generates the <code>&lt;NOBR&gt;</code> and <code>&lt;/NOBR&gt;</code> tags which turn off line-breaking in</td>
</tr>
<tr>
<td></td>
<td>a section of text</td>
</tr>
<tr>
<td>NOFRAMESCLOSE Procedure</td>
<td>Generates the <code>&lt;/NOFRAMES&gt;</code> tag which marks the end of a no-frames section</td>
</tr>
<tr>
<td>NOFRAMESOPEN Procedure</td>
<td>Generates the <code>&lt;NOFRAMES&gt;</code> tag which mark the beginning of a no-frames</td>
</tr>
<tr>
<td></td>
<td>section</td>
</tr>
</tbody>
</table>
### Table 216-1  (Cont.) HTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLISTCLOSE Procedure</td>
<td>Generates the <code>&lt;OL&gt;</code> tag which defines the end of an ordered list</td>
</tr>
<tr>
<td>OLISTOPEN Procedure</td>
<td>Generates the <code>&lt;OL&gt;</code> tag which marks the beginning of an ordered list</td>
</tr>
<tr>
<td>PARA Procedure</td>
<td>Generates the <code>&lt;P&gt;</code> tag which indicates that the text that comes after the tag is to be formatted as a paragraph</td>
</tr>
<tr>
<td>PARAM Procedure</td>
<td>Generates the <code>&lt;PARAM&gt;</code> tag which specifies parameter values for Java applets</td>
</tr>
<tr>
<td>PLAINTEXT Procedure</td>
<td>Generates the <code>&lt;PLAINTEXT&gt;</code> and <code>&lt;/PLAINTEXT&gt;</code> tags which direct the browser to render the text they surround in fixed-width type</td>
</tr>
<tr>
<td>PRECLOSE Procedure</td>
<td>Generates the <code>&lt;/PRE&gt;</code> tag which marks the end of a section of preformatted text in the body of the HTML page</td>
</tr>
<tr>
<td>PREOPEN Procedure</td>
<td>Generates the <code>&lt;PRE&gt;</code> tag which marks the beginning of a section of preformatted text in the body of the HTML page</td>
</tr>
<tr>
<td>PRINT Procedures</td>
<td>Generates the specified parameter as a string terminated with the <code>\n</code> newline character</td>
</tr>
<tr>
<td>PRINTS Procedure</td>
<td>Generates a string and replaces the following characters with the corresponding escape sequence</td>
</tr>
<tr>
<td>PRN Procedures</td>
<td>Generates the specified parameter as a string</td>
</tr>
<tr>
<td>PS Procedure</td>
<td>Generates a string and replaces the following characters with the corresponding escape sequence.</td>
</tr>
<tr>
<td>S Procedure</td>
<td>Generates the <code>&lt;S&gt;</code> and <code>&lt;/S&gt;</code> tags which direct the browser to render the text they surround in strikethrough type</td>
</tr>
<tr>
<td>SAMPLE Procedure</td>
<td>Generates the <code>&lt;SAMP&gt;</code> and <code>&lt;/SAMP&gt;</code> tags which direct the browser to render the text they surround in monospace font or however &quot;sample&quot; is defined stylistically</td>
</tr>
<tr>
<td>SCRIPT Procedure</td>
<td>Generates the <code>&lt;SCRIPT&gt;</code> and <code>&lt;/SCRIPT&gt;</code> tags which contain a script written in languages such as JavaScript and VBscript</td>
</tr>
<tr>
<td>SMALL Procedure</td>
<td>Generates the <code>&lt;SMALL&gt;</code> and <code>&lt;/SMALL&gt;</code> tags, which direct the browser to render the text they surround using a small font</td>
</tr>
<tr>
<td>STRIKE Procedure</td>
<td>Generates the <code>&lt;STRIKE&gt;</code> and <code>&lt;/STRIKE&gt;</code> tags which direct the browser to render the text they surround in strikethrough type</td>
</tr>
<tr>
<td>STRONG Procedure</td>
<td>Generates the <code>&lt;STRONG&gt;</code> and <code>&lt;/STRONG&gt;</code> tags which direct the browser to render the text they surround in bold or however &quot;strong&quot; is defined stylistically</td>
</tr>
<tr>
<td>STYLE Procedure</td>
<td>Generates the <code>&lt;STYLE&gt;</code> and <code>&lt;/STYLE&gt;</code> tags which include a style sheet in a Web page</td>
</tr>
<tr>
<td>SUB Procedure</td>
<td>Generates the <code>&lt;SUB&gt;</code> and <code>&lt;/SUB&gt;</code> tags which direct the browser to render the text they surround as subscript</td>
</tr>
<tr>
<td>SUP Procedure</td>
<td>Generates the <code>&lt;SUP&gt;</code> and <code>&lt;/SUP&gt;</code> tags which direct the browser to render the text they surround as superscript</td>
</tr>
</tbody>
</table>
Table 216-1  (Cont.) HTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLECAPTION Procedure</td>
<td>Generates the <code>&lt;CAPTION&gt;</code> and <code>&lt;/CAPTION&gt;</code> tags which place a caption in an HTML table</td>
</tr>
<tr>
<td>TABLECLOSE Procedure</td>
<td>Generates the <code>&lt;/TABLE&gt;</code> tag which marks the end of an HTML table</td>
</tr>
<tr>
<td>TABLEDATA Procedure</td>
<td>Generates the <code>&lt;TD&gt;</code> and <code>&lt;/TD&gt;</code> tags which insert data into a cell of an HTML table</td>
</tr>
<tr>
<td>TABLEHEADER Procedure</td>
<td>Generates the <code>&lt;TH&gt;</code> and <code>&lt;/TH&gt;</code> tags which insert a header cell in an HTML table.</td>
</tr>
<tr>
<td>TABLEOPEN Procedure</td>
<td>Generates the <code>&lt;TABLE&gt;</code> tag which marks the beginning of an HTML table</td>
</tr>
<tr>
<td>TABLEROWCLOSE Procedure</td>
<td>Generates the <code>&lt;/TR&gt;</code> tag which marks the end of a new row in an HTML table</td>
</tr>
<tr>
<td>TABLEROWOPEN Procedure</td>
<td>Generates the <code>&lt;TR&gt;</code> tag which marks the beginning of a new row in an HTML table</td>
</tr>
<tr>
<td>TELETYPETE Procedure</td>
<td>Generates the <code>&lt;TT&gt;</code> and <code>&lt;/TT&gt;</code> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font</td>
</tr>
<tr>
<td>TITLE Procedure</td>
<td>Generates the <code>&lt;TITLE&gt;</code> and <code>&lt;/TITLE&gt;</code> tags which specify the text to display in the titlebar of the browser window</td>
</tr>
<tr>
<td>ULISTCLOSE Procedure</td>
<td>Generates the <code>&lt;/UL&gt;</code> tag which marks the end of an unordered list</td>
</tr>
<tr>
<td>ULISTOPEN Procedure</td>
<td>Generates the <code>&lt;UL&gt;</code> tag which marks the beginning of an unordered list</td>
</tr>
<tr>
<td>UNDERLINE Procedure</td>
<td>Generates the <code>&lt;U&gt;</code> and <code>&lt;/U&gt;</code> tags, which direct the browser to render the text they surround with an underline</td>
</tr>
<tr>
<td>VARIABLE Procedure</td>
<td>Generates the <code>&lt;VAR&gt;</code> and <code>&lt;/VAR&gt;</code> tags which direct the browser to render the text they surround in italics or however &quot;variable&quot; is defined stylistically.</td>
</tr>
<tr>
<td>WBR Procedure</td>
<td>Generates the <code>&lt;WBR&gt;</code> tag, which inserts a soft line break within a section of <code>NOBR</code> text</td>
</tr>
</tbody>
</table>

216.5.1 ADDRESS Procedure

This procedure generates the `<ADDRESS>` and `</ADDRESS>` tags which specify the address, author and signature of a document.

Syntax

```
HTP.ADDRESS (cvalue         IN       VARCHAR2,
cnowrap        IN       VARCHAR2   DEFAULT NULL,
cclear         IN       VARCHAR2   DEFAULT NULL,
cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-2 ADDRESS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The string that goes between the <code>&lt;ADDRESS&gt;</code> and <code>&lt;/ADDRESS&gt;</code> tags.</td>
</tr>
<tr>
<td>cnoswap</td>
<td>If the value for this parameter is not NULL, the NOWRAP attribute is included in the tag.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<Address CLEAR="cclear" NOWRAP cattributes>cvalue</Address>
```

216.5.2 ANCHOR Procedure

Like the ANCHOR2 procedure, this procedure generates the `<A>` and `</A>` HTML tags which specify the source or destination of a hypertext link.

The difference between these subprograms is that the ANCHOR2 Procedure provides a target and therefore can be used for a frame.

Syntax

```
HTP.ANCHOR (curl IN VARCHAR2,
            ctext IN VARCHAR2,
            cname IN VARCHAR2 DEFAULT NULL,
            cattributes IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 216-3 ANCHOR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the HREF attribute.</td>
</tr>
<tr>
<td>ctext</td>
<td>The string that goes between the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```
Usage Notes

This tag accepts several attributes, but either **HREF** or **NAME** must be present. **HREF** specifies where to link. **NAME** allows this tag to be a target of a hypertext link.

216.5.3 ANCHOR2 Procedure

Like the ANCHOR procedure, this procedure generates the `<A>` and `</A>` HTML tags which specify the source or destination of a hypertext link.

The difference between this procedure and the ANCHOR procedure is that this procedure provides a target and therefore can be used for a frame.

Syntax

```
HTP.ANCHOR2 (
    curl           IN       VARCHAR2,
    ctext          IN       VARCHAR2,
    cname          IN       VARCHAR2   DEFAULT NULL,
    ctarget        in       varchar2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-4 ANCHOR2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the HREF attribute.</td>
</tr>
<tr>
<td>ctext</td>
<td>The string that goes between the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<A HREF="curl" NAME="cname" TARGET = "ctarget" cattributes>ctext</A>
```

216.5.4 APPLETCLOSE Procedure

This procedure closes the applet invocation with the `</APPLET>` tag.

You must first invoke the a Java applet using **APPLETOPEN Procedure**.

Syntax

```
HTP.APPLETCLOSE;
```
216.5.5 APPLETOPEN Procedure

This procedure generates the `<APPLET>` tag which begins the invocation of a Java applet.

You close the applet invocation with APPLETCLOSE Procedure which generates the `</APPLET>` tag.

Syntax

```
HTP.APPLETOPEN (  
code          IN       VARCHAR2,  
cheight        IN       NUMBER,  
cwidth         IN       NUMBER,  
cattributes    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 216-5  APPLETOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>The value for the CODE attribute which specifies the name of the applet class.</td>
</tr>
<tr>
<td>cheight</td>
<td>The value for the HEIGHT attribute.</td>
</tr>
<tr>
<td>cwidth</td>
<td>The value for the WIDTH attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<APPLET CODE="ccode" HEIGHT="cheight" WIDTH="cwidth" cattributes>
```

so that, for example,

```
HTP.appletopen('testclass.class', 100, 200, 'CODEBASE="/ows-applets"')
```

generates

```html
<APPLET CODE=""testclass.class"" height=100 width=200 CODEBASE="/ows-applets"
```

Usage Notes

- Specify parameters to the Java applet using the PARAM Procedure.
- Use the cattributes parameter to specify the CODEBASE attribute since the PL/SQL cartridge does not know where to find the class files. The CODEBASE attribute specifies the virtual path containing the class files.
216.5.6 AREA Procedure

This procedure generates the `<AREA>` tag, which defines a client-side image map. The `<AREA>` tag defines areas within the image and destinations for the areas.

Syntax

```sql
HTP.AREA ( 
    ccoords        IN       VARCHAR2
    cshape         IN       VARCHAR2   DEFAULT NULL,
    chref          IN       VARCHAR2   DEFAULT NULL,
    cnohref        IN       VARCHAR2   DEFAULT NULL,
    ctarget        IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccoords</td>
<td>The value for the COORDS attribute.</td>
</tr>
<tr>
<td>cshape</td>
<td>The value for the SHAPE attribute.</td>
</tr>
<tr>
<td>chref</td>
<td>The value for the HREF attribute.</td>
</tr>
<tr>
<td>cnohref</td>
<td>If the value for this parameter is not NULL, the NOHREF attribute is added to the tag.</td>
</tr>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget" cattributes>
```

216.5.7 BASE Procedure

This procedure generates the `<BASE>` tag which records the URL of the document.

Syntax

```sql
HTP.BASE ( 
    ctarget        IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute which establishes a window name to which all links in this document are targeted.</td>
</tr>
</tbody>
</table>
Table 216-7  (Cont.) BASE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<BASE HREF="<current URL>" TARGET="ctarget" cattributes>
```  

### 216.5.8 BASEFONT Procedure

This procedure generates the `<BASEFONT>` tag which specifies the base font size for a Web page.

**Syntax**

```lisp
HTP.BASEFONT (nsize IN INTEGER);
```

**Parameters**

Table 216-8  BASEFONT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsize</td>
<td>The value for the SIZE attribute.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<BASEFONT SIZE="nsize">
```  

### 216.5.9 BGSOUND Procedure

This procedure generates the `<BGSOUND>` tag which includes audio for a Web page.

**Syntax**

```lisp
HTP.BGSOUND (csrc           IN       VARCHAR2,
             cloop          IN       VARCHAR2   DEFAULT NULL,
             cattributes    IN       VARCHAR2   DEFAULT NULL);
```

**Parameters**

Table 216-9  BGSOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute.</td>
</tr>
</tbody>
</table>
Table 216-9  (Cont.) BGSOUND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clloop</td>
<td>The value for the LOOP attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples
This procedure generates

```html
<BGSOUND SRC="csrc" LOOP="clloop" cattributes>
```

216.5.10 BIG Procedure

This procedure generates the `<BIG>` and `</BIG>` tags which direct the browser to render the text in a bigger font.

Syntax

```sql
HTP.BIG (ctext          IN       VARCHAR2,
cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-10  BIG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that goes between the tags.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples
This procedure generates

```html
<BIG cattributes>ctext</BIG>
```

216.5.11 BLOCKQUOTECLOSE Procedure

This procedure generates the `</BLOCKQUOTE>` tag which mark the end of a section of quoted text.

You mark the beginning of a section of text by means of the BLOCKQUOTEOPEN Procedure.

Syntax

```sql
HTP.BLOCKQUOTECLOSE;
```

Examples
This procedure generates
216.5.12 BLOCKQUOTEOPEN Procedure

This procedure generates the `<BLOCKQUOTE>` tag, which marks the beginning of a section of quoted text.

You mark the end of a section of text by means of the BLOCKQUOTECLOSE Procedure.

Syntax

```sql
HTP.BLOCKQUOTEOPEN ( cnowrap        IN       VARCHAR2   DEFAULT NULL,
                    cclear         IN       VARCHAR2   DEFAULT NULL,
                    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-11  BLOCKQUOTEOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnowrap</td>
<td>If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>
```

216.5.13 BODYCLOSE Procedure

This procedure generates the `</BODY>` tag which marks the end of a body section of an HTML document.

You mark the beginning of a body section by means of the BODYOPEN Procedure.

Syntax

```sql
HTP.BODYCLOSE;
```

Examples

This procedure generates

```html
</BODY>
```
216.5.14 BODYOPEN Procedure

This procedure generates the `<BODY>` tag which marks the beginning of the body section of an HTML document.

You mark the end of a body section by means of the BODYCLOSE Procedure.

Syntax

```plaintext
HTP.BODYOPEN ( cbackground IN VARCHAR2 DEFAULT NULL, cattributes IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbackground</td>
<td>The value for the BACKGROUND attribute which specifies a graphic file to use for the background of the document.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```plaintext
<BODY background="cbackground" cattributes>
```

so that

```plaintext
HTP.BODYOPEN('/img/background.gif');
```

generates:

```plaintext
<BODY background="/img/background.gif">
```

216.5.15 BOLD Procedure

This procedure generates the `<B>` and `</B>` tags which direct the browser to display the text in boldface.

Syntax

```plaintext
HTP.BOLD ( ctext IN VARCHAR2, cattributes IN VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that goes between the tags.</td>
</tr>
</tbody>
</table>
Table 216-13  (Cont.) BOLD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<B cattributes>ctext</B>
```

### 216.5.16 BR Procedure

This procedure generates the `<BR>` tag which begins a new line of text.

It performs the same operation as the NL Procedure.

Syntax

```
HTP.BR(
    cclear         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-14  BR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<BR CLEAR="cclear" cattributes>
```

### 216.5.17 CENTER Procedure

This procedure generates the `<CENTER>` and `</CENTER>` tags which center a section of text within a Web page.

Syntax

```
HTP.CENTER (   ctext          IN       VARCHAR2);
```
Parameters

Table 216-15  CENTER Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that goes between the tags.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

\(<\text{CENTER}>\text{ctext}</\text{CENTER}>\)

216.5.18 CENTERCLOSE Procedure

This procedure generates the <\text{CENTER}> tag which marks the end of a section of text to center.

You mark the beginning of a section of text to center with the CENTEROPEN Procedure.

Syntax

\(\text{HTP.CENTERCLOSE;}\)

Examples

This procedure generates

\(<\text{CENTER}>\)

216.5.19 CENTEROPEN Procedure

This procedure generates the <\text{CENTER}> tag which mark the beginning of a section of text to center.

You mark the beginning of a of a section of text to center by means of the CENTERCLOSE Procedure.

Syntax

\(\text{HTP.CENTEROPEN;}\)

Examples

This procedure generates

\(<\text{CENTER}>\)
216.5.20 CITE Procedure

This procedure generates the `<CITE>` and `</CITE>` tags which direct the browser to render the text as a citation.

**Syntax**

```
HTP.CITE (ctext IN VARCHAR2,
cattributes IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render as citation.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This procedure generates

```
<CITE cattributes>ctext</CITE>
```

216.5.21 CODE Procedure

This procedure generates the `<CODE>` and `</CODE>` tags which direct the browser to render the text in monospace font or however "code" is defined stylistically.

**Syntax**

```
HTP.CODE (ctext IN VARCHAR2,
cattributes IN VARCHAR2 DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render as code.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag</td>
</tr>
</tbody>
</table>

**Examples**

This procedure generates

```
<CODE cattributes>ctext</CODE>
```
216.5.22 COMMENT Procedure

This procedure generates the comment tags.

Syntax

```sql
HTP.COMMENT (ctext IN VARCHAR2);
```

Parameters

Table 216-18 COMMENT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The comment.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

`<!-- ctext -->`

216.5.23 DFN Procedure

This procedure generates the `<DFN>` and `</DFN>` tags which direct the browser to mark the text in italics or however "definition" is described stylistically.

Syntax

```sql
HTP.DFN (ctext IN VARCHAR2);
```

Parameters

Table 216-19 DFN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in italics.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

`<DFN>ctext</DFN>`

216.5.24 DIRLISTCLOSE Procedure

This procedure generates the `<DIR>` tag which ends a directory list section. You start a directory list section with the DIRLISTOPEN Procedure.

Syntax

```sql
HTP.DIRLISTCLOSE;
```
Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Procedure so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This procedure generates

</DIR>

Related Topics

• DIRLISTOPEN Procedure
  This procedure generates the <DIR> which starts a directory list section. You end a directory list section with the DIRLISTCLOSE Procedure.

216.5.25 DIRLISTOPEN Procedure

This procedure generates the <DIR> which starts a directory list section. You end a directory list section with the DIRLISTCLOSE Procedure.

Syntax

HTP.DIRLISTOPEN;

Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the LISTITEM Procedure so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

Examples

This procedure generates

<DIR>

Related Topics

• DIRLISTCLOSE Procedure
  This procedure generates the </DIR> tag which ends a directory list section. You start a directory list section with the DIRLISTOPEN Procedure.

216.5.26 DIV Procedure

This procedure generates the <DIV> tag which creates document divisions.

Syntax

HTP.DIV (  
calign IN VARCHAR2 DEFAULT NULL,  
cattributes IN VARCHAR2 DEFAULT NULL);

ORACLE
Parameters

Table 216-20   DIV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<DIV ALIGN="calign" cattributes>
```

216.5.27 DLISTCLOSE Procedure

This procedure generates the `</DL>` tag which ends a definition list. You start a definition list by means of the DLISTOPEN Procedure.

Syntax

```sql
HTP.DLISTCLOSE;
```

Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Procedure and definitions are inserted using the DLISTDEF Procedure.

Examples

This procedure generates

```html
</DL>
```

Related Topics

- DLISTOPEN Procedure
  This procedure generates the `</DL>` tag which starts a definition list. You end a definition list by means of the DLISTCLOSE Procedure.

216.5.28 DLISTDEF Procedure

This procedure generates the `<DD>` tag, which inserts definitions of terms. Use this tag for a definition list `<DL>`. Terms are tagged `<DT>` and definitions are tagged `<DD>`.

Syntax

```sql
HTP.DLISTDEF ( 
  ctext          IN       VARCHAR2   DEFAULT NULL,
  cclear         IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-21  DLISTDEF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The definition of the term.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<DD CLEAR="cclear" cattributes>ctext
```

216.5.29 DLISTOPEN Procedure

This procedure generates the `<DL>` tag which starts a definition list. You end a definition list by means of the DLISTCLOSE Procedure.

Syntax

```java
HTP.DLISTOPEN ( cclear IN VARCHAR2 DEFAULT NULL, cattributes IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 216-22  DLISTOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the DLISTTERM Procedure and definitions are inserted using the DLISTDEF Procedure.

Examples

This procedure generates

```html
<DL CLEAR="cclear" cattributes>
```

Related Topics

- DLISTCLOSE Procedure
  This procedure generates the `</DL>` tag which ends a definition list. You start a definition list by means of the DLISTOPEN Procedure.
216.5.30 DLISTTERM Procedure

This procedure generates the `<DT>` tag which defines a term in a definition list `<DL>`.

**Syntax**

```sql
HTP.DLISTTERM (  
    ctext          IN       VARCHAR2   DEFAULT NULL,  
    cclear         IN       VARCHAR2   DEFAULT NULL,  
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The term.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This procedure generates

```html
<DT CLEAR="cclear" cattributes>ctext
```

216.5.31 EM Procedure

This procedure generates the `<EM>` and `</EM>` tags, which define text to be emphasized.

It performs the same task as the EMPHASIS Procedure.

**Syntax**

```sql
HTP.EM(  
    ctext          IN       VARCHAR2,  
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to emphasize.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This procedure generates

```html
<EM cattributes>ctext</EM>
```
216.5.32 EMPHASIS Procedure

This procedure generates the `<EM>` and `</EM>` tags, which define text to be emphasized.

It performs the same task as the EM Procedure.

Syntax

```sql
HTP.EMPHASIS(
    ctext        IN       VARCHAR2,
    cattributes  IN       VARCHAR2 DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to emphasize.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<EM cattributes>ctext</EM>
```

216.5.33 ESCAPE_SC Procedure

This procedure replaces characters that have special meaning in HTML with their escape sequences.

The following characters are converted:

- `&` to `&amp;`
- `"` to `&quot;`
- `<` to `&lt;`
- `>` to `&gt;`

This procedure performs the same operation as PRINTS Procedures and PS Procedure.

Syntax

```sql
HTP.ESCAPE_SC(
    ctext        IN       VARCHAR2);
```
216.5.34 FONTCLOSE Procedure

This procedure generates the \</\texttt{FONT}\> tag which marks the end of a section of text with the specified font characteristics.

You mark the beginning of the section text by means of the FONTPOPEN Procedure.

Syntax

\begin{verbatim}
HTP.FONTCLOSE;
\end{verbatim}

Examples

This procedure generates

\begin{verbatim}
</\texttt{FONT}>
\end{verbatim}

216.5.35 FONTPOPEN Procedure

This procedure generates the \texttt{<FONT>} which marks the beginning of section of text with the specified font characteristics.

You mark the end of the section text by means of the FONTCLOSE Procedure.

Syntax

\begin{verbatim}
HTP.FONTOPEN(
    ccolor         IN       VARCHAR2   DEFAULT NULL,
    cface          IN       VARCHAR2   DEFAULT NULL,
    csize          IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
\end{verbatim}

Parameters

Table 216-27  FONTPOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccolor</td>
<td>The value for the \texttt{COLOR} attribute.</td>
</tr>
<tr>
<td>cface</td>
<td>The value for the \texttt{FACE} attribute</td>
</tr>
<tr>
<td>csize</td>
<td>The value for the \texttt{SIZE} attribute</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates
216.5.36 FORMCHECKBOX Procedure

This procedure generates the `<INPUT>` tag with TYPE="checkbox" which inserts a checkbox element in a form.

A checkbox element is a button that the user toggles on or off.

Syntax

```
HTP.FORMCHECKBOX(
  cname          IN       VARCHAR2,
  cvalue         IN       VARCHAR2   DEFAULT 'ON',
  cchecked       IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

**Table 216-28**  FORMCHECKBOX Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cchecked</td>
<td>If the value for this parameter is not NULL, the CHECKED attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

216.5.37 FORMCLOSE Procedure

This procedure generates the `</FORM>` tag which marks the end of a form section in an HTML document.

You mark the beginning of the form section by means of the FORMOPEN Procedure.

Syntax

```
HTP.FORMCLOSE;
```

Examples

This procedure generates

```
</FORM>
```
216.5.38 FORMOPEN Procedure

This procedure generates the `<FORM>` tag which marks the beginning of a form section in an HTML document.

You mark the end of the form section by means of the FORMCLOSE Procedure.

Syntax

```
HTP.FORMOPEN(
    curl           IN      VARCHAR2,
    cmethod        IN      VARCHAR2   DEFAULT 'POST',
    ctarget        IN      VARCHAR2   DEFAULT NULL,
    cenctype       IN      VARCHAR2   DEFAULT NULL,
    cattributes    IN      VARCHAR2   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The URL of the WRB or CGI script where the contents of the form is sent. This parameter is required.</td>
</tr>
<tr>
<td>cmethod</td>
<td>The value for the METHOD attribute. The value can be &quot;GET&quot; or &quot;POST&quot;.</td>
</tr>
<tr>
<td>ctarget</td>
<td>The value for the TARGET attribute.</td>
</tr>
<tr>
<td>cenctype</td>
<td>The value for the ENCTYPE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarget" ENCTYPE="cenctype" cattributes>
```

216.5.39 FORMFILE Procedure

This procedure generates the `<INPUT>` tag with TYPE="file" which inserts a file form element. This is used for file uploading for a given page.

Syntax

```
HTP.FORMFILE(
    cname          IN      VARCHAR2,
    caccept        IN      VARCHAR2   DEFAULT NULL,
    cattributes    IN      VARCHAR2   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
</tbody>
</table>
Table 216-30  (Cont.) FORMFILE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caccept</td>
<td>A comma-delimited list of MIME types for upload.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<INPUT TYPE="file" NAME="cname" ACCEPT="caccept" cattributes> 
```

216.5.40 FORMHIDDEN Procedure

This procedure generates the `<INPUT>` tag with TYPE="hidden", which inserts a hidden form element.

This element is not seen by the user. It submits additional values to the script.

Syntax

```sql
HTP.FORMHIDDEN(
    cname          IN       VARCHAR2,
    cvalue         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-31  FORMHIDDEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes> 
```

216.5.41 FORMIMAGE Procedure

This procedure generates the `<INPUT>` tag with TYPE="image" which creates an image field that the user clicks to submit the form immediately.

The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/value pairs. The x coordinate is submitted under the name of the field with .x appended, and the y coordinate with .y appended. Any VALUE attribute is ignored.
Syntax

HTP.FORMIMAGE(
    cname          IN       VARCHAR2,
    csrc           IN       VARCHAR2,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);

Parameters

Table 216-32  FORMIMAGE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute that specifies the image file.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

216.5.42 FORMPASSWORD Procedure

This procedure generates the `<INPUT>` tag with `TYPE="password"` which creates a single-line text entry field. When the user enters text in the field, each character is represented by one asterisk. This is used for entering passwords.

Syntax

HTP.FORMPASSWORD(
    cname          IN       VARCHAR2,
    csize          IN       VARCHAR2,
    cmaxlength     IN       VARCHAR2   DEFAULT NULL,
    cvalue         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);

Parameters

Table 216-33  FORMPASSWORD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>csize</td>
<td>The value for the SIZE attribute.</td>
</tr>
<tr>
<td>cmaxlength</td>
<td>The value for the MAXLENGTH attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples

This procedure generates

```html
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength" VALUE="cvalue" cattributes>
```

216.5.43 FORMRADIO Procedure

This procedure generates the `<INPUT>` tag with `TYPE="radio"`, which creates a radio button on the HTML form. Within a set of radio buttons, the user selects only one. Each radio button in the same set has the same name, but different values. The selected radio button generates a name/value pair.

Syntax

```sql
HTP.FORMRADIO(
  cname          IN       VARCHAR2,
  cvalue         IN       VARCHAR2,
  cchecked       IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

**Table 216-34** FORMRADIO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cchecked</td>
<td>If the value for this parameter is not NULL, the CHECKED attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

216.5.44 FORMRESET Procedure

This procedure generates the `<INPUT>` tag with `TYPE="reset"` which creates a button that, when selected, resets the form fields to their initial values.

Syntax

```sql
HTP.FORMRESET(
  cvalue         IN       VARCHAR2   DEFAULT 'Reset',
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-35  FORMRESET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

216.5.45 FORMSELECTCLOSE Procedure

This procedure generates the `</SELECT>` tag which marks the end of a Select form element.

A Select form element is a listbox where the user selects one or more values. You mark the beginning of Select form element by means of the FORMSELECTOPEN Procedure. The values are inserted using FORMSELECTOPTION Procedure.

Syntax

```sql
HTP.FORMSELECTCLOSE;
```

Examples

This procedure generates

```html
</SELECT>
```

as shown under Examples of the FORMSELECTOPEN Procedure.

216.5.46 FORMSELECTOPEN Procedure

This procedure generates the `<SELECT>` tags which creates a Select form element.

A Select form element is a listbox where the user selects one or more values. You mark the end of Select form element by means of the FORMSELECTCLOSE Procedure. The values are inserted using FORMSELECTOPTION Procedure.

Syntax

```sql
FORMSELECTOPEN(
    cname          IN       VARCHAR2,
    cprompt        IN       VARCHAR2   DEFAULT NULL,
    nsize          IN       INTEGER    DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-36  FORMSELECTOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cprompt</td>
<td>The string preceding the list box.</td>
</tr>
<tr>
<td>nsize</td>
<td>The value for the SIZE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```sql
cprompt <SELECT NAME="cname" SIZE="nsize" cattributes>
</SELECT>
```

so that

```sql
HTP.FORMSELECTOPEN('greatest_player';
  'Pick the greatest player:');
HTP.FORMSELECTOPTION('Messier');
HTP.FORMSELECTOPTION('Howe');
HTP.FORMSELECTOPTION('Gretzky');.
HTP.FORMSELECTCLOSE;
```

generates

```
Pick the greatest player:
<SELECT NAME="greatest_player">
  <OPTION>Messier
  <OPTION>Howe
  <OPTION>Gretzky
</SELECT>
```

216.5.47 FORMSELECTOPTION Procedure

This procedure generates the `<OPTION>` tag which represents one choice in a Select element.

Syntax

```sql
HTP.FORMSELECTOPTION(
  cvalue         IN       VARCHAR2,
  cselected      IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-37  FORMSELECTOPTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The text for the option.</td>
</tr>
</tbody>
</table>
Table 216-37  (Cont.) FORMSELECTOPTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>If the value for this parameter is not NULL, the SELECTED attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<OPTION SELECTED cattributes>cvalue
```

as shown under Examples of the FORMSELECTOPEN Procedure.

216.5.48 FORMSUBMIT Procedure

This procedure generates the `<INPUT>` tag with `TYPE="submit"` which creates a button that, when clicked, submits the form. If the button has a `NAME` attribute, the button contributes a name/value pair to the submitted data.

Syntax

```sql
HTP.FORMSUBMIT(
    cname          IN       VARCHAR2   DEFAULT NULL,
    cvalue        IN       VARCHAR2   DEFAULT 'Submit',
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-38  FORMSUBMIT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the <code>NAME</code> attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the <code>VALUE</code> attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

216.5.49 FORMTEXT Procedure

This procedure generates the `<INPUT>` tag with `TYPE="text"`, which creates a field for a single line of text.

Syntax

```sql
HTP.FORMTEXT(
    cname          IN       VARCHAR2,
```

Chapter 216

Summary of HTP Subprograms
Parameters

Table 216-39  FORMTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>csize</td>
<td>The value for the SIZE attribute.</td>
</tr>
<tr>
<td>cmaxlength</td>
<td>The value for the MAXLENGTH attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

216.5.50 FORMTEXTAREA Procedure

This procedure generates the `<TEXTAREA>` tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text.

The same operation is performed by the FORMTEXTAREA2 Procedure which in addition has the cwrap parameter that lets you specify a wrap style.

Syntax

```sql
HTP.FORMTEXTAREA(
    cname          IN       VARCHAR2,
    nrows          IN       INTEGER,
    ncolumns       IN       INTEGER,
    calign      , IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-40  FORMTEXTAREA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the ROWS attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the COLS attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
216.5.51 FORMTEXTAREA2 Procedure

This procedure generates the `<TEXTAREA>` tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text.

The same operation is performed by the FORMTEXTAREA Procedure except that in that case you cannot specify a wrap style.

Syntax

```
HTP.FORMTEXTAREA2(
   cname          IN       VARCHAR2,
   nrows          IN       INTEGER,
   ncolumns       IN       INTEGER,
   calign         IN       VARCHAR2   DEFAULT NULL,
   cwrap          IN       VARCHAR2   DEFAULT NULL,
   cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the ROWS attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the COLS attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the WRAP attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes"></TEXTAREA>
```

216.5.52 FORMTEXTAREACLOSE Procedure

This procedure generates the `</TEXTAREA>` tag which ends a text area form element.

You open a text area element by means of either FORMTEXTAREAOPEN Procedure or FORMTEXTAREAOPEN2 Procedure.

Syntax

```
HTP.FORMTEXTAREACLOSE;
```
Examples
This procedure generates
</TEXTAREA>

216.5.53 FORMTEXTAREAOPEN Procedure

This procedure generates the <TEXTAREA> which marks the beginning of a text area form element.

The same operation is performed by the FORMTEXTAREAOPEN2 Procedure which in addition has the cwrap parameter that lets you specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Procedure.

Syntax

HTP.FORMTEXTAREAOPEN(
  cname          IN       VARCHAR2,
  nrows          IN       INTEGER,
  ncolumns       IN       INTEGER,
  calign         IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);

Parameters

Table 216-42 FORMTEXTAREAOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the ROWS attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the COLS attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples
This procedure generates

<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes>

216.5.54 FORMTEXTAREAOPEN2 Procedure

This procedure generates the <TEXTAREA> which marks the beginning of a text area form element.

The same operation is performed by the FORMTEXTAREAOPEN Procedure except that in that case you cannot specify a wrap style. You mark the end of a text area form element by means of the FORMTEXTAREACLOSE Procedure.

Syntax

HTP.FORMTEXTAREAOPEN2(
  cname        IN       VARCHAR2,
  cwrap        IN       VARCHAR2 DEFAULT NULL,
  cattributes  IN       VARCHAR2 DEFAULT NULL);
nrows IN INTEGER, ncolumns IN INTEGER, calign IN VARCHAR2 DEFAULT NULL, cwrap IN VARCHAR2 DEFAULT NULL, cattributes IN VARCHAR2 DEFAULT NULL);

Parameters

Table 216-43 FORMTEXAREAOPEN2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctextareaopenname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>nrows</td>
<td>The value for the ROWS attribute. This is an integer.</td>
</tr>
<tr>
<td>ncolumns</td>
<td>The value for the COLS attribute. This is an integer.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the WRAP attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<TEXAREA NAME="ctextareaopenname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP = "cwrap" cattributes>
```

216.5.55 FRAME Procedure

This procedure generates the `<FRAME>` tag which begins the characteristics of a frame created by a `<FRAMESET>` tag.

Syntax

```sql
HTP.FRAME( csrcref IN VARCHAR2,
    ctextareaopenname IN VARCHAR2 DEFAULT NULL,
    cmarginwidth IN VARCHAR2 DEFAULT NULL,
    cmarginheight IN VARCHAR2 DEFAULT NULL,
    cscrolling IN VARCHAR2 DEFAULT NULL,
    cnoresize IN VARCHAR2 DEFAULT NULL,
    cattributes IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 216-44 FRAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csrcref</td>
<td>The URL to display in the frame.</td>
</tr>
<tr>
<td>ctextareaopenname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cmarginwidth</td>
<td>The value for the MARGINWIDTH attribute.</td>
</tr>
<tr>
<td>cscrolling</td>
<td>The value for the SCROLLING attribute.</td>
</tr>
</tbody>
</table>
### Table 216-44 (Cont.) FRAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnoresize</td>
<td>If the value for this parameter is not NULL, the NORESIZE attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

### Examples

This procedure generates

```html
<FRAME SRC="csrc" NAME="cname" MARGINWIDTH="cmarginwidth"
MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE cattributes>
```

### 216.5.56 FRAMESETCLOSE Procedure

This procedure generates the `</FRAMESET>` tag which ends a frameset section.

You mark the beginning of a frameset section by means of the FRAMESETOPEN Procedure.

**Syntax**

```sql
HTP.FRAMESETCLOSE;
```

**Examples**

This procedure generates

`</FRAMESET>`

### 216.5.57 FRAMESETOPEN Procedure

This procedure generates the `<FRAMESET>` tag which define a frameset section.

You mark the end of a frameset section by means of the FRAMESETCLOSE Procedure.

**Syntax**

```sql
HTP.FRAMESETOPEN(
  crows          IN       VARCHAR2   DEFAULT NULL,
  ccols          IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crows</td>
<td>The value for the ROWS attribute.</td>
</tr>
<tr>
<td>ccols</td>
<td>The value for the COLS attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>
Examples

This procedure generates

```html
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

216.5.58 HEADCLOSE Procedure

This procedure generates the `</HEAD>` tag which marks the end of an HTML document head section.

You mark the beginning of an HTML document head section by means of the HEADOPEN Procedure.

Syntax

```jsp
HTP.HEADCLOSE;
```

Examples

This procedure generates

```html
</HEAD>
```

216.5.59 HEADER Procedure

This procedure generates opening heading tags (`<H1>` to `<H6>`) and their corresponding closing tags (`</H1>` to `/</H6>`).

Syntax

```jsp
HTP.HEADER(
    nsize          IN       INTEGER,
    cheader        IN       VARCHAR2,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cnowrap        IN       VARCHAR2   DEFAULT NULL,
    cclear         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-46  HEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsize</td>
<td>The heading level. This is an integer between 1 and 6.</td>
</tr>
<tr>
<td>cheader</td>
<td>The text to display in the heading.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>The value for the NOWRAP attribute.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

```jsp
HTP.header (1,'Overview');
```
216.5.60 HEADOPEN Procedure

This procedure generates the `<HEAD>` tag which marks the beginning of the HTML document head section.

You mark the end of an HTML document head section by means of the HEADCLOSE Procedure.

Syntax

```
HTP.HEADOPEN;
```

Examples

This procedure generates

```
<HEAD>
```

216.5.61 HR Procedure

This procedure generates the `<HR>` tag, which generates a line in the HTML document.

This subprogram performs the same operation as the LINE Procedure.

Syntax

```
HTP.HR(
    cclear         IN       VARCHAR2   DEFAULT NULL,
    csrc           IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-47   HR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute which specifies a custom image as the source of the line.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```
216.5.62 HTMLCLOSE Procedure

This procedure generates the `<HTML>` tag which marks the end of an HTML document.

You use the HTMLOPEN Procedure to mark the beginning of an HTML document.

Syntax

```
HTP.HTCLOSE;
```

Examples

This procedure generates

```
</HTML>
```

216.5.63 HTMLOPEN Procedure

This procedure generates the `<HTML>` tag which marks the beginning of an HTML document.

You use the HTMLCLOSE Procedure to mark the end of the an HTML document.

Syntax

```
HTP.HTMLOPEN;
```

Examples

This procedure generates

```
<HTML>
```

216.5.64 IMG Procedure

This procedure generates the `<IMG>` tag which directs the browser to load an image onto the HTML page.

The IMG2 Procedure performs the same operation but additionally uses the cusemap parameter.

Syntax

```
HTP.IMG(
    curl           IN       VARCHAR2   DEFAULT NULL,
    calign         IN       VARCHAR2   DEFAULT NULL,
    calt           IN       VARCHAR2   DEFAULT NULL,
    cismap         IN       VARCHAR2  DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-48  IMG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the SRC attribute.</td>
</tr>
</tbody>
</table>
Table 216-48  (Cont.) IMG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>calt</td>
<td>The value for the ALT attribute which specifies alternative text to display if the browser does not support images.</td>
</tr>
<tr>
<td>cismap</td>
<td>If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

216.5.65 IMG2 Procedure

This procedure generates the `<IMG>` tag, which directs the browser to load an image onto the HTML page.

The IMG Procedure performs the same operation but does not use the cusemap parameter.

Syntax

```sql
HTP.IMG2(
    curl           IN       VARCHAR2   DEFAULT NULL,
    calign         IN       VARCHAR2   DEFAULT NULL,
    calt           IN       VARCHAR2   DEFAULT NULL,
    cismap         IN       VARCHAR2  DEFAULT NULL,
    cusemap        IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-49  IMG2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>calt</td>
<td>The value for the ALT attribute which specifies alternative text to display if the browser does not support images.</td>
</tr>
<tr>
<td>cismap</td>
<td>If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.</td>
</tr>
<tr>
<td>cusemap</td>
<td>The value for the USEMAP attribute which specifies a client-side image map.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates
216.5.66 ISINDEX Procedure

This procedure creates a single entry field with a prompting text, such as "enter value," then sends that value to the URL of the page or program.

Syntax

```
HTP.ISINDEX(
    cprompt        IN       VARCHAR2    DEFAULT NULL,
    curl           IN       VARCHAR2    DEFAULT NULL);
```

Parameters

**Table 216-50  ISINDEX Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cprompt</td>
<td>The value for the PROMPT attribute.</td>
</tr>
<tr>
<td>curl</td>
<td>The value for the HREF attribute.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<ISINDEX PROMPT="cprompt" HREF="curl">
```

216.5.67 ITALIC Procedure

This procedure generates the `<I>` and `</I>` tags which direct the browser to render the text in italics.

Syntax

```
HTP.ITALIC(
    ctext          IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

**Table 216-51  ITALIC Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in italics.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<I cattributes>ctext</I>
```
216.5.68 KBD Procedure

This procedure generates the `<KBD>` and `</KBD>` tags which direct the browser to render the text in monospace font.

This subprogram performs the same operation as the KEYBOARD Procedure.

Syntax

```sql
HTP.KBD(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-52  KBD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in monospace.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<KBD cattributes>ctext</KBD>
```

216.5.69 KEYBOARD Procedure

This procedure generates the `<KBD>` and `</KBD>` tags, which direct the browser to render the text in monospace font.

This subprogram performs the same operation as the KBD Procedure.

Syntax

```sql
HTP.KEYBOARD(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-53  KEYBOARD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in monospace.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<KBD cattributes>ctext</KBD>
```
216.5.70 LINE Procedure

This procedure generates the `<HR>` tag, which generates a line in the HTML document.

This subprogram performs the same operation as the HR Procedure.

Syntax

```
HTP.LINE(
    cclear         IN       VARCHAR2   DEFAULT NULL,
    csrc           IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-54  LINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute which specifies a custom image as the source of the line.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

216.5.71 LINKREL Procedure

This procedure generates the `<LINK>` tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present.

This is the opposite of LINKREV Procedure. This tag indicates a relationship between documents but does not create a link. To create a link, use the ANCHOR Procedure.

Syntax

```
HTP.LINKREL(
    crel           IN       VARCHAR2,
    curl           IN       VARCHAR2,
    ctitle         IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-55  LINKREL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crel</td>
<td>The value for the REL attribute.</td>
</tr>
<tr>
<td>curl</td>
<td>The value for the URL attribute.</td>
</tr>
</tbody>
</table>
Table 216-55  (Cont.) LINKREL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctitle</td>
<td>The value for the TITLE attribute.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<LINK REL="crel" HREF="curl" TITLE="ctitle">
```

216.5.72 LINKREV Procedure

This procedure generates the `<LINK>` tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor.

This is the opposite of the LINKREL Procedure. This tag indicates a relationship between documents, but does not create a link. To create a link, use the ANCHOR Procedure.

Syntax

```
HTP.LINKREV(
    crel           IN       VARCHAR2,
    curl           IN       VARCHAR2,
    ctitle         IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-56  LINKREV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crel</td>
<td>The value for the REV attribute.</td>
</tr>
<tr>
<td>curl</td>
<td>The value for the URL attribute.</td>
</tr>
<tr>
<td>ctitle</td>
<td>The value for the TITLE attribute.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<LINK REV="crel" HREF="curl" TITLE="ctitle">
```

216.5.73 LISTHEADER Procedure

This procedure generates the `<LH>` and `</LH>` tags which print an HTML tag at the beginning of the list.

Syntax

```
HTP.LISTHEADER(
    ctext          IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-57  LISTHEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to place between &lt;LH&gt; and &lt;/LH&gt;.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<LH cattributes>ctext</LH>
```

216.5.74 LISTINGCLOSE Procedure

This procedure generates the `</LISTING>` tags which marks the end of a section of fixed-width text in the body of an HTML page.

To mark the beginning of a section of fixed-width text in the body of an HTML page, use the LISTINGOPEN Procedure.

Syntax

```html
HTP.LISTINGCLOSE;
```

Examples

This procedure generates

```
</LISTING>
```

216.5.75 LISTINGOPEN Procedure

This procedure generates the `<LISTING>` tag which marks the beginning of a section of fixed-width text in the body of an HTML page.

To mark the end of a section of fixed-width text in the body of an HTML page, use the LISTINGCLOSE Procedure.

Syntax

```html
HTP.LISTINGOPEN;
```

Examples

This procedure generates

```html
<Listing>
```
216.5.76 LISTITEM Procedure

This procedure generates the `<LI>` tag, which indicates a list item.

Syntax

```
HTP.LISTITEM(
  ctext          IN       VARCHAR2   DEFAULT NULL,
  cclear         IN       VARCHAR2   DEFAULT NULL,
  cdingbat       IN       VARCHAR2   DEFAULT NULL,
  csrc           IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-58  LISTITEM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text for the list item.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cdingbat</td>
<td>The value for the DINGBAT attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext
```

216.5.77 MAILTO Procedure

This procedure generates the `<A>` tag with the HREF set to 'mailto' prepended to the mail address argument.

Syntax

```
HTP.MAILTO(
  caddress       IN       VARCHAR2,
  ctext          IN       VARCHAR2,
  cname          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-59  MAILTO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caddress</td>
<td>The email address of the recipient.</td>
</tr>
<tr>
<td>ctext</td>
<td>The clickable portion of the link.</td>
</tr>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
</tbody>
</table>
Table 216-59  (Cont.) MAILTO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<A HREF="mailto:caddress" NAME="cname" cattributes>ctext</A>
```

so that

```
HTP.mailto('pres@white_house.gov','Send Email to the President');
```

generates:

```
<A HREF="mailto:pres@white_house.gov">Send Email to the President</A>
```

216.5.78 MAPCLOSE Procedure

This procedure generates the `</MAP>` tag which marks the end of a set of regions in a client-side image map.

To mark the beginning of a set of regions in a client-side image map, use the MAPOPEN Procedure.

Syntax

```
HTP.MAPCLOSE;
```

Examples

This procedure generates

```
</MAP>
```

216.5.79 MAPOPEN Procedure

This procedure generates the `<MAP>` tag which mark the beginning of a set of regions in a client-side image map.

To mark the end of a set of regions in a client-side image map, use the MAPCLOSE Procedure.

Syntax

```
HTP.MAPOPEN(
  cname          IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-60 MAPOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<MAP NAME="cname" cattributes>
```

216.5.80 MENULISTCLOSE Procedure

This procedure generates the `</MENU>` tag which ends a list that presents one line for each item.

To begin a list of this kind, use the MENULISTOPEN Procedure. The items in the list appear more compact than an unordered list. The LISTITEM Procedure defines the list items in a menu list.

Syntax

```
HTP.MENULISTCLOSE;
```

Examples

This procedure generates

```
</MENU>
```

216.5.81 MENULISTOPEN Procedure

This procedure generates the `<MENU>` tag which begins a list that presents one line for each item.

To end a list of this kind, use the MENULISTCLOSE Procedure. The items in the list appear more compact than an unordered list. The LISTITEM Procedure defines the list items in a menu list.

Syntax

```
HTP.MENULISTOPEN;
```

Examples

This procedure generates

```
</MENU>
```
216.5.82 META Procedure

This procedure generates the `<META>` tag, which embeds meta-information about the document and also specifies values for HTTP headers. For example, you can specify the expiration date, keywords, and author name.

**Syntax**

```sql
HTP.META(
    chttp_equiv    IN       VARCHAR2,
    cname          IN       VARCHAR2,
    ccontent       IN       VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chttp_equiv</code></td>
<td>The value for the <code>CHTTP_EQUIV</code> attribute.</td>
</tr>
<tr>
<td><code>cname</code></td>
<td>The value for the <code>NAME</code> attribute.</td>
</tr>
<tr>
<td><code>ccontent</code></td>
<td>The value for the <code>CONTENT</code> attribute.</td>
</tr>
</tbody>
</table>

**Examples**

This procedure generates

```html
<META HTTP-EQUIV="chttp_equiv" NAME="cname" CONTENT="ccontent">
```

so that

```sql
HTP.meta ('Refresh', NULL, 120);
```

generates

```html
<META HTTP-EQUIV="Refresh" CONTENT="120">
```

On some Web browsers, this causes the current URL to be reloaded automatically every 120 seconds.

216.5.83 NL Procedure

This procedure generates the `<BR>` tag which begins a new line of text.

It performs the same operation as the BR Procedure.

**Syntax**

```sql
HTP.NL(
    cclear         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-62  NL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples
This procedure generates
<pre>&lt;BR CLEAR="cclear" cattributes&gt;</pre>

216.5.84 NOBR Procedure

This procedure generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text.

Syntax

HTP.NOBR(
ctext        IN        VARCHAR2);

Parameters

Table 216-63  NOBR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text that is to be rendered on one line.</td>
</tr>
</tbody>
</table>

Examples
This procedure generates

<NOBR>ctext</NOBR>

216.5.85 NOFRAMESCLOSE Procedure

This procedure generates the </NOFRAMES> tag which marks the end of a no-frames section.

To mark the beginning of a no-frames section, use the FRAMESETOPEN Procedure. See also FRAME Procedure, FRAMESETOPEN Procedure and FRAMESETCLOSE Procedure.

Syntax

HTP.NOFRAMESCLOSE;

Examples
This procedure generates
216.5.86 NOFRAMESOPEN Procedure

This procedure generates the `<NOFRAMES>` tag which mark the beginning of a no-frames section.

To mark the end of a no-frames section, use the FRAMESETCLOSE Procedure. See also FRAME Procedure, FRAMESETOPEN Procedure and FRAMESETCLOSE Procedure.

Syntax

```
HTP.NOFRAMESOPEN;
```

Examples

This procedure generates

```
<NOFRAMES>
```

216.5.87 OLISTCLOSE Procedure

This procedure generates the `</OL>` tag which defines the end of an ordered list. An ordered list presents a list of numbered items.

To mark the beginning of a list of this kind, use the OLISTOPEN Procedure. Numbered items are added using LISTITEM Procedure.

Syntax

```
HTP.OLISTCLOSE;
```

Examples

This procedure generates

```
</OL>
```

216.5.88 OLISTOPEN Procedure

This procedure generates the `<OL>` tag which marks the beginning of an ordered list. An ordered list presents a list of numbered items.

To mark the end of a list of this kind, use the OLISTCLOSE Procedure. Numbered items are added using LISTITEM Procedure.

Syntax

```
HTP.OLISTOPEN(cclear IN VARCHAR2 DEFAULT NULL, cwrap IN VARCHAR2 DEFAULT NULL, cattributes IN VARCHAR2 DEFAULT NULL);
```
Parameters

Table 216-64  OLISTOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the WRAP attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

216.5.89 PARA Procedure

This procedure generates the `<P>` tag which indicates that the text that comes after the tag is to be formatted as a paragraph.

You can add attributes to the tag by means of the PARAGRAPH Procedure.

Syntax

```sql
HTP.PARA;
```

Examples

This procedure generates

```html
<P>
```

216.5.90 PARAGRAPH Procedure

You can use this procedure to add attributes to the `<P>` tag created by the PARA Procedure.

Syntax

```sql
HTP.PARAGRAPH(
    calign         IN       VARCHAR2   DEFAULT NULL,
    cnowrap        IN       VARCHAR2   DEFAULT NULL,
    cclear         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-65  PARAGRAPH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
</tbody>
</table>
Table 216-65  (Cont.) PARAGRAPH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnowrap</td>
<td>If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples
This procedure generates

```html
<P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

Related Topics
- PARA Procedure
  This procedure generates the `<P>` tag which indicates that the text that comes after the tag is to be formatted as a paragraph.

216.5.91 PARAM Procedure
This procedure generates the `<PARAM>` tag which specifies parameter values for Java applets.

The values can reference HTML variables. To invoke a Java applet from a Web page, use APPLETOPEN Procedure to begin the invocation. Use one PARAM Procedure for each desired name-value pair, and use APPLETCLOSE Procedure to end the applet invocation.

Syntax

```sql
HTP.PARAM(
    cname          IN       VARCHAR2
    cvalue         IN       VARCHAR2);
```

Parameters

Table 216-66  PARAM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The value for the NAME attribute.</td>
</tr>
<tr>
<td>cvalue</td>
<td>The value for the VALUE attribute.</td>
</tr>
</tbody>
</table>

Examples
This procedure generates

```html
<PARAM NAME="cname" VALUE="cvalue">
```
216.5.92 PLAINTEXT Procedure

This procedure generates the `<PLAINTEXT>` and `</PLAINTEXT>` tags which direct the browser to render the text they surround in fixed-width type.

Syntax

```
HTP.PLAINTEXT(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in fixed-width font.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

`<PLAINTEXT cattributes>ctext</PLAINTEXT>`

216.5.93 PRECLOSE Procedure

This procedure generates the `</PRE>` tag which marks the end of a section of preformatted text in the body of the HTML page.

To mark the beginning of a section of preformatted text in the body of the HTML page, use the PREOPEN Procedure.

Syntax

```
HTP.PRECLOSE;
```

Examples

This procedure generates

`</PRE>`

216.5.94 PREOPEN Procedure

This procedure generates the `<PRE>` tag which marks the beginning of a section of preformatted text in the body of the HTML page.

To mark the end of a section of preformatted text in the body of the HTML page, use the PRECLOSE Procedure.

Syntax

```
HTP.PREOPEN(
  cclear         IN       VARCHAR2   DEFAULT NULL,
```
Parameters

Table 216-68  PREOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cwidth</td>
<td>The value for the WIDTH attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
```

216.5.95 PRINT Procedures

These procedures generate the specified parameter as a string terminated with the \n newline character.

The PRN Procedures performs the same operation but does not terminate with a newline character.

Syntax

```sql
HTP.PRINT (cbuf IN VARCHAR2);
HTP.PRINT (dbuf IN DATE);
HTP.PRINT (nbuf IN NUMBER);
```

Parameters

Table 216-69  PRINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbuf</td>
<td>The string to generate terminated by a newline.</td>
</tr>
<tr>
<td>dbuf</td>
<td>The string to generate terminated by a newline.</td>
</tr>
<tr>
<td>nbuf</td>
<td>The string to generate terminated by a newline.</td>
</tr>
</tbody>
</table>

Usage Notes

- The \n character is not the same as <BR>. The \n character formats the HTML source but it does not affect how the browser renders the HTML source. Use <BR> to control how the browser renders the HTML source.
- These procedures do not have function equivalents.
216.5.96 PRINTS Procedure

This procedure generates a string and replaces certain characters with a corresponding escape sequence.

The following characters are replaced with the corresponding escape sequence.

- `<` to `&lt;`
- `>` to `&gt;`
- `"` to `&quot;`
- `&` to `&amp;`

If not replaced, the special characters are interpreted as HTML control characters and produce garbled output. This procedure an the PS Procedure perform the same operation as the PRN Procedures but with character substitution.

**Syntax**

```
HTP.PRINTS (ctext IN VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The string where to perform character substitution.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This procedure does not have an HTF function equivalent (see Operational Notes for the HTF implementation).

216.5.97 PRN Procedures

These procedures generate the specified parameter as a string.

Unlike the PRINT Procedures the string is not terminated with the `\n` newline character.

**Syntax**

```
HTP.PRN (cbuf IN VARCHAR2);
HTP.PRN (dbuf IN DATE);
HTP.PRN (nbuf IN NUMBER);
```
Parameters

Table 216-71  PRN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbuf</td>
<td>The string to generate (not terminated by a newline).</td>
</tr>
<tr>
<td>dbuf</td>
<td>The string to generate (not terminated by a newline).</td>
</tr>
<tr>
<td>nbuf</td>
<td>The string to generate (not terminated by a newline).</td>
</tr>
</tbody>
</table>

Usage Notes

These procedures do not have function equivalents.

216.5.98 PS Procedure

This procedure generates a string and replaces certain characters with the corresponding escape sequence.

This procedure replaces the following characters with the corresponding escape sequence.

- `<` to `&lt;`
- `>` to `&gt;`
- `"` to `&quot;`
- `&` to `&amp;`

If not replaced, the special characters are interpreted as HTML control characters and produce garbled output. This procedure and the PRINTS Procedure perform the same operation as the PRN Procedures but with character substitution.

Syntax

```sql
HTP.PS (ctext IN VARCHAR2);
```

Parameters

Table 216-72  PS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The string where to perform character substitution.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure does not have an HTF function equivalent (see Operational Notes for the HTF implementation).
216.5.99 S Procedure

This procedure generates the `<S>` and `</S>` tags which direct the browser to render the text they surround in strikethrough type.

This performs the same operation as STRIKE Procedure.

Syntax

```sql
HTP.S (  
  ctext          IN       VARCHAR2,  
  cattributes    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 216-73  S Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in strikethrough type.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html```
<S cattributes>ctext</S>
```

216.5.100 SAMPLE Procedure

This procedure generates the `<SAMP>` and `</SAMP>` tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically.

Syntax

```sql
HTP.SAMPLE (  
  ctext          IN       VARCHAR2,  
  cattributes    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 216-74  SAMPLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in monospace font.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html```
<SAMP cattributes>ctext</SAMP>
```

---

ORACLE
216.5.101 SCRIPT Procedure

This procedure generates the `<SCRIPT>` and `</SCRIPT>` tags which contain a script written in languages such as JavaScript and VBscript.

Syntax

```
HTP.SCRIPT ( 
    cscript        IN       VARCHAR2, 
    clanguage      IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-75  SCRIPT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cscript</td>
<td>The text of the script. This is the text that makes up the script itself, not the name of a file containing the script.</td>
</tr>
<tr>
<td>clanguage</td>
<td>The language in which the script is written. If this parameter is omitted, the user's browser determines the scripting language.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<SCRIPT LANGUAGE=clanguage>cscript</SCRIPT>
```

so that

```
HTP.script ('Erupting_Volcano', 'Javascript');
```

generates

```
<SCRIPT LANGUAGE=Javascript>"script text here"</SCRIPT>
```

This causes the browser to run the script enclosed in the tags.

216.5.102 SMALL Procedure

This procedure generates the `<SMALL>` and `</SMALL>` tags, which direct the browser to render the text they surround using a small font.

Syntax

```
HTP.SMALL ( 
    ctext          IN       VARCHAR2, 
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-76  SMALL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in small font.</td>
</tr>
</tbody>
</table>
Table 216-76  (Cont.) SMALL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

<SMALL cattributes>ctext</SMALL>

216.5.103 STRIKE Procedure

This procedure generates the `<STRIKE>` and `</STRIKE>` tags which direct the browser to render the text they surround in strikethrough type.

This performs the same operation as S Procedure.

Syntax

```
HTP.STRIKE (          
  ctext          IN       VARCHAR2,   
  cattributes    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 216-77  STRIKE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be rendered in strikethrough type.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

<STRIKE cattributes>ctext</STRIKE>

216.5.104 STRONG Procedure

This procedure generates the `<STRONG>` and `</STRONG>` tags which direct the browser to render the text they surround in bold, or however "strong" is defined.

Syntax

```
HTP.STRONG(          
  ctext          IN       VARCHAR2,   
  cattributes    IN       VARCHAR2 DEFAULT NULL);
```
Parameters

Table 216-78  STRONG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to be emphasized.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to</td>
</tr>
<tr>
<td></td>
<td>be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

<STRONG cattributes>ctext</STRONG>

216.5.105 STYLE Procedure

This procedure generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page.

You can get more information about style sheets at http://www.w3.org. This feature is not compatible with browsers that support only HTML versions 2.0 or earlier. Such browsers will ignore this tag.

Syntax

HTP.STYLE(
  cstyle          IN       VARCHAR2);

Parameters

Table 216-79  STYLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cstyle</td>
<td>The the style information to</td>
</tr>
<tr>
<td></td>
<td>include.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

<STYLE>cstyle</STYLE>

216.5.106 SUB Procedure

This procedure generates the <SUB> and </SUB> tags which direct the browser to render the text they surround as subscript.

Syntax

HTP.SUB(
  ctext          IN       VARCHAR2,
  calign         in       VARCHAR2 DEFAULT NULL,
  cattributes    IN       VARCHAR2 DEFAULT NULL);
Parameters

Table 216-80  SUB Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in subscript.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<SUB ALIGN="calign" cattributes>ctext</SUB>
```

216.5.107 SUP Procedure

This procedure generates the `<SUP>` and `</SUP>` tags which direct the browser to render the text they surround as superscript.

Syntax

```sql
HTP.SUP(
  ctext          IN       VARCHAR2,
  calign         in       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-81  SUP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in superscript.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

216.5.108 TABLECAPTION Procedure

This procedure generates the `<CAPTION>` and `</CAPTION>` tags which place a caption in an HTML table.

Syntax

```sql
HTP.TABLECAPTION(
  ccaption       IN       VARCHAR2,
```

```sql
  ...);  
```
Parameters

Table 216-82 TABLECAPTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text for the caption.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

216.5.109 TABLECLOSE Procedure

This procedure generates the ```/TABLE``` tag which marks the end of an HTML table.

To define the beginning of an HTML table, use the TABLEOPEN Procedure.

Syntax

```pascal
HTP.TABLECLOSE;
```

Examples

This procedure generates

```html
</TABLE>
```

216.5.110 TABLEDATA Procedure

This procedure generates the `<TD>` and `</TD>` tags which insert data into a cell of an HTML table.

Syntax

```pascal
HTP.TABLEDATA(
    cvalue         IN       VARCHAR2   DEFAULT NULL,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cdp            IN       VARCHAR2   DEFAULT NULL,
    cnoswap        IN       VARCHAR2   DEFAULT NULL,
    crowspan       IN       VARCHAR2   DEFAULT NULL,
    ccolsrupan     IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```
Parameters

Table 216-83  TABLEDATA Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The data for the cell in the table.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cdp</td>
<td>The value for the DP attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to</td>
</tr>
<tr>
<td>ccolspan</td>
<td>the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The value for the COLSPAN attribute.</td>
</tr>
<tr>
<td></td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```html
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP cattributes>
cvalue</TD>
```

216.5.111 TABLEHEADER Procedure

This procedure generates the `<TH>` and `</TH>` tags which insert a header cell in an HTML table.

The `<TH>` tag is similar to the `<TD>` tag except that the text in this case the rows are usually rendered in bold type.

Syntax

```sql
HTP.TABLEHEADER(
  cvalue         IN       VARCHAR2   DEFAULT NULL,
  calign         IN       VARCHAR2   DEFAULT NULL,
  cdp            IN       VARCHAR2   DEFAULT NULL,
  cnowrap        IN       VARCHAR2   DEFAULT NULL,
  crowspan       IN       VARCHAR2   DEFAULT NULL,
  ccolspan       IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 216-84  TABLEHEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvalue</td>
<td>The data for the cell in the table.</td>
</tr>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cdp</td>
<td>The value for the DP attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to</td>
</tr>
<tr>
<td>ccolspan</td>
<td>the tag.</td>
</tr>
<tr>
<td>crispen</td>
<td>The value for the ROWSPAN attribute.</td>
</tr>
</tbody>
</table>
Table 216-84  (Cont.) TABLEHEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colspan</td>
<td>The value for the COLSPAN attribute.</td>
</tr>
<tr>
<td>attributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This procedure generates

```html
<TABLE ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
attributes>cvalue</TH>
```

216.5.112 TABLEOPEN Procedure

This procedure generates the `<TABLE>` tag which marks the beginning of an HTML table.

To define the end of an HTML table, use the TABLECLOSE Procedure.

**Syntax**

```sql
HTP.TABLEOPEN(
    cborder        IN       VARCHAR2   DEFAULT NULL,
    calign         IN       VARCHAR2   DEFAULT NULL,
    cnowrap        IN       VARCHAR2   DEFAULT NULL,
    cclear         IN       VARCHAR2   DEFAULT NULL,
    cattributes    IN       VARCHAR2   DEFAULT NULL);
```

**Parameters**

Table 216-85  TABLEOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border</td>
<td>The value for the BORDER attribute.</td>
</tr>
<tr>
<td>align</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>nowrap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>clear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>attributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

**Examples**

This procedure generates

```html
<TABLE "cborder" NOWRAP ALIGN="calign" CLEAR="cclear" cattributes>
```

216.5.113 TABLEROWCLOSE Procedure

This procedure generates the `</TR>` tag which marks the end of a new row in an HTML table.

To mark the beginning of a new row, use the TABLEROWOPEN Procedure.
Syntax

HTP.TABLEROWCLOSE;

Examples

This procedure generates

</TABLE>

216.5.114 TABLEROWOPEN Procedure

This procedure generates the <TR> tag which marks the beginning of a new row in an HTML table.

To mark the end of a new row, use the TABLEROWCLOSE Procedure.

Syntax

HTP.TABLEROWOPEN(
    calign IN VARCHAR2 DEFAULT NULL,
    cvalign IN VARCHAR2 DEFAULT NULL,
    cdp IN VARCHAR2 DEFAULT NULL,
    cnowrap IN VARCHAR2 DEFAULT NULL,
    cattributes IN VARCHAR2 DEFAULT NULL);

Parameters

Table 216-86   TABLEROWOPEN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calign</td>
<td>The value for the ALIGN attribute.</td>
</tr>
<tr>
<td>cvalign</td>
<td>The value for the VALIGN attribute.</td>
</tr>
<tr>
<td>cdp</td>
<td>The value for the DP attribute.</td>
</tr>
<tr>
<td>cnowrap</td>
<td>If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

<<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>>

216.5.115 TELETYPE Procedure

This procedure generates the <TT> and </TT> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font.

Syntax

HTP.TELETYPE(
    ctext IN VARCHAR2,
    cattributes IN VARCHAR2 DEFAULT NULL);
Parameters

Table 216-87  TELETYPE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in a fixed width typewriter font.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<TT cattributes>ctext</TT>
```

216.5.116 TITLE Procedure

This procedure generates the `<TITLE>` and `</TITLE>` tags which specify the text to display in the titlebar of the browser window.

Syntax

```
HTP.TITLE(
    ctitle          IN       VARCHAR2);
```

Parameters

Table 216-88  TITLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctitle</td>
<td>The text to display in the titlebar of the browser window.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<TITLE>ctitle</TITLE>
```

216.5.117 ULISTCLOSE Procedure

This procedure generates the `</UL>` tag which marks the end of an unordered list. An unordered list presents items with bullets.

To mark the beginning of an unordered list, use the ULISTOPEN Procedure. Add list items with LISTITEM Procedure.

Syntax

```
HTP.ULISTCLOSE;
```

Examples

This procedure generates

```
</TABLE>
```
216.5.118 ULISTOPEN Procedure

This procedure generates the `<UL>` tag which marks the beginning of an unordered list. An unordered list presents items with bullets.

To mark the end of an unordered list, use the ULISTCLOSE Procedure. Add list items with LISTITEM Procedure.

Syntax

```
HTP.ULISTOPEN(
  cclear         IN       VARCHAR2   DEFAULT NULL,
  cwrap          IN       VARCHAR2   DEFAULT NULL,
  cdingbat       IN       VARCHAR2   DEFAULT NULL,
  csrc           IN       VARCHAR2   DEFAULT NULL,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

**Table 216-89  ULISTOPEN Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cclear</td>
<td>The value for the CLEAR attribute.</td>
</tr>
<tr>
<td>cwrap</td>
<td>The value for the WRAP attribute.</td>
</tr>
<tr>
<td>cdingbat</td>
<td>The value for the DINGBAT attribute.</td>
</tr>
<tr>
<td>csrc</td>
<td>The value for the SRC attribute.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

216.5.119 UNDERLINE Procedure

This procedure generates the `<U>` and `</U>` tags, which direct the browser to render the text they surround with an underline.

Syntax

```
HTP.UNDERLINE(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL);
```

Parameters

**Table 216-90  UNDERLINE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render with an underline.</td>
</tr>
</tbody>
</table>
Table 216-90  (Cont.) UNDERLINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

\(<U \text{cattributes}\text{ctext}</U>\)

216.5.120 VARIABLE Procedure

This procedure generates the \(<\text{VAR}>\) and \(<</\text{VAR}>\) tags which direct the browser to render the text they surround in italics or however “variable” is defined stylistically.

Syntax

HTP.VARIABLE(
  ctext          IN       VARCHAR2,
  cattributes    IN       VARCHAR2   DEFAULT NULL);

Parameters

Table 216-91  VARIABLE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctext</td>
<td>The text to render in italics.</td>
</tr>
<tr>
<td>cattributes</td>
<td>The other attributes to be included as-is in the tag.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

\(<\text{VAR} \text{cattributes}\text{ctext}</\text{VAR}>\)

216.5.121 WBR Procedure

This procedure generates the \(<\text{WBR}>\) tag, which inserts a soft line break within a section of NOBR text.

Syntax

HTP.WBR;

Examples

This procedure generates

\(<\text{WBR}>\)
ORD_AUDIO

This Oracle Multimedia package supports the management and manipulation of audio data stored in BLOBs and BFILEs.

Note:

ORD_AUDIO is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The Oracle Multimedia ORD_AUDIO package provides functions and procedures to perform common operations such as exporting audio data from Oracle Database into operating system files, and extracting information from audio data. Information from audio data such as format, encoding type, and sampling rate can be extracted from audio data into an XML string and optionally into individual attributes.

For a complete description of this package within the context of Oracle Multimedia, see ORD_AUDIO in the Oracle Multimedia Reference.
This Oracle Multimedia package supports the management and manipulation of Digital Imaging and Communications in Medicine (DICOM) content stored in BLOBs or BFILEs.

Note:

ORD_DICOM was deprecated in Oracle Database 12c Release 2 (12.2). Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The DICOM standard is the dominant standard for radiology imaging and communication, to which all major manufacturers of radiological devices must conform. Oracle Multimedia DICOM provides native support for DICOM format medical images and other content, such as single-frame and multiframe images, waveforms, slices of 3-D volumes, video segments, and structured reports.

The Oracle Multimedia ORD_DICOM package provides functions and procedures to extract standard and private DICOM metadata from DICOM content into customizable XML documents, to perform image processing operations such as format conversion and thumbnail image generation, and to create new DICOM content. This Oracle Multimedia package also provides functions and procedures to check DICOM content for conformance based on a set of user-specified conformance rules, and to make DICOM content anonymous based on user-defined rules that specify the set of attributes to be made anonymous and the actions to be taken to make those attributes anonymous.

For a complete description of this package within the context of Oracle Multimedia, see ORD_DICOM in the Oracle Multimedia DICOM Developer's Guide.
This Oracle Multimedia package is used by Oracle Multimedia Digital Imaging and Communications in Medicine (DICOM) administrators to maintain the Oracle Multimedia DICOM data model repository.

Note:

ORD_DICOM_ADMIN was deprecated in Oracle Database 12c Release 2 (12.2). Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The DICOM data model repository is a collection of documents. An initial set of documents is loaded during installation. After installation, DICOM administrators can use the procedures and functions provided in the data model repository API of the Oracle Multimedia ORD_DICOM_ADMIN package to obtain document content as well as to insert, edit, and delete documents in the data model repository.

For a complete description of this package within the context of Oracle Multimedia, see ORD_DICOM_ADMIN in the Oracle Multimedia DICOM Developer’s Guide.
This Oracle Multimedia package supports the management and manipulation of media data stored in BLOBs and BFILEs. This package can be used in applications that require different types of multimedia data such as image, audio, and video to be stored in the same BLOB or BFILE column.

**Note:**

ORD_DOC is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The Oracle Multimedia ORD_DOC package provides functions and procedures to perform common operations such as exporting multimedia data from Oracle Database into operating system files, and extracting information from multimedia data. Information from image, audio, and video data such as format, mime type, and content length can be extracted into an XML string and optionally into individual attributes.

For a complete description of this package within the context of Oracle Multimedia, see ORD_DOC in the Oracle Multimedia Reference.
This Oracle Multimedia package supports the management and manipulation of image data stored in BLOBs and BFILES.

**Note:**

ORD_IMAGE is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The Oracle Multimedia ORD_IMAGE package provides functions and procedures to perform common operations such as exporting image data from Oracle Database into operating system files, creating thumbnail images, converting images from one format to another, and extracting information from image data. Information from image data such as format, width, height, and compression type can be extracted from image data into an XML string and optionally into individual attributes.

For a complete description of this package within the context of Oracle Multimedia, see ORD_IMAGE in the Oracle Multimedia Reference.
**ORD_VIDEO**

This Oracle Multimedia package supports the management and manipulation of video data stored in BLOBs and BFILES.

**Note:**

**ORD_VIDEO** is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The Oracle Multimedia **ORD_VIDEO** package provides functions and procedures to perform common operations such as exporting video data from Oracle Database into operating system files, and extracting information from video data. Information from video data such as format, width, height, and frame rate can be extracted from video data into an XML string and optionally into individual attributes.

For a complete description of this package within the context of Oracle Multimedia, see **ORD_VIDEO** in the *Oracle Multimedia Reference*. 
The `OWA_CACHE` package provides an interface that enables the PL/SQL Gateway cache to improve the performance of PL/SQL Web applications.

### See Also:
For more information about implementation of this package:
- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

The chapter contains the following topics:
- Constants
- Summary of OWA_CACHE Subprograms

#### 223.1 OWA_CACHE Constants

`OWA_CACHE` defines several constants to use when specifying parameter values.

- `system_level` CONSTANT VARCHAR(6) := 'SYSTEM';
- `user_level` CONSTANT VARCHAR(4) := 'USER';

#### 223.2 Summary of OWA_CACHE Subprograms

This table lists the OWA_CACHE subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE Procedure</td>
<td>Disables the cache for this particular request</td>
</tr>
<tr>
<td>GET_ETAG Function</td>
<td>Returns the tag associated with the cached content (used in the Validation technique model only)</td>
</tr>
<tr>
<td>GET_LEVEL Function</td>
<td>Returns the caching level (used in the Validation technique model only)</td>
</tr>
<tr>
<td>SET_CACHE Procedure</td>
<td>Sets up the cache headers for validation model cache type</td>
</tr>
<tr>
<td>SET_EXPIRES Procedure</td>
<td>Sets up the cache headers for expires model cache type</td>
</tr>
<tr>
<td>SET_NOT_MODIFIED Procedure</td>
<td>Sets up the headers for a not modified cache hit (used in the Validation technique model only)</td>
</tr>
<tr>
<td>SET_SURROGATE_CONTROL Procedure</td>
<td>Sets up the headers for a surrogate-control header for Web cache</td>
</tr>
</tbody>
</table>
223.2.1 DISABLE Procedure

This procedure disables the cache for this particular request.

**Syntax**

```sql
OWA_CACHE.DISABLE;
```

223.2.2 GET_ETAG Function

This function returns the tag associated with the cached content. It is used in the Validation technique only.

**Syntax**

```sql
OWA_CACHE.GET_ETAG
    RETURN VARCHAR2;
```

**Return Values**

The tag for cache hit, otherwise `NULL`.

223.2.3 GET_LEVEL Function

This returns the caching level. It is used in the Validation technique model only.

**Syntax**

```sql
OWA_CACHE.GET_LEVEL
    RETURN VARCHAR2;
```

**Return Values**

The caching level string ("USER" or "SYSTEM") for cache hit, otherwise `NULL`.

223.2.4 SET_CACHE Procedure

This procedure sets up the cache headers for validation model cache type.

**Syntax**

```sql
OWA_CACHE.SET_CACHE(
    p_etag        IN       VARCHAR2,
    p_level       IN       VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_etag</td>
<td>The <code>etag</code> associated with this content</td>
</tr>
<tr>
<td>p_level</td>
<td>The caching level (&quot;USER&quot; or &quot;SYSTEM&quot;).</td>
</tr>
</tbody>
</table>
Exceptions

.VALUE_ERROR is thrown if

• p_etag is greater than 55
• p_level is not 'USER' or 'SYSTEM'

223.2.5 SET_EXPIRES Procedure

This procedure sets up the cache headers for expires model cache type.

Syntax

OWA_CACHE.SET_EXPIRES(
    p_expires   IN       NUMBER,
    p_level     IN       VARCHAR2);

Parameters

Table 223-3 SET_EXPIRES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_expires</td>
<td>The number of minutes this content is valid.</td>
</tr>
<tr>
<td>p_level</td>
<td>The caching level ('USER' or 'SYSTEM').</td>
</tr>
</tbody>
</table>

Exceptions

.VALUE_ERROR is thrown if

• p_expires is negative or zero
• p_level is not 'USER' or 'SYSTEM'
• p_expires is > 525600 (1 year)

223.2.6 SET_NOT_MODIFIED Procedure

This procedure sets up the headers for a not-modified cache hit. It is used in the Validation technique only.

Syntax

OWA_CACHE.SET_NOT_MODIFIED;

Exceptions

.VALUE_ERROR is thrown if If the etag was not passed in
223.2.7 SET_SURROGATE_CONTROL Procedure

This procedure sets the headers for a surrogate-control header for Web cache.

**Syntax**

```sql
OWA_CACHE.SET_SURROGATE_CONTROL(
    p_value IN VARCHAR2);
```

**Parameters**

**Table 223-4  SET_SURROGATE_CONTROL Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value</td>
<td>The value to be passed as the Surrogate-Control header.</td>
</tr>
</tbody>
</table>

**Exceptions**

`VALUE_ERROR` is thrown if `p_value` is greater than 55 in length.
The OWA_COOKIE package provides an interface for sending and retrieving HTTP cookies from the client's browser.

See Also:
For more information about implementation of this package:
- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

The chapter contains the following topics:
- Overview
- Types
- Rules and Limits
- Summary of OWA_COOKIE Subprograms

224.1 OWA_CUSTOM Overview
Cookies are opaque strings sent to the browser to maintain state between HTTP calls. State can be maintained throughout the client's sessions, or longer if an expiration date is included. The system date is calculated with reference to the information specified in the OWA_CUSTOM package.

224.2 OWA_COOKIE Types
This datatype contains cookie name-value pairs.

Since the HTTP standard allows cookie names to be overloaded (that is, multiple values can be associated with the same cookie name), there is a PL/SQL RECORD holding all values associated with a given cookie name.

```sql
TYPE vc_arr IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER.

TYPE COOKIE IS RECORD (
    name           VARCHAR2(4000),
    vals           vc_arr,
    num_vals       INTEGER);
```

224.3 OWA_COOKIE Rules and Limits
All HTTP headers must be in English and the ASCII character set. If the headers are generated from the database, verify they are created in the English language.
224.4 Summary of OWA_COOKIE Subprograms

This table lists the OWA_COOKIE subprograms and briefly describes them.

### Table 224-1  OWA_COOKIE Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET Function</td>
<td>Gets the value of the specified cookie</td>
</tr>
<tr>
<td>GET_ALL Procedure</td>
<td>Gets all cookie name-value pairs</td>
</tr>
<tr>
<td>REMOVE Procedure</td>
<td>Removes the specified cookie</td>
</tr>
<tr>
<td>SEND procedure</td>
<td>Generates a &quot;Set-Cookie&quot; line in the HTTP header</td>
</tr>
</tbody>
</table>

#### 224.4.1 GET Function

This function returns the values associated with the specified cookie. The values are returned in a OWA_COOKIE.COOKIE DATA TYPE.

**Syntax**

```sql
OWA_COOKIE.GET(
    name           IN       VARCHAR2)
RETURN COOKIE;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the cookie.</td>
</tr>
</tbody>
</table>

**Return Values**

OWA_COOKIE.COOKIE DATA TYPE.

#### 224.4.2 GET_ALL Procedure

This procedure returns all cookie names and their values from the client's browser. The values appear in the order in which they were sent from the browser.

**Syntax**

```sql
OWA_COOKIE.GET_ALL(
    names   OUT       vc_arr,
    vals    OUT       vc_arr,
    num_vals OUT       INTEGER);
```
Parameters

### Table 224-3 GET_ALL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>names</td>
<td>The names of the cookies.</td>
</tr>
<tr>
<td>vals</td>
<td>The values of the cookies.</td>
</tr>
<tr>
<td>num_vals</td>
<td>The number of cookie-value pairs.</td>
</tr>
</tbody>
</table>

#### 224.4.3 REMOVE Procedure

This procedure forces a cookie to expire immediately by setting the "expires" field of a Set-Cookie line in the HTTP header to "01-Jan-1990".

This procedure must be called within the context of an HTTP header.

**Syntax**

```sql
OWA_COOKIE.REMOVE(
    name IN VARCHAR2,
    val IN VARCHAR2,
    path IN VARCHAR2 DEFAULT NULL);
```

### Parameters

#### Table 224-4 REMOVE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the cookie to expire.</td>
</tr>
<tr>
<td>val</td>
<td>The value of the cookie.</td>
</tr>
<tr>
<td>path</td>
<td>[Currently unused]</td>
</tr>
</tbody>
</table>

#### 224.4.4 SEND procedure

This procedure generates a Set-Cookie line, which transmits a cookie to the client.

This procedure must occur in the context of an HTTP header.

**Syntax**

```sql
OWA_COOKIE.SEND(
    name in varchar2,  
    value in varchar2,  
    expires in date DEFAULT NULL,  
    path in varchar2 DEFAULT NULL,  
    domain in varchar2 DEFAULT NULL,  
    secure in varchar2 DEFAULT NULL);
```
### Parameters

**Table 224-5  SEND Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the cookie.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the cookie.</td>
</tr>
<tr>
<td>expires</td>
<td>The date at which the cookie will expire</td>
</tr>
<tr>
<td>path</td>
<td>The value for the path field.</td>
</tr>
<tr>
<td>domain</td>
<td>The value for the domain field.</td>
</tr>
<tr>
<td>secure</td>
<td>If the value of this parameter is not NULL, the &quot;secure&quot; field is added to the line.</td>
</tr>
</tbody>
</table>
The `OWA_CUSTOM` package provides a Global PLSQL Agent Authorization callback function. It is used when PLSQL Agent's authorization scheme is set to `GLOBAL` or `CUSTOM` when there is no overriding `OWA_CUSTOM` package.

### See Also:

For more information about implementation of this package:
- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod_plsql*

The chapter contains the following topics:
- **Constants**
- **Summary of OWA_CUSTOM Subprograms**

#### 225.1 OWA_CUSTOM Constants

`OWA_CUSTOM` defines several constants to use when specifying parameter values.

- `dbms_server_timezone CONSTANT VARCHAR2(3) := 'PST';`
- `dbms_server_gmtdiff CONSTANT NUMBER := NULL;`

#### 225.2 Summary of OWA_CUSTOM Subprograms

The `AUTHORIZE` function is the only subprogram in the `OWA_CUSTOM` package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHORIZE Function</td>
<td>Provides a Global PLSQL Agent Authorization callback function</td>
</tr>
</tbody>
</table>

#### 225.2.1 AUTHORIZE Function

This function is used when PLSQL Agent's authorization scheme is set to `GLOBAL` or `CUSTOM` when there is no overriding `OWA_CUSTOM` package.

**Syntax**

```plsql
OWA_CUSTOM.AUTHORIZE
    RETURN BOOLEAN;
```
The OWA_IMAGE package provides an interface to access the coordinates where a user clicked on an image.

The chapter contains the following topics:

- Overview
- Types
- Variables
- Examples
- Summary of OWA_IMAGE Subprograms

See Also:

For more information about implementation of this package:

- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

226.1 OWA_IMAGE Overview

Use this package when you have any image map whose destination links invoke the PL/SQL Gateway.

226.2 OWA_IMAGE Types

This datatype (point) contain the X and Y values of a coordinate, and so provides the coordinates of a user's click on an imagemap.

It is defined as:

```
TYPE POINT IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER
```

226.3 OWA_IMAGE Variables

This package variable (null_point) of TYPE POINT is used to default point parameters. Both the X and the Y fields of this variable are NULL.
### 226.4 OWA_IMAGE Examples

This example shows use of OWA_IMAGE.

```sql
CREATE OR REPLACE PROCEDURE process_image
  (my_img IN OWA_IMAGE.POINT)
AS
  x integer := OWA_IMAGE.GET_X(my_img);
  y integer := OWA_IMAGE.GET_Y(my_img);
BEGIN
  /* process the coordinate */
END;
```

### 226.5 Summary of OWA_IMAGE Subprograms

This table lists the OWA_IMAGE subprograms and briefly describes them.

**Table 226-1   OWA_IMAGE Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_X Function</td>
<td>Gets the X value of a point type</td>
</tr>
<tr>
<td>GET_Y Function</td>
<td>Gets the Y value of a point type</td>
</tr>
</tbody>
</table>

#### 226.5.1 GET_X Function

This function returns the x coordinate of the point where the user clicked on an image map.

**Syntax**

```sql
OWA_IMAGE.GET_X(
  p    IN    point
) RETURN INTEGER;
```

**Parameters**

**Table 226-2   GET_X Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>The point where the user clicked.</td>
</tr>
</tbody>
</table>

**Return Values**

The X coordinate as an integer.
226.5.2 GET_Y Function

This function returns the Y coordinate of the point where the user clicked on an image map.

Syntax

OWA_IMAGE.GET_Y(p IN point) RETURN INTEGER;

Parameters

Table 226-3  GET_Y Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>The point where the user clicked.</td>
</tr>
</tbody>
</table>

Return Values

The Y coordinate as an integer.
The `OWA_OPT_LOCK` package contains subprograms that impose optimistic locking strategies so as to prevent lost updates.

This chapter contains the following topics:

- Overview
- Types
- Summary of `OWA_OPT_LOCK` Subprograms

### See Also:

For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod_plsql*

## 227.1 OWA_OPT_LOCK Overview

The `OWA_OPT_LOCK` package contains subprograms that impose optimistic locking strategies, so as to prevent lost updates.

It checks if the row that the user is interested in updating has been changed by someone else in the meantime.

The PL/SQL Gateway cannot use conventional database locking schemes because HTTP is a stateless protocol. The `OWA_OPT_LOCK` package gives you two ways of dealing with the lost update problem:

- The hidden fields method stores the previous values in hidden fields in the HTML page. When the user requests an update, the PL/SQL Gateway checks these values against the current state of the database. The update operation is performed only if the values match. To use this method, call the `owa_opt_lock.store_values` procedure.

- The checksum method stores a checksum rather than the values themselves. To use this method, call the `owa_opt_lock.checksum` function.

These methods are optimistic. They do not prevent other users from performing updates, but they do reject the current update if an intervening update has occurred.

## 227.2 OWA_TEXT Types

This datatype is a PL/SQL table intended to hold ROWIDs.

```
TYPE VCARRAY IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER
```
Note that this is different from the OWA_TEXT.VC_ARR DATA TYPE.

227.3 Summary of OWA_OPT_LOCK Subprograms

This table lists the OWA_OPT_LOCK subprograms and briefly describes them.

Table 227-1   OWA_OPT_LOCK Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKSUM Functions</td>
<td>Returns the checksum value</td>
</tr>
<tr>
<td>GET_ROWID Function</td>
<td>Returns the ROWID value</td>
</tr>
<tr>
<td>STORE_VALUES Procedure</td>
<td>Stores unmodified values in hidden fields for later verification</td>
</tr>
<tr>
<td>VERIFY_VALUES Function</td>
<td>Verifies the stored values against modified values</td>
</tr>
</tbody>
</table>

227.3.1 CHECKSUM Functions

This function returns a checksum value for a specified string, or for a row in a table. For a row in a table, the function calculates the checksum value based on the values of the columns in the row. This function comes in two versions.

The first version returns a checksum based on the specified string. This is a "pure" 32-bit checksum executed by the database and based on the Internet 1 protocol.

The second version returns a checksum based on the values of a row in a table. This is a "impure" 32-bit checksum based on the Internet 1 protocol.

Syntax

```
OWA_OPT_LOCK.CHECKSUM(
    p_buff        IN       VARCHAR2)
RETURN NUMBER;
```

```
OWA_OPT_LOCK.CHECKSUM(
    p_owner       IN      VARCHAR2,
    p_tname       IN       VARCHAR2,
    p_rowid       IN       ROWID)
RETURN NUMBER;
```

Parameters

Table 227-2   CHECKSUM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_buff</td>
<td>The nstring where you want to calculate the checksum.</td>
</tr>
<tr>
<td>p_owner</td>
<td>The owner of the table.</td>
</tr>
<tr>
<td>p_tname</td>
<td>The table name.</td>
</tr>
<tr>
<td>p_rowid</td>
<td>The row in p_tname where you want to calculate the checksum value. Use the GET_ROWID Function to convert VCARRAY values to proper rowids.</td>
</tr>
</tbody>
</table>
227.3.2 GET_ROWID Function

This function returns the ROWID datatype from the specified OWA_OPT_LOCK.VCARRAY DATA TYPE.

Syntax

```
OWA_OPT_LOCK.GET_ROWID(
    p_old_values      IN      vcarray)
RETURN ROWID;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_old_values</td>
<td>This parameter is usually passed in from an HTML form.</td>
</tr>
</tbody>
</table>

227.3.3 STORE_VALUES Procedure

This procedure stores the column values of the row that you want to update later. The values are stored in hidden HTML form elements.

Syntax

```
OWA_OPT_LOCK.STORE_VALUES(
    p_owner        IN       VARCHAR2,
    p_tname        IN       VARCHAR2,
    p_rowid        IN       ROWID);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_owner</td>
<td>The owner of the table.</td>
</tr>
<tr>
<td>p_tname</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>p_rowid</td>
<td>The row where you want to store values.</td>
</tr>
</tbody>
</table>

Usage Notes

Before updating the row, compare these values with the current row values to ensure that the values in the row have not been changed. If the values have changed, you can warn the users and let them decide if the update should take place.

The procedure generates series of hidden form elements:

- One hidden form element is created for the table owner. The name of the element is "old_p_tname", where p_tname is the name of the table. The value of the element is the owner name.
• One hidden form element is created for the table name. The name of the element is "old_p_tname", where p_tname is the name of the table. The value of the element is the table name.

• One element is created for each column in the row. The name of the element is "old_p_tname", where p_tname is the name of the table. The value of the element is the column value.

See also the VERIFY_VALUES Function.

227.3.4 VERIFY_VALUES Function

This function verifies whether values in the specified row have been updated since the last query.

Use this function with the STORE_VALUES Procedure.

Syntax

OWA_OPT_LOCK.VERIFY_VALUES(
  p_old_values IN vcarray)
RETURN BOOLEAN;

Parameters

<table>
<thead>
<tr>
<th>Table 227-5  VERIFY_VALUES Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>p_old_values</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Return Values

TRUE if no other update has been performed, otherwise FALSE.
The OWA_PATTERN package provides an interface to locate text patterns within strings and replace the matched string with another string.

See Also:
For more information about implementation of this package, see the following:
- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

The chapter contains the following topics:
- Types
- Operational Notes
- Summary of OWA_PATTERN Subprograms

### 228.1 OWA_PATTERN Types

You can use a pattern as both an input and output parameter. Because of this, you can pass the same regular expression to OWA_PATTERN function calls, and it only has to be parsed once.

- OWA_PATTERN.PATTERN

### 228.2 OWA_PATTERN Operational Notes

The OWA_PATTERN subprograms are overloaded. Specifically, there are six versions of MATCH, and four each of AMATCH and CHANGE.

The subprograms use the following parameters:

- **line** - This is the target to be examined for a match. It can be more than one line of text or a owa_text.multi_line datatype.
- **pat** - This is the pattern that the subprograms attempt to locate in line. The pattern can contain regular expressions. In the owa_pattern.change function and procedure, this parameter is called from_str.
- **flags** - This specifies whether the search is case-sensitive or if substitutions are done globally.

Use regular expressions with the subprograms in this package. You Specify a regular expression by creating the string you want to match interspersed with various wildcard tokens and quantifiers.

- Wildcards
228.2.1 OWA_PATTERN Wildcards

Wildcard tokens match something other than themselves.

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Matches newline or the beginning of the target</td>
</tr>
<tr>
<td>$</td>
<td>Matches newline or the end of the target</td>
</tr>
<tr>
<td>\n</td>
<td>Matches newline</td>
</tr>
<tr>
<td>.</td>
<td>Matches any character except newline</td>
</tr>
<tr>
<td>\t</td>
<td>Matches tab</td>
</tr>
<tr>
<td>\d</td>
<td>Matches digits [0-9]</td>
</tr>
<tr>
<td>\D</td>
<td>Matches non-digits [not 0-9]</td>
</tr>
<tr>
<td>\w</td>
<td>Matches word characters (0-9, a-z, A-Z, or _)</td>
</tr>
<tr>
<td>\W</td>
<td>Matches non-word characters (not 0-9, a-z, A-Z, or _)</td>
</tr>
<tr>
<td>\s</td>
<td>Matches whitespace characters (blank, tab, or newline)</td>
</tr>
<tr>
<td>\S</td>
<td>Matches non-whitespace characters (not blank, tab, or newline)</td>
</tr>
<tr>
<td>\b</td>
<td>Matches &quot;word&quot; boundaries (between \w and \W)</td>
</tr>
<tr>
<td>\x&lt;HEX&gt;</td>
<td>Matches the value in the current character set of the two hexadecimal digits</td>
</tr>
<tr>
<td>&lt;OCT&gt;</td>
<td>Matches the value in the current character set of the two or three octal digits</td>
</tr>
<tr>
<td>\</td>
<td>Followed by any character not covered by another case matches that character</td>
</tr>
</tbody>
</table>

& Applies only to CHANGE. This causes the string that matched the regular expression to be included in the string that replaces it. This differs from the other tokens in that it specifies how a target is changed rather than how it is matched. This is explained further under CHANGE Functions and Procedures.

228.2.2 OWA_PATTERN Quantifiers

Any tokens except & can have their meaning extended by any of the following quantifiers. You can also apply these quantifiers to literals.

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>0 or 1 occurrence(s)</td>
</tr>
<tr>
<td>*</td>
<td>0 or more occurrences</td>
</tr>
<tr>
<td>+</td>
<td>1 or more occurrence(s)</td>
</tr>
<tr>
<td>{n}</td>
<td>Exactly n occurrences</td>
</tr>
<tr>
<td>{n,}</td>
<td>At least n occurrences</td>
</tr>
</tbody>
</table>
### Table 228-2 (Cont.) Quantifiers

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>([n,m])</td>
<td>At least (n), but not more than (m), occurrences</td>
</tr>
</tbody>
</table>

#### 228.2.3 OWA_PATTERN Flags

In addition to targets and regular expressions, the `OWA_PATTERN` functions and procedures use flags to affect how they are interpreted.

### Table 228-3 Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>This indicates a case-insensitive search.</td>
</tr>
<tr>
<td>g</td>
<td>This applies only to <code>CHANGE</code>. It indicates a global replace. That is, all portions of the target that match the regular expression are replaced.</td>
</tr>
</tbody>
</table>

#### 228.3 Summary of OWA_PATTERN Subprograms

This table lists the OWA_PATTERN subprograms and briefly describes them.

### Table 228-4 OWA_PATTERN Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMATCH Function</td>
<td>Determines if a string contains the specified pattern. It lets you specify where in the string the match has to occur</td>
</tr>
<tr>
<td>CHANGE Functions and Procedures</td>
<td>Replaces a pattern within a string. If you call it as a function it returns the number of times the regular expression was found and replaced</td>
</tr>
<tr>
<td>GETPAT Procedure</td>
<td>Generates a pattern datatype from a <code>VARCHAR2</code> type</td>
</tr>
<tr>
<td>MATCH Function</td>
<td>Determines if a string contains the specified pattern</td>
</tr>
</tbody>
</table>

#### 228.3.1 AMATCH Function

This function specifies if a pattern occurs in a particular location in a string.

There are four versions to this function:

- The first and second versions of the function do not save the matched tokens (these are saved in the `backrefs` parameters in the third and fourth versions). The difference between the first and second versions is the `pat` parameter, which can be a `VARCHAR2` or a pattern datatype.
- The third and fourth versions of the function save the matched tokens in the `backrefs` parameter. The difference between the third and fourth versions is the `pat` parameter, which can be a `VARCHAR2` or a pattern datatype.
Note:

If multiple overlapping strings match the regular expression, this function takes the longest match.

Syntax

OWA_PATTERN.AMATCH(
  line           IN       VARCHAR2,
  from_loc       IN       INTEGER,
  pat            IN       VARCHAR2,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN INTEGER;

OWA_PATTERN.AMATCH(
  line           IN       VARCHAR2,
  from_loc       IN       INTEGER,
  pat            IN OUT   PATTERN,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN INTEGER;

OWA_PATTERN.AMATCH(
  line           IN       VARCHAR2
  from_loc       IN       INTEGER
  pat            in       varchar2
  backrefs       OUT      owa_text.vc_arr
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN INTEGER;

OWA_PATTERN.AMATCH(
  line           IN       VARCHAR2
  from_loc       IN       INTEGER
  pat            IN OUT   PATTERN
  backrefs       OUT      owa_text.vc_arr
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN INTEGER;

Parameters

Table 228-5  AMATCH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>The text to search in.</td>
</tr>
<tr>
<td>from_loc</td>
<td>The location (in number of characters) in line where the search is to begin.</td>
</tr>
<tr>
<td>pat</td>
<td>The string to match. It can contain regular expressions. This can be either a VARCHAR2 or a pattern. If it is a pattern, the output value of this parameter is the pattern matched.</td>
</tr>
<tr>
<td>backrefs</td>
<td>The text that is matched. Each token that is matched is placed in a cell in the OWA_TEXT.VC_ARR DATA TYPE PL/SQL table.</td>
</tr>
<tr>
<td>flags</td>
<td>Whether or not the search is case-sensitive. If the value of this parameter is &quot;i&quot;, the search is case-insensitive. Otherwise the search is case-sensitive.</td>
</tr>
</tbody>
</table>
Return Values
The index of the character after the end of the match, counting from the beginning of line. If there was no match, the function returns 0.

228.3.2 CHANGE Functions and Procedures

This function or procedure searches and replaces a string or multi_line datatype. If multiple overlapping strings match the regular expression, this subprogram takes the longest match.

Syntax

OWA_PATTERN.CHANGE(
    line           IN OUT    VARCHAR2,
    from_str       IN        VARCHAR2,
    to_str         IN        VARCHAR2,
    flags          IN        VARCHAR2   DEFAULT NULL
) RETURN INTEGER;

OWA_PATTERN.CHANGE(
    line           IN OUT   VARCHAR2,
    from_str       IN       VARCHAR2,
    to_str         IN       VARCHAR2,
    flags          IN       VARCHAR2   DEFAULT NULL);

owa_pattern.change(
    mline          IN OUT   owa_text.multi_line,
    from_str       IN       VARCHAR2,
    to_str         IN       VARCHAR2,
    flags          IN       VARCHAR2   DEFAULT NULL)
RETURN INTEGER;

owa_pattern.change(
    ml          IN OUT   owa_text.multi_line,
    from_str       IN       VARCHAR2,
    to_str         IN       VARCHAR2,
    flags          IN       VARCHAR2   DEFAULT NULL);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>The text to search in. The output value of this parameter is the altered string.</td>
</tr>
<tr>
<td>mline</td>
<td>The text to search in. This is a owa_text.multi_line datatype. The output value of this parameter is the altered string.</td>
</tr>
<tr>
<td>from_str</td>
<td>The regular expression to replace.</td>
</tr>
<tr>
<td>to_str</td>
<td>The substitution pattern.</td>
</tr>
<tr>
<td>flags</td>
<td>Whether or not the search is case-sensitive, and whether or not changes are to be made globally. If &quot;i&quot; is specified, the search is case-insensitive. If &quot;g&quot; is specified, changes are made to all matches. Otherwise, the function stops after the first substitution is made.</td>
</tr>
</tbody>
</table>
Return Values

As a function, it returns the number of substitutions made. If the flag "g" is not used, this number can only be 0 or 1 and only the first match is replaced. The flag "g" specifies to replace all matches with the regular expression.

Examples

**Example:**

```
OWA_PATTERN.CHANGE('Cats in pajamas', 'C.+in', '& red ')
```

The regular expression matches the substring "Cats in". It then replaces this string with "& red". The ampersand character "&" indicates "Cats in" because that is what matched the regular expression. Thus, this procedure replaces the string "Cats in pajamas" with "Cats in red". If you call this as a function instead of a procedure, the value returned is 1, indicating that a single substitution has been made.

**Example 2:**

```
CREATE OR REPLACE PROCEDURE test_pattern as theline VARCHAR2(256);
num_found INTEGER;
BEGIN
    theline := 'what is the goal?';
    num_found := OWA_PATTERN.CHANGE(theline, 'goal', 'idea', 'g');
    HTP.PRINT(num_found); -- num_found is 1
    HTP.PRINT(theline); -- theline is 'what is the idea?'
END;
/ SHOW ERRORS
```

### 228.3.3 GETPAT Procedure

This procedure converts a VARCHAR2 string into an OWA_PATTERN.PATTERN DATA TYPE.

**Syntax**

```
OWA_PATTERN.GETPAT(
    arg IN VARCHAR2,
    pat IN OUT pattern);
```

**Parameters**

**Table 228-7** GETPAT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg</td>
<td>The string to convert.</td>
</tr>
<tr>
<td>pat</td>
<td>the OWA_PATTERN.PATTERN DATA TYPE initialized with arg.</td>
</tr>
</tbody>
</table>
228.3.4 MATCH Function

This function determines if a string contains the specified pattern. The pattern can contain regular expressions. If multiple overlapping strings can match the regular expression, this function takes the longest match.

Syntax

```
owa_pattern.match(
  line           IN       VARCHAR2,
  pat            IN       VARCHAR2,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN BOOLEAN;
```

```
owa_pattern.match(
  line           IN       VARCHAR2,
  pat            IN OUT   PATTERN,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN BOOLEAN;
```

```
owa_pattern.match(
  line           IN       VARCHAR2,
  pat            IN       VARCHAR2,
  backrefs       OUT      owa_text.vc_arr,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN BOOLEAN;
```

```
OWA_PATTERN.MATCH(
  line           IN       VARCHAR2,
  pat            IN OUT   PATTERN,
  backrefs       OUT      owa_text.vc_arr,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN BOOLEAN;
```

```
owa_pattern.match(
  mline          IN       owa_text.multi_line,
  pat            IN       VARCHAR2,
  rlist          OUT      owa_text.row_list,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN BOOLEAN;
```

```
OWA_PATTERN.MATCH(
  mline          IN       owa_text.multi_line,
  pat            IN OUT   pattern,
  rlist          OUT      owa_text.row_list,
  flags          IN       VARCHAR2   DEFAULT NULL)
RETURN BOOLEAN;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>The line to search in.</td>
</tr>
<tr>
<td>mline</td>
<td>The text to search in. This is an owa_text.multi_line datatype.</td>
</tr>
</tbody>
</table>
Table 228-8  (Cont.) MATCH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pat</td>
<td>The pattern to match. This is either a VARCHAR2 or a OWA_PATTERN.PATTERN DATA TYPE. It is a pattern, the output value of this parameter is the pattern matched.</td>
</tr>
<tr>
<td>backrefs</td>
<td>The text that is matched. Each token that is matched is placed in a cell in the OWA_TEXT.VC_ARR DATA TYPE PL/SQL table. This parameter is a row_list that holds each string in the target that was matched by a sequence of tokens in the regular expression.</td>
</tr>
<tr>
<td>rlist</td>
<td>An output parameter containing a list of matches.</td>
</tr>
<tr>
<td>flags</td>
<td>Whether or not the search is case-sensitive. If the value of this parameter is &quot;i&quot;, the search is case-insensitive. Otherwise the search is case-sensitive.</td>
</tr>
</tbody>
</table>

Return Values

TRUE if a match was found, FALSE otherwise.

Examples

KAZOO is the target where it is searching for the zoo.* regular expression. The period indicates any character other than newline, and the asterisk matches 0 or more of the preceding characters. In this case, it matches any character other than the newline.

Therefore, this regular expression specifies that a matching target consists of zoo, followed by any set of characters neither ending in nor including a newline (which does not match the period). The i flag indicates to ignore case in the search. In this case, the function returns TRUE, which indicates that a match had been found.

boolean foundMatch;
foundMatch := owa_pattern.match('KAZOO', 'zoo.*', 'i');

The following example searches for the string “goal” followed by any number of characters in sometext. If found,

sometext  VARCHAR2(256);
pat       VARCHAR2(256);
sometext := 'what is the goal?'
pat := 'goal.*';
IF OWA_PATTERN.MATCH(sometext, pat)
  THEN
    HTP.PRINT('Match found');
  ELSE
    HTP.PRINT('Match not found');
END IF;

Operational Notes

- The regular expression in this function can be either a VARCHAR2 or an OWA_PATTERN.PATTERN DATA TYPE. Create an OWA_PATTERN.PATTERN DATA TYPE from a string using the OWA_PATTERN.GETPAT procedure.

Chapter 228
Summary of OWA_PATTERN Subprograms
• Create a MULTI_LINE DATA TYPE from a long string using the OWA_TEXT.STREAM2MULTI procedure. If a multi_line is used, the rlist parameter specifies a list of chunks where matches were found.

• If the line is a string and not a multi_line, you can add an optional output parameter called backrefs. This parameter is a row_list that holds each string in the target that was matched by a sequence of tokens in the regular expression.
The **OWA_SEC** package provides an interface for custom authentication.

**See Also:**

For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod_plsql*

The chapter contains the following topics:

- Operational Notes
- Summary of OWA_SEC Subprograms

### 229.1 OWA_SEC Operational Notes

Parameters that have default values are optional.

### 229.2 Summary of OWA_SEC Subprograms

This table lists the OWA_SEC subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_CLIENT_HOSTNAME Function</td>
<td>Returns the client's hostname</td>
</tr>
<tr>
<td>GET_CLIENT_IP Function</td>
<td>Returns the client's IP address</td>
</tr>
<tr>
<td>GET_PASSWORD Function</td>
<td>Returns the password that the user entered</td>
</tr>
<tr>
<td>GET_USER_ID Function</td>
<td>Returns the username that the user entered</td>
</tr>
<tr>
<td>SET_AUTHORIZATION Procedure</td>
<td>Enables the PL/SQL application to use custom authentication</td>
</tr>
<tr>
<td>SET_PROTECTION_REALM Procedure</td>
<td>Defines the realm that the page is in</td>
</tr>
</tbody>
</table>
229.2.1 GET_CLIENT_HOSTNAME Function

This function returns the hostname of the client.

Syntax

```sql
OWA_SEC.GET_CLIENT_HOSTNAME
RETURN VARCHAR2;
```

Return Values

The hostname.

229.2.2 GET_CLIENT_IP Function

This function returns the IP address of the client.

Syntax

```sql
OWA_SEC.GET_CLIENT_IP
RETURN OWA_UTIL.IP_ADDRESS;
```

Return Values

The IP address. The `owa_util.ip_address` datatype is a PL/SQL table where the first four elements contain the four numbers of the IP address. For example, if the IP address is 123.45.67.89 and the variable `ipaddr` is of the `owa_util.ip_address` datatype, the variable would contain the following values:

- `ipaddr(1) = 123`
- `ipaddr(2) = 45`
- `ipaddr(3) = 67`
- `ipaddr(4) = 89`

229.2.3 GET_PASSWORD Function

This function returns the password that the user used to log in.

Syntax

```sql
OWA_SEC.GET_PASSWORD
RETURN VARCHAR2;
```

Return Values

The password.

Usage Notes

For security reasons, this function returns a true value only when custom authentication is used. If you call this function when you are not using custom authentication, the function returns an undefined value. Thus, the database passwords are not exposed.
229.2.4 GET_USER_ID Function

This function returns the username that the user used to log in.

Syntax

OWA_SEC.GET_USER_ID
RETURN VARCHAR2;

Return Values

The username.

229.2.5 SET_AUTHORIZATION Procedure

This procedure, called in the initialization portion of the OWA_CUSTOM package, sets the authorization scheme for the PL/SQL Gateway.

This implements your authorize function, which authorizes the user before his requested procedure is run. The placement of the authorize function depends on the scheme you select.

Syntax

OWA_SEC.SET_AUTHORIZATION(
    scheme IN INTEGER);
Parameters

**Table 229-2** SET_AUTHORIZATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme</td>
<td>The authorization scheme. It is one of the following schemes for SET_AUTHORIZATION:</td>
</tr>
<tr>
<td></td>
<td>· <strong>OWA_SEC.NO_CHECK</strong> - Specifies that the PL/SQL application is not to do any custom authentication. This is the default.</td>
</tr>
<tr>
<td></td>
<td>· <strong>OWA_SEC.GLOBAL</strong> - Defines an authorize function that is called for all users and all procedures. This is the <strong>OWA_CUSTOM.AUTHORIZE Function</strong> in the &quot;sys&quot; schema.</td>
</tr>
<tr>
<td></td>
<td>· <strong>OWA_SEC.PER_PACKAGE</strong> - Define an authorize function that is called when procedures in a package or anonymous procedures are called. If the procedures are in a package, the package AUTHORIZE function in the user's schema is called to authorize the user. If the procedures are not in a package, then the anonymous authorize function in the user's schema is called.</td>
</tr>
<tr>
<td></td>
<td>· <strong>OWA_SEC.CUSTOM</strong> - Implements different authorize functions for each user. The function <strong>OWA_CUSTOM.AUTHORIZE Function</strong> in the user's schema is called to authorize the user. If the user's schema does not contain an <strong>OWA_CUSTOM.AUTHORIZE Function</strong>, the PL/SQL Gateway looks for it in the &quot;sys&quot; schema.</td>
</tr>
<tr>
<td></td>
<td>The custom authorize function has the following signature:</td>
</tr>
<tr>
<td></td>
<td>FUNCTION AUTHORIZE RETURN BOOLEAN;</td>
</tr>
<tr>
<td></td>
<td>If the function returns TRUE, authentication succeeded. If it returns FALSE, authentication failed. If the authorize function is not defined, the Gateway returns an error and fails.</td>
</tr>
</tbody>
</table>

**Related Topics**

- **OWA_CUSTOM**

  The **OWA_CUSTOM** package provides a Global PLSQL Agent Authorization callback function.

### 229.2.6 SET_PROTECTION_REALM Procedure

This procedure sets the realm of the page that is returned to the user. The user enters a username and login that already exist in the realm.

**Syntax**

```sql
OWA_SEC.SET_PROTECTION_REALM(
    realm IN VARCHAR2);
```
Parameters

Table 229-3  SET_PROTECTION_REALM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realm</td>
<td>The realm where the page belongs. This string is displayed to the user.</td>
</tr>
</tbody>
</table>
The OWA_TEXT package contains subprograms used by OWA_PATTERN for manipulating strings. They are externalized so you can use them directly.

See Also:
For more information about implementation of this package:
- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

The chapter contains the following topics:
- Types
- Summary of OWA_TEXT Subprograms

230.1 OWA_OPT_LOCK Types

There are three OWA_OPT_LOCK lock types.

- MULTI_LINE DATA TYPE
- ROW_LIST DATA TYPE
- VC_ARR DATA TYPE

230.1.1 MULTI_LINE DATA TYPE

This datatype is a PL/SQL record that holds large amounts of text. The rows field, of type OWA_TEXT.VC_ARR DATA TYPE, contains the text data in the record.

TYPE multi_line IS RECORD (  
  rows           vc_arr,  
  num_rows       INTEGER,  
  partial_row    BOOLEAN);  

230.1.2 ROW_LIST DATA TYPE

This is the datatype for holding data to be processed.

TYPE row_list IS RECORD (  
  rows             int_arr,  
  num_rows         INTEGER);  

int_arr IS DEFINED AS:

TYPE int_arr IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
230.1.3 VC_ARR DATA TYPE

This is a component of the MULTI_LINE DATA TYPE and is used for holding large amounts of text.

```
TYPE vc_arr IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

Related Topics

- MULTI_LINE DATA TYPE
  This datatype is a PL/SQL record that holds large amounts of text. The rows field, of type OWA_TEXT.VC_ARR DATA TYPE, contains the text data in the record.

230.2 Summary of OWA_TEXT Subprograms

This table lists the OWA_TEXT subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD2MULTI Procedure</td>
<td>Adds text to an existing multi_line type</td>
</tr>
<tr>
<td>NEW_ROW_LIST Function and Procedure</td>
<td>Creates a new row_list</td>
</tr>
<tr>
<td>PRINT_MULTI Procedure</td>
<td>Prints out the contents of a multi_list</td>
</tr>
<tr>
<td>PRINT_ROW_LIST Procedure</td>
<td>Prints out the contents of a row_list</td>
</tr>
<tr>
<td>STREAM2MULTI Procedure</td>
<td>Converts a varchar2 to a multi_line type</td>
</tr>
</tbody>
</table>

230.2.1 ADD2MULTI Procedure

This procedure adds content to an existing Multi_Line Data Type

**Syntax**

```
OWA_TEXT.ADD2MULTI(
    stream         IN       VARCHAR2,
    mline          IN OUT   multi_line,
    continue       IN       BOOLEAN DEFAULT TRUE);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>The text to add.</td>
</tr>
<tr>
<td>mline</td>
<td>The OWA_TEXT.MULTI_LINE DATA TYPE. The output of this parameter contains stream.</td>
</tr>
</tbody>
</table>
Table 230-2 (Cont.) ADD2MULTI Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue</td>
<td>If TRUE, the procedure appends <em>stream</em> within the previous final row (assuming it is less than 32K). If FALSE, the procedure places <em>stream</em> in a new row.</td>
</tr>
</tbody>
</table>

### 230.2.2 NEW_ROW_LIST Function and Procedure

This function or procedure creates a new `OWA_TEX.ROW_LIST` data type.

The function version uses no parameters and returns a new empty `row_list`. The procedure version creates the `row_list` datatype as an output parameter.

**Syntax**

```
OWA_TEXT.NEW_ROW_LIST
RETURN ROW_LIST;
```

```
OWA_TEXT.NEW_ROW_LIST(    
  rlist OUT row_list);
```

**Parameters**

Table 230-3 NEW_ROW_LIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rlist</td>
<td>This is an output parameter containing the new <code>row_list</code> datatype</td>
</tr>
</tbody>
</table>

**Return Values**

The function version returns the new `row_list` datatype.

### 230.2.3 PRINT_MULTI Procedure

This procedure uses the PRINT Procedures or the PRN Procedures to print the "rows" field of the `OWA_TEXT.MULTI_LINE` data type.

**Syntax**

```
OWA_TEXT.PRINT_MULTI(    
  mline IN multi_line);
```

**Parameters**

**Table 230-4 PRINT_MULTI Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mline</td>
<td>The <code>multi_line</code> datatype to print.</td>
</tr>
</tbody>
</table>
Return Values

The contents of the multi_line.

Related Topics

• **PRINT Procedures**
  These procedures generate the specified parameter as a string terminated with the \n newline character.

• **PRN Procedures**
  These procedures generate the specified parameter as a string.

230.2.4 PRINT_ROW_LIST Procedure

This procedure uses the PRINT Procedures or the PRN Procedures to print the "rows" field of the OWA_TEXT.ROW_LIST DATA TYPE.

Syntax

```sql
OWA_TEXT.PRINT_ROW_LIST(
    rlist       IN       multi_line);
```

Parameters

**Table 230-5**  PRINT_ROW_LIST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rlist</td>
<td>The row_list datatype to print.</td>
</tr>
</tbody>
</table>

Return Values

The contents of the row_list.

Related Topics

• **PRINT Procedures**
  These procedures generate the specified parameter as a string terminated with the \n newline character.

• **PRN Procedures**
  These procedures generate the specified parameter as a string.

230.2.5 STREAM2MULTI Procedure

This procedure converts a string to a multi_line datatype.

Syntax

```sql
OWA_TEXT.STREAM2MULTI(
    stream         IN       VARCHAR2
    mline          OUT      multi_line);
```
### Parameters

#### Table 230-6  STREAM2MULTI Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stream</code></td>
<td>The string to convert.</td>
</tr>
<tr>
<td><code>mline</code></td>
<td>The stream in OWA_TEXT.MULTI_LINE DATA TYPE format</td>
</tr>
</tbody>
</table>
The OWA_UTIL package contains utility subprograms for performing operations such as getting the value of CGI environment variables, printing the data that is returned to the client, and printing the results of a query in an HTML table.

See Also:

For more information about implementation of this package:

• Oracle Fusion Middleware Administrator’s Guide for Oracle HTTP Server
• Oracle Fusion Middleware User’s Guide for mod_plsql

This chapter contains the following topics:

• Overview
• Types
• Summary of OWA_UTIL Subprograms

231.1 OWA_UTIL Overview

The OWA_UTIL package contains three types of utility subprograms.

• Dynamic SQL Utilities enable you to produce pages with dynamically generated SQL code.
• HTML utilities enable you to retrieve the values of CGI environment variables and perform URL redirects.
• Date utilities enable correct date-handling. Date values are simple strings in HTML, but are treated as a datatype by the Oracle database.

231.2 OWA_UTIL Types

OWA_UTIL uses Types to specify creating information.

• DATATYPE Datatype
• IDENT_ARR Datatype
• IP_ADDRESS Datatype
231.2.1 DATETYPE Datatype

The TODATE Function converts an item of this type to the type DATE, which is understood and properly handled as data by the database. The procedure CHOOSE_DATE Procedure enables the user to select the desired date.

```
TYPE dateType IS TABLE OF VARCHAR2(10) INDEX BY BINARY_INTEGER;
```

Related Topics

- **TODATE Function**
  This function converts the DATETYPE Datatype to the standard Oracle DATE type.

- **CHOOSE_DATE Procedure**
  This procedure generates three HTML form elements that allow the user to select the day, the month, and the year.

231.2.2 IDENT_ARR Datatype

This datatype is used for an array.

```
TYPE ident_arr IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

231.2.3 IP_ADDRESS Datatype

This datatype is used by the GET_CLIENT_IP Function in the OWA_SEC package.

```
TYPE ip_address IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
```

Related Topics

- **GET_CLIENT_IP Function**
  This function returns the IP address of the client.

- **OWA_SEC**
  The OWA_SEC package provides an interface for custom authentication.

231.3 Summary of OWA_UTIL Subprograms

This table lists the OWA_UTIL subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIND_VARIABLES Function</td>
<td>Prepares a SQL query and binds variables to it</td>
</tr>
<tr>
<td>CALENDARPRINT Procedures</td>
<td>Prints a calendar</td>
</tr>
<tr>
<td>CELLSPRINT Procedures</td>
<td>Prints the contents of a query in an HTML table</td>
</tr>
<tr>
<td>CHOOSE_DATE Procedure</td>
<td>Generates HTML form elements that allow the user to select a date</td>
</tr>
<tr>
<td>GET_CGI_ENV Function</td>
<td>Returns the value of the specified CGI environment variable</td>
</tr>
<tr>
<td>GET_OWA_SERVICE_PATH Function</td>
<td>Returns the full virtual path for the PL/SQL Gateway</td>
</tr>
</tbody>
</table>
Table 231-1  (Cont.) OWA_UTIL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_PROCEDURE Function</td>
<td>Returns the name of the procedure that is invoked by the PL/SQL Gateway</td>
</tr>
<tr>
<td>HTTP_HEADER_CLOSE Procedure</td>
<td>Closes the HTTP header</td>
</tr>
<tr>
<td>LISTPRINT Procedure</td>
<td>Generates a HTML form element that contains data from a query</td>
</tr>
<tr>
<td>MIME_HEADER Procedure</td>
<td>Generates the Content-type line in the HTTP header</td>
</tr>
<tr>
<td>PRINT_CGI_ENV Procedure</td>
<td>Generates a list of all CGI environment variables and their values</td>
</tr>
<tr>
<td>REDIRECT_URL Procedure</td>
<td>Generates the Location line in the HTTP header</td>
</tr>
<tr>
<td>SHOWPAGE Procedure</td>
<td>Prints a page generated by the HTP and HTF packages in SQL*Plus</td>
</tr>
<tr>
<td>SHOWSOURCE Procedure</td>
<td>Prints the source for the specified subprogram</td>
</tr>
<tr>
<td>SIGNATURE procedure</td>
<td>Prints a line that says that the page is generated by the PL/SQL Agent</td>
</tr>
<tr>
<td>STATUS_LINE Procedure</td>
<td>Generates the Status line in the HTTP header</td>
</tr>
<tr>
<td>TABLEPRINT Function</td>
<td>Prints the data from a table in the database as an HTML table</td>
</tr>
<tr>
<td>TODATE Function</td>
<td>Converts date type data to the standard PL/SQL date type</td>
</tr>
<tr>
<td>WHO_CALLED_ME Procedure</td>
<td>Returns information on the caller of the procedure.</td>
</tr>
</tbody>
</table>

231.3.1 BIND_VARIABLES Function

This function prepares a SQL query by binding variables to it, and stores the output in an opened cursor. Use this function as a parameter to a procedure sending a dynamically generated query. Specify up to 25 bind variables.

Syntax

```sql
OWA_UTIL.BIND_VARIABLES(
    theQuery       IN       VARCHAR2   DEFAULT NULL,
    bv1Name        IN       VARCHAR2   DEFAULT NULL,
    bv1Value       IN       VARCHAR2   DEFAULT NULL,
    bv2Name        IN       VARCHAR2   DEFAULT NULL,
    bv2Value       IN       VARCHAR2   DEFAULT NULL,
    bv3Name        IN       VARCHAR2   DEFAULT NULL,
    bv3Value       IN       VARCHAR2   DEFAULT NULL,
    ...
    bv25Name       IN       VARCHAR2   DEFAULT NULL,
    bv25Value      IN       VARCHAR2   DEFAULT NULL)
RETURN INTEGER;
```
Parameters

Table 231-2  BIND_VARIABLES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theQuery</td>
<td>The SQL query statement which must be a SELECT statement</td>
</tr>
<tr>
<td>bv1Name</td>
<td>The name of the variable</td>
</tr>
<tr>
<td>bv1Value</td>
<td>The value of the variable</td>
</tr>
</tbody>
</table>

Return Values

An integer identifying the opened cursor.

231.3.2 CALENDARPRINT Procedures

These procedures creates a calendar in HTML with a visible border. Each date in the calendar can contain any number of hypertext links.

This procedure has 2 versions.

- Version 1 uses a hard-coded query stored in a varchar2 string.
- Version 2 uses a dynamic query prepared with the BIND_VARIABLES Function.

Syntax

```
OWA_UTIL.CALENDARPRINT(
    p_query        IN       VARCHAR2,
    p_mf_only      IN       VARCHAR2   DEFAULT 'N');
```

```
OWA_UTIL.CALENDARPRINT(
    p_cursor       IN       INTEGER,
    p_mf_only      IN       VARCHAR2   DEFAULT 'N');
```

Parameters

Table 231-3  CALENDARPRINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_query</td>
<td>A PL/SQL query.</td>
</tr>
<tr>
<td>p_cursor</td>
<td>A PL/SQL cursor containing the same format as p_query.</td>
</tr>
<tr>
<td>p_mf_only</td>
<td>If &quot;N&quot; (the default), the generated calendar includes Sunday through Saturday. Otherwise, it includes Monday through Friday only.</td>
</tr>
</tbody>
</table>

Usage Notes

Design your query as follows:

- The first column is a DATE. This correlates the information produced by the query with the calendar output generated by the procedure.
- The query output must be sorted on this column using ORDER BY.
• The second column contains the text, if any, that you want printed for that date.

• The third column contains the destination for generated links. Each item in the second column becomes a hypertext link to the destination given in this column. If this column is omitted, the items in the second column are simple text, not links.

231.3.3 CELLSPRINT Procedures

This procedure generates an HTML table from the output of a SQL query. SQL atomic data items are mapped to HTML cells and SQL rows to HTML rows. You must write the code to begin and end the HTML table.

There are nine versions of this procedure:

• The first version passes the results of a query into an index table. Perform the query and CELLSPRINT does the formatting. To have more control in generating an HTML table from the output of an SQL query, use the FORMAT_CELL Function in the "HTF" package.

• The second and third versions display rows (up to the specified maximum) returned by the query or cursor.

• The fourth and fifth versions exclude a specified number of rows from the HTML table. Use the fourth and fifth versions to scroll through result sets by saving the last row seen in a hidden form element.

• The sixth through ninth versions are the same as the first four versions, except that they return a row count output parameter.

Syntax

```sql
OWA_UTIL.CELLSPRINT(
p_colCnt          IN    INTEGER,
p_resultTbl       IN    vc_arr,
p_format_numbers  IN    VARCHAR2   DEFAULT NULL);

OWA_UTIL.CELLSPRINT(
p_theQuery        IN    VARCHAR2,
p_max_rows        IN    NUMBER     DEFAULT 100,
p_format_numbers  IN    VARCHAR2   DEFAULT NULL);

OWA_UTIL.CELLSPRINT(
p_theCursor       IN    INTEGER,
p_max_rows        IN    NUMBER     DEFAULT 100,
p_format_numbers  IN    VARCHAR2   DEFAULT NULL);

OWA_UTIL.CELLSPRINT(
p_theQuery        IN    VARCHAR2,
p_max_rows        IN    NUMBER     DEFAULT 100,
p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
p_skip_rec        IN    NUMBER      DEFAULT 0,
p_more_data       OUT   BOOLEAN);

OWA_UTIL.CELLSPRINT(
p_theCursor       IN    INTEGER,
p_max_rows        IN    NUMBER     DEFAULT 100,
p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
p_skip_rec        IN    NUMBER      DEFAULT 0,
p_more_data       OUT   BOOLEAN);
```

Chapter 231
OWA_UTIL.CELLSPRINT(
    p_theQuery        IN    VARCHAR2,
    p_max_rows        IN    NUMBER     DEFAULT 100,
    p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
    p_reccnt          OUT   NUMBER);  

OWA_UTIL.CELLSPRINT(
    p_theCursor       IN    INTEGER,
    p_max_rows        IN    NUMBER     DEFAULT 100,
    p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
    p_reccnt          OUT   NUMBER);  

OWA_UTIL.CELLSPRINT(
    p_theQuery        IN    VARCHAR2,
    p_max_rows        IN    NUMBER     DEFAULT 100,
    p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
    p_skip_rec        IN    NUMBER     DEFAULT 0,
    p_more_data       OUT   BOOLEAN
    p_reccnt          OUT   NUMBER);  

OWA_UTIL.CELLSPRINT(
    p_theCursor       IN    INTEGER,
    p_max_rows        IN    NUMBER     DEFAULT 100,
    p_format_numbers  IN    VARCHAR2   DEFAULT NULL,
    p_skip_rec        IN    NUMBER     DEFAULT 0,
    p_more_data       OUT   BOOLEAN,
    p_reccnt          OUT   NUMBER);  

Parameters

Table 231-4  CELLSPRINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_query</td>
<td>A PL/SQL query.</td>
</tr>
<tr>
<td>p_colCnt</td>
<td>The number of columns in the table.</td>
</tr>
<tr>
<td>p_theQuery</td>
<td>A SQL SELECT statement.</td>
</tr>
<tr>
<td>p_theCursor</td>
<td>A cursor ID. This can be the return value from the BIND_VARIABLES Function.</td>
</tr>
<tr>
<td>p_max_rows</td>
<td>The maximum number of rows to print.</td>
</tr>
<tr>
<td>p_format_numbers</td>
<td>If the value of this parameter is not NULL, number fields are right</td>
</tr>
<tr>
<td></td>
<td>justified and rounded to two decimal places.</td>
</tr>
<tr>
<td>p_skip_rec</td>
<td>The number of rows to exclude from the HTML table.</td>
</tr>
<tr>
<td>p_more_data</td>
<td>TRUE if there are more rows in the query or cursor, FALSE otherwise.</td>
</tr>
<tr>
<td>p_reccnt</td>
<td>The number of rows that have been returned by the query. This value</td>
</tr>
<tr>
<td></td>
<td>does not include skipped rows (if any).</td>
</tr>
<tr>
<td>p_resultTbl</td>
<td>The index table which will contain the result of the query. Each entry</td>
</tr>
<tr>
<td></td>
<td>in the query will correspond to one column value.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates
231.3.4 CHOOSE_DATE Procedure

This procedure generates three HTML form elements that allow the user to select the day, the month, and the year.

Syntax

```
OWA_UTIL.CHOOSE_DATE(
  p_name    IN    VARCHAR2,
  p_date    IN    DATE       DEFAULT SYSDATE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>The name of the form elements.</td>
</tr>
<tr>
<td>p_date</td>
<td>The initial date that is selected when the HTML page is displayed.</td>
</tr>
</tbody>
</table>

Usage Notes

- The parameter in the procedure that receives the data from these elements must be a `GET_CGI_ENV` Function.
- Use the `TODATE` Function to convert the `GET_CGI_ENV` Function value to the standard Oracle `DATE` datatype.

Examples

```
<SELECT NAME="p_name" SIZE="1">
  <OPTION value="01">1
  ...
  <OPTION value="31">31
</SELECT>
-<SELECT NAME="p_name" SIZE="1">
  <OPTION value="01">JAN
  ...
  <OPTION value="12">DEC
</SELECT>
-<SELECT NAME="p_name" SIZE="1">
  <OPTION value="1992">1992
  ...
  <OPTION value="2002">2002
</SELECT>
```
231.3.5 GET_CGI_ENV Function

This function returns the value of the specified CGI environment variable.

Syntax

```sql
OWA_UTIL.GET_CGI_ENV(
    param_name       IN      VARCHAR2)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param_name</td>
<td>The name of the CGI environment variable. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Return Values

The value of the specified CGI environment variable. If the variable is not defined, the function returns NULL.

231.3.6 GET_OWA_SERVICE_PATH Function

This function returns the full virtual path of the PL/SQL Gateway that is handling the request.

Syntax

```sql
OWA_UTIL.GET_OWA_SERVICE_PATH
RETURN VARCHAR2;
```

Return Values

A virtual path of the PL/SQL Gateway that is handling the request.

231.3.7 GET_PROCEDURE Function

This function returns the name of the procedure that is being invoked by the PL/SQL Gateway.

Syntax

```sql
OWA_UTIL.GET_PROCEDURE
RETURN VARCHAR2;
```

Return Values

The name of a procedure, including the package name if the procedure is defined in a package.
231.3.8 HTTP_HEADER_CLOSE Procedure

This procedure generates a newline character to close the HTTP header.

Syntax

```
OWA_UTIL.HTTP_HEADER_CLOSE;
```

Return Values

A newline character, which closes the HTTP header.

Usage Notes

- Use this procedure if you have not closed the header by using the `bclose_header` parameter in calls such as `MIME_HEADER Procedure`, `REDIRECT_URL Procedure`, or `STATUS_LINE Procedure`.
- The HTTP header must be closed before any `HTP.PRINT` or `HTP.PRN` calls.

231.3.9 LISTPRINT Procedure

This procedure generates an HTML selection list form element from the output of a SQL query.

There are two versions of this procedure.

- The first version contains a hard-coded SQL query.
- The second version uses a dynamic query prepared with the `BIND_VARIABLES` Function.

Syntax

```
OWA_UTIL.LISTPRINT(
    p_theQuery     IN       VARCHAR2,
    p_cname        IN       VARCHAR2,
    p_nsize        IN       NUMBER,
    p_multiple     IN       BOOLEAN   DEFAULT FALSE);

OWA_UTIL.LISTPRINT(
    p_theCursor    IN    INTEGER,
    p_cname        IN       VARCHAR2,
    p_nsize        IN       NUMBER,
    p_multiple     IN       BOOLEAN   DEFAULT FALSE);
```

Parameters

Table 231-7  LISTPRINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_theQuery</td>
<td>The SQL query.</td>
</tr>
<tr>
<td>p_theCursor</td>
<td>The cursor ID. This can be the return value from the <code>BIND_VARIABLES</code> Function.</td>
</tr>
<tr>
<td>p_cname</td>
<td>The name of the HTML form element.</td>
</tr>
</tbody>
</table>
Table 231-7  (Cont.) LISTPRINT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_nsize</td>
<td>The size of the form element (this controls how many items the user can see without scrolling).</td>
</tr>
<tr>
<td>p_multiple</td>
<td>Whether multiple selection is permitted.</td>
</tr>
</tbody>
</table>

Usage Notes

The columns in the output of the query are handled in the following manner:

- The first column specifies the values that are sent back. These values are for the VALUE attribute of the OPTION tag.
- The second column specifies the values that the user sees.
- The third column specifies whether or not the row is marked as SELECTED in the OPTION tag. If the value is not NULL, the row is selected.

Examples

```<SELECT NAME="p_cname" SIZE="p_nsize">
  <OPTION SELECTED value='value_from_the_first_column'>value_from_the_second_column
  <OPTION SELECTED value='value_from_the_first_column'>value_from_the_second_column
  ...
</SELECT>```

231.3.10 MIME_HEADER Procedure

This procedure changes the default MIME header that the script returns. This procedure must come before any HTP.PRINT or HTP.PRNT calls to direct the script not to use the default MIME header.

Syntax

```OWA_UTIL.MIME_HEADER(
  ccontent_type    IN       VARCHAR2   DEFAULT 'text/html',
  bclose_header    IN       BOOLEAN    DEFAULT TRUE,
  ccharset         IN       VARCHAR2   DEFAULT NULL);
```

Parameters

Table 231-8  MIME_HEADER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccontent_type</td>
<td>The MIME type to generate</td>
</tr>
<tr>
<td>bclose_header</td>
<td>Whether or not to close the HTTP header. If TRUE, two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.</td>
</tr>
<tr>
<td>ccharset</td>
<td>The character set to use. The character set only makes sense if the MIME type is of type 'text'. Therefore, the character set is only tagged on to the Content-Type header only if the MIME type passed in is of type 'text'. Any other MIME type, such as 'image', will not have any character set tagged on.</td>
</tr>
</tbody>
</table>
Examples

Content-type: <ccontent_type>; charset=<ccharset>
so that

owa_util.mime_header('text/plain', false, 'ISO-8859-4')

generates

Content-type: text/plain; charset=ISO-8859-4

231.3.11 PRINT_CGI_ENV Procedure

This procedure generates all the CGI environment variables and their values made available by the PL/SQL Gateway to the stored procedure.

Syntax

OWA_UTIL.PRINT_CGI_ENV;

Examples

This procedure generates a list in the following format:

cgi_env_var_name = value

231.3.12 REDIRECT_URL Procedure

This procedure specifies that the application server is to visit the specified URL. The URL may specify either a Web page to return or a program to execute.

Syntax

OWA_UTIL.REDIRECT_URL(
    curl           IN       VARCHAR2
    bclose_header  IN       BOOLEAN    DEFAULT TRUE);

Parameters

Table 231-9  REDIRECT_URL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>The URL to visit.</td>
</tr>
<tr>
<td>bclose_header</td>
<td>Whether or not to close the HTTP header. If TRUE, two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.</td>
</tr>
</tbody>
</table>

Usage Notes

This procedure must come before any HTP procedure or HTF function call.

Examples

This procedure generates
231.3.13 SHOWPAGE Procedure

This procedure prints out the HTML output of a procedure in SQL*Plus.

The procedure must use the HTP or HTF packages to generate the HTML page, and this procedure must be issued after the HTP or HTF page-generating subprogram has been called and before any other HTP or HTF subprograms are directly or indirectly called.

Syntax

OWA_UTIL.SHOWPAGE;

Usage Notes

• This method is useful for generating pages filled with static data.
• This procedure uses the DBMS_OUTPUT package and is limited to 32767 characters for each line and an overall buffer size of 1,000,000 bytes.

Examples

The output of htp procedure is displayed in SQL*Plus, SQL*DBA, or Oracle Server Manager. For example:

SQL> set serveroutput on
SQL> spool gretzky.html
SQL> execute hockey.pass("Gretzky")
SQL> execute owa_util.showpage
SQL> exit

This would generate an HTML page that could be accessed from Web browsers.

231.3.14 SHOWSOURCE Procedure

This procedure prints the source of the specified procedure, function, or package. If a procedure or function which belongs to a package is specified, then the entire package is displayed.

Syntax

OWA_UTIL.SHOWSOURCE (cname IN VARCHAR2);

Parameters

Table 231-10  SHOWSOURCE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The function or procedure whose source you want to show.</td>
</tr>
</tbody>
</table>
231.3.15 SIGNATURE procedure

This procedure generates an HTML line followed by a signature line on the HTML document. If a parameter is specified, the procedure also generates a hypertext link to view the PL/SQL source for that procedure. The link calls the SHOWSOURCE Procedure.

Syntax

OWA_UTIL.SIGNATURE;

OWA_UTIL.SIGNATURE (cname IN VARCHAR2);

Parameters

Table 231-11 SIGNATURE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>The function or procedure whose source you want to show.</td>
</tr>
</tbody>
</table>

Examples

Without a parameter, the procedure generates a line that looks like the following:

This page was produced by the PL/SQL Agent on August 9, 2001 09:30.

With a parameter, the procedure generates a signature line in the HTML document that looks like the following:

This page was produced by the PL/SQL Agent on 8/09/01 09:30
View PL/SQL Source

231.3.16 STATUS_LINE Procedure

This procedure sends a standard HTTP status code to the client. This procedure must come before any \texttt{htp.print} or \texttt{htp.prn} calls so that the status code is returned as part of the header, rather than as "content data".

Syntax

OWA_UTIL.STATUS_LINE(nstatus IN INTEGER, creason IN VARCHAR2 DEFAULT NULL, bclose_header IN BOOLEAN DEFAULT TRUE);

Parameters

Table 231-12 STATUS_LINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nstatus</td>
<td>The status code.</td>
</tr>
<tr>
<td>creason</td>
<td>The string for the status code.</td>
</tr>
</tbody>
</table>
Table 231-12  (Cont.) STATUS_LINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bclose_header</td>
<td>Whether or not to close the HTTP header. If TRUE, two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.</td>
</tr>
</tbody>
</table>

Examples

This procedure generates

```
Status: <nstatus> <creason>
```

231.3.17 TABLEPRINT Function

This function generates either preformatted tables or HTML tables (depending on the capabilities of the user's browser) from database tables.

Syntax

```sql
OWA_UTIL.TABLEPRINT(
    ctable         IN       VARCHAR2,
    cattributes    IN       VARCHAR2   DEFAULT NULL,
    ntable_type    IN       INTEGER    DEFAULT HTML_TABLE,
    ccolumns       IN       VARCHAR2   DEFAULT '*',
    cclauses       IN    VARCHAR2   DEFAULT NULL,
    ccol_aliases   IN       VARCHAR2   DEFAULT NULL,
    nrow_min       IN       NUMBER     DEFAULT 0,
    nrow_max       IN       NUMBER     DEFAULT NULL)
RETURN BOOLEAN;
```

Parameters

Table 231-13  TABLEPRINT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctable</td>
<td>The database table.</td>
</tr>
<tr>
<td>cattributes</td>
<td>Other attributes to be included as-is in the tag.</td>
</tr>
<tr>
<td>ntable_type</td>
<td>How to generate the table. Specify HTML_TABLE to generate the table using <code>&lt;TABLE&gt;</code> tags or PRE_TABLE to generate the table using the <code>&lt;PRE&gt;</code> tags. These are constants:</td>
</tr>
<tr>
<td>ccolumns</td>
<td>A comma-delimited list of columns from ctable to include in the generated table.</td>
</tr>
<tr>
<td>cclauses</td>
<td>WHERE or ORDER BY clauses, which specify which rows to retrieve from the database table, and how to order them.</td>
</tr>
<tr>
<td>ccol_aliases</td>
<td>A comma-delimited list of headings for the generated table.</td>
</tr>
<tr>
<td>nrow_min</td>
<td>The first row, of those retrieved, to display.</td>
</tr>
<tr>
<td>nrow_max</td>
<td>The last row, of those retrieved, to display.</td>
</tr>
</tbody>
</table>
Return Values

Returns TRUE if there are more rows beyond the nrow_max requested, FALSE otherwise.

Usage Notes

- RAW columns are supported, but LONG RAW columns are not. References to LONG RAW columns will print the result 'Not Printable'.
- Note that in this function, cattributes is the second rather than the last parameter.

Examples

For browsers that do not support HTML tables, create the following procedure:

```sql
CREATE OR REPLACE PROCEDURE showemps IS
    ignore_more BOOLEAN;
BEGIN
    ignore_more := OWA_UTIL.TABLEPRINT('emp', 'BORDER', OWA_UTIL.PRE_TABLE);
END;
```

Requesting a URL such as

http://myhost:7777/pls/hr/showemps

returns to the following to the client:

```
<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>CLERK</td>
<td>7902</td>
<td>17-DEC-80</td>
<td>800</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>20-FEB-81</td>
<td>1600</td>
<td>300</td>
<td>30</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>SALESMAN</td>
<td>7698</td>
<td>22-FEB-81</td>
<td>1250</td>
<td>500</td>
<td>30</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
<td>7839</td>
<td>02-APR-81</td>
<td>2975</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>SALESMAN</td>
<td>7698</td>
<td>28-SEP-81</td>
<td>1250</td>
<td>1400</td>
<td>30</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>MANAGER</td>
<td>7839</td>
<td>01-MAY-81</td>
<td>2850</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>MANAGER</td>
<td>7839</td>
<td>09-JUN-81</td>
<td>2450</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>7566</td>
<td>09-DEC-82</td>
<td>3000</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>PRESIDENT</td>
<td>7788</td>
<td>17-NOV-81</td>
<td>5000</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>SALESMAN</td>
<td>7698</td>
<td>08-SEP-81</td>
<td>1500</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>CLERK</td>
<td>7788</td>
<td>12-JAN-83</td>
<td>1100</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>CLERK</td>
<td>7698</td>
<td>03-DEC-81</td>
<td>950</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
<td>7566</td>
<td>03-DEC-81</td>
<td>3000</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>CLERK</td>
<td>7782</td>
<td>23-JAN-82</td>
<td>1300</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
```

To view the employees in department 10, and only their employee ids, names, and salaries, create the following procedure:

```sql
CREATE OR REPLACE PROCEDURE showemps_10 IS
    ignore_more BOOLEAN;
BEGIN
    ignore_more := OWA_UTIL.TABLEPRINT('emp', 'BORDER', OWA_UTIL.PRE_TABLE, 'empno, ename, sal', 'WHERE deptno=10 ORDER BY empno', 'Employee Number, Name, Salary');
END;
```

A request for a URL like
would return the following to the client:

```
<table>
<thead>
<tr>
<th>Employee Number</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>2450</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>5000</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>1300</td>
</tr>
</tbody>
</table>
```

For browsers that support HTML tables, to view the department table in an HTML table, create the following procedure:

```sql
CREATE OR REPLACE PROCEDURE showdept IS
    ignore_more BOOLEAN;
BEGIN
    ignore_more := oWA_UTIL.TABLEPRINT('dept', 'BORDER');
END;
```

A request for a URL like

http://myhost:7777/pls/hr/showdept

would return the following to the client:

```
<TABLE BORDER>
  <TR>
    <TH>DEPTNO</TH>
    <TH>DNAME</TH>
    <TH>LOC</TH>
  </TR>
  <TR>
    <TD ALIGN="LEFT">10</TD>
    <TD ALIGN="LEFT">ACCOUNTING</TD>
    <TD ALIGN="LEFT">NEW YORK</TD>
  </TR>
  <TR>
    <TD ALIGN="LEFT">20</TD>
    <TD ALIGN="LEFT">RESEARCH</TD>
    <TD ALIGN="LEFT">DALLAS</TD>
  </TR>
  <TR>
    <TD ALIGN="LEFT">30</TD>
    <TD ALIGN="LEFT">SALES</TD>
    <TD ALIGN="LEFT">CHICAGO</TD>
  </TR>
  <TR>
    <TD ALIGN="LEFT">40</TD>
    <TD ALIGN="LEFT">OPERATIONS</TD>
    <TD ALIGN="LEFT">BOSTON</TD>
  </TR>
</TABLE>
```

A Web browser would format this to look like the following table:
### 231.3.18 TODATE Function

This function converts the DATETYPE Datatype to the standard Oracle DATE type.

**Syntax**

```sql
OWA_UTIL.TODATE(
    p_dateArray      IN     dateType)
RETURN DATE;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_dateArray</td>
<td>The value to convert.</td>
</tr>
</tbody>
</table>

**Related Topics**

- **DATETYPE Datatype**
  
The TODATE Function converts an item of this type to the type DATE, which is understood and properly handled as data by the database. The procedure CHOOSE_DATE Procedure enables the user to select the desired date.

### 231.3.19 WHO_CALLED_ME Procedure

This procedure returns information (in the form of output parameters) about the PL/SQL code unit that invoked it.

**Syntax**

```sql
OWA_UTIL.WHO_CALLED_ME(
    owner          OUT      VARCHAR2,
    name           OUT      VARCHAR2,
    lineno         OUT      NUMBER,
    caller_t       OUT      VARCHAR2);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>The owner of the program unit.</td>
</tr>
</tbody>
</table>
Table 231-15  (Cont.) WHO_CALLED_ME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the program unit. This is the name of the package, if the calling program unit is wrapped in a package, or the name of the procedure or function if the calling program unit is a standalone procedure or function. If the calling program unit is part of an anonymous block, this is <strong>NULL</strong>.</td>
</tr>
<tr>
<td>lineno</td>
<td>The line number within the program unit where the call was made.</td>
</tr>
<tr>
<td>caller_t</td>
<td>The type of program unit that made the call. The possibilities are: package body, anonymous block, procedure, and function. Procedure and function are only for standalone procedures and functions.</td>
</tr>
</tbody>
</table>
The **SDO_CS** package contains functions and procedures for working with coordinate systems.

You can perform explicit coordinate transformations on a single geometry or an entire layer of geometries (that is, all geometries in a specified column in a table).

For a complete description of this package within the context of Oracle Spatial, see **SDO_CS** in the *Oracle Spatial and Graph Developer's Guide*. 
The SDO_CSW_PROCESS package contains subprograms for various processing operations related to support for Catalog Services for the Web (CSW).

For a complete description of this package within the context of Oracle Spatial, see SDO_CSW_PROCESS in the Oracle Spatial and Graph Developer's Guide.
SDO_GCDR

The SDO_GCDR package contains the Oracle Spatial geocoding subprograms, which let you geocode unformatted postal addresses.

For a complete description of this package within the context of Oracle Spatial, see SDO_GCDR in the Oracle Spatial and Graph Developer's Guide.
The SDO_GEOM package contains the geometry functions.

These can be grouped into the following categories (with examples of each):

- **Relationship (True/False) between two objects**: RELATE, WITHIN_DISTANCE
- **Validation**: VALIDATE_GEOMETRY_WITH_CONTEXT, VALIDATE_LAYER_WITH_CONTEXT
- **Single-object operations**: SDO_ARC_DENSIFY, SDO_AREA, SDO_BUFFER, SDO_CENTROID, SDO_CONVEXHULL, SDO_LENGTH, SDO_MBR, SDO_POINTONSURFACE
- **Two-object operations**: SDO_DISTANCE, SDO_DIFFERENCE, SDO_INTERSECTION, SDO_UNION, SDO_XOR

For a complete description of this package within the context of Oracle Spatial, see SDO_GEOM in the *Oracle Spatial and Graph Developer’s Guide*. 
The `SDO_GEOR` package contains functions and procedures for the Oracle Spatial GeoRaster feature, which lets you store, index, query, analyze, and deliver raster image data and its associated spatial vector geometry data and metadata.

For complete description of this package within the context of Oracle Spatial, see `SDO_GEOR` in the *Oracle Spatial and Graph GeoRaster Developer's Guide*. 
The `SDO_GEOR_ADMIN` package contains subprograms for administrative operations related to GeoRaster.

For a complete description of this package within the context of Oracle Spatial, see `SDO_GEOR_ADMIN` in the *Oracle Spatial and Graph GeoRaster Developer's Guide*. 
The SDO_GEOR_AGGR package provides an interface to the SDO_GEOR_AGGR package for performing aggregate operations on GeoRaster objects.

For a complete description of this package within the context of Oracle Spatial, see SDO_GEOR_AGGR in the Oracle Spatial and Graph GeoRaster Developer’s Guide.
The `SDO_GEOR_RA` package provides an interface to the `SDO_GEOR_RA` package for performing raster algebra and analytic operations related to GeoRaster.

For a complete description of this package within the context of Oracle Spatial, see `SDO_GEOR_RA` in the *Oracle Spatial and Graph GeoRaster Developer's Guide*. 
The SDO_GEOR_UTL package contains utility functions and procedures for the Oracle Spatial GeoRaster feature, including those related to using triggers with GeoRaster data.

For complete description of this package within the context of Oracle Spatial, see SDO_GEOR_UTL in the Oracle Spatial and Graph GeoRaster Developer's Guide.
The `SDO_LRS` package contains functions that create, modify, query, and convert linear referencing elements.

For a complete description of this package within the context of Oracle Spatial, see `SDO_LRS` in the *Oracle Spatial and Graph Developer's Guide*. 
The SDO_MIGRATE package lets you upgrade geometry tables from previous releases of Oracle Spatial.

For a complete description of this package within the context of Oracle Spatial, see SDO_MIGRATE in the Oracle Spatial and Graph Developer’s Guide.
The SDO_NET package contains functions and procedures for working with data modeled as nodes and links in a network.

For a complete description of this package within the context of Oracle Spatial, see SDO_NET in the Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide.
The SDO_NFE package contains functions and procedures for performing network feature editing.

For a complete description of this package within the context of Oracle Spatial and Graph, see SDO_NFE in the Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide.
The `SDO_OLS` package contains subprograms for Spatial OpenLS support.

For a complete description of this package within the context of Oracle Spatial, see `SDO_OLS` in the *Oracle Spatial and Graph Developer’s Guide*.
The SDO_PC_PKG package contains subprograms to support the use of point clouds in Spatial.

For a complete description of this package within the context of Oracle Spatial, see SDO_PC_PKG in the Oracle Spatial and Graph Developer's Guide.
The SDO_SAM package contains functions and procedures for spatial analysis and data mining.

For a complete description of this package within the context of Oracle Spatial, see SDO_SAM in the Oracle Spatial and Graph Developer's Guide.
The `SDO_TIN_PKG` package contains subprograms to support the use of triangulated irregular networks (TINs) in Spatial.

For a complete description of this package within the context of Oracle Spatial, see `SDO_TIN_PKG` in the
The `SDO_TOPO` package contains subprograms for creating and managing Oracle Spatial topologies.

For a complete description of this package within the context of Oracle Spatial, see `SDO_TOPO` in the *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*. 
The SDO_TOPO_MAP package contains subprograms for editing Oracle Spatial topologies using a cache (TopoMap object).

For a complete description of this package within the context of Oracle Spatial, see SDO_TOPO_MAP in the Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide.
The SDO_TUNE package contains Spatial tuning functions and procedures.

For complete description of this package within the context of Oracle Spatial, see SDO_TUNE in the Oracle Spatial and Graph Developer's Guide.
252

SDO_UTIL

The SDO_UTIL package contains the utility functions and procedures for Oracle Spatial.

For complete description of this package within the context of Oracle Spatial, see SDO_UTIL in the Oracle Spatial and Graph Developer's Guide.
The SDO_WFS_LOCK package contains subprograms for WFS support for registering and unregistering feature tables. Registering a feature table enables the table for WFS transaction locking; unregistering a feature table disables the table for WFS transaction locking.

For a complete description of this package within the context of Oracle Spatial, see SDO_WFS_LOCK in the Oracle Spatial and Graph Developer's Guide.
The `SDO_WFS_PROCESS` package contains subprograms for various processing operations related to support for Web Feature Services.

For a complete description of this package within the context of Oracle Spatial, see `SDO_WFS_PROCESS` in the *Oracle Spatial and Graph Developer's Guide*. 
The SEM_APIS package contains subprograms for working with the Resource Description Framework (RDF) and Web Ontology Language (OWL) in an Oracle database.

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM_APIS in the Oracle Spatial and Graph RDF Semantic Graph Developer's Guide.
The `SEM_OLS` package provides an interface to the `SEM_OLS` package for providing triple-level security to RDF data, using Oracle Label Security (OLS).

For a complete description of this package, see `SEM_OLS` in the *Oracle Spatial and Graph RDF Semantic Graph Developer's Guide*. 

---

**Oracle**
SEM_PERF

The SEM_PERF package contains subprograms for examining and enhancing the performance of the Resource Description Framework (RDF) and Web Ontology Language (OWL) support in an Oracle database.

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM_PERF in the Oracle Spatial and Graph RDF Semantic Graph Developer's Guide.
SEM_RDFCTX

The SEM_RDFCTX package contains subprograms for managing extractor policies and semantic indexes created for documents.

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM_RDFCTX in the Oracle Spatial and Graph RDF Semantic Graph Developer's Guide.
SEM_RDFSA

The SEM_RDFSA package contains subprograms for providing fine-grained access control to RDF data, using either a virtual private database (VPD) or Oracle Label Security (OLS).

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM_RDFSA in the Oracle Spatial and Graph RDF Semantic Graph Developer's Guide.
UTL_CALL_STACK

The UTL_CALL_STACK package provides an interface to provide information about currently executing subprograms.

Functions return subprogram names, unit names, owner names, edition names, and line numbers for given dynamic depths. Other functions return error stack information.

This chapter contains the following topics:

- Overview
- Security Model
- UTL_CALL_STACK Operational Notes
- UTL_CALL_STACK Exceptions
- Data Structures
- Summary of UTL_CALL_STACK Subprograms

See Also:

- Oracle Database PL/SQL Language Reference regarding Conditional Compilation
- Oracle Database Development Guide regarding Using PL/Scope and Using the PL/SQL Hierarchical Profiler

260.1 UTL_CALL_STACK Overview

The UTL_CALL_STACK package provides an interface for PL/SQL programmers to obtain information about currently executing programs including the subprogram name from dynamic and lexical stacks and the depths of those stacks.

Individual functions return subprogram names, unit names, owner names, edition names, and line numbers for given dynamic depths. More functions return error stack information. Such information can be used to create more revealing error logs and application execution traces.

Dynamic Depth

The dynamic depth of an executing instance of a PL/SQL subprogram is defined recursively.

- The dynamic depth of the currently executing subprogram instance is one.
- Otherwise, the dynamic depth of the subprogram instance is one more than the dynamic depth of the subprogram it invoked.
- If there is a SQL, Java, or other non-PL/SQL context that invoked or was invoked by an executing subprogram, it occupies a level on the call stack as if it were a subprogram.
In the case of a call stack in which A calls B, which calls C, which calls D, which calls E, which calls F, which calls E, this stack can be written as a line with the dynamic depths underneath:

A B C D E F E
7 6 5 4 3 2 1

Lexical Depth

The lexical depth of a PL/SQL subprogram is defined recursively.

- The lexical depth of a unit, an anonymous block, trigger, or ADT is one (1).
- The lexical depth of a subprogram defined within another object is one plus the lexical depth of that object.

Blocks do not affect lexical depth.

Error Depth

The error depth is the number of errors on the error stack.

For example, consider the following anonymous block.

```
BEGIN
  BEGIN
    ... (1)
    raise zero_divide;
  EXCEPTION
    when others then
    raise no_data_found;
  END;
  EXCEPTION
    WHEN others THEN
    ... (2)
END;
```

The error depth at (1) is zero and at (2) is two.

Backtrace

The backtrace is a trace from where the exception was thrown to where the backtrace was examined.

Consider a call stack in which A calls B which calls C and C raises an exception. If the backtrace was examined in C, the backtrace would have one unit, C, and the backtrace depth would be one. If it was examined in A, it would have three units, A, B and C, and backtrace depth would be three.

The depth of a backtrace is zero in the absence of an exception.

260.2 UTL_CALL_STACK Security Model

EXECUTE on UTL_CALL_STACK is granted to PUBLIC.

The UTL_CALL_STACK package does not show wrapped program units. For example, consider a call stack in which program unit A calls B, which calls C, and in turn calls UTL_CALL_STACK to determine the subprogram list. If program unit B is wrapped, then the subprogram list only shows program unit C.
260.3 UTL_CALL_STACK Operational Notes

Certain operational notes apply to UTL_CALL_STACK.

- Compiler optimizations can change lexical, dynamic and backtrace depth.
- UTL_CALL_STACK is not supported past RPC boundaries. For example, if A calls remote procedure B, B will not be able to obtain information about A using UTL_CALL_STACK.
- Lexical unit information is available through the PL/SQL conditional compilation feature and is therefore not exposed through UTL_CALL_STACK.

260.4 UTL_CALL_STACK Exceptions

This table lists the exceptions raised by UTL_CALL_STACK.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_DEPTH_INDICATOR</td>
<td>64610</td>
<td>This exception is raised when a provided depth is out of bounds. Dynamic and lexical depth are positive integer values. Error and backtrace depths are non-negative integer values and are zero only in the absence of an exception.</td>
</tr>
</tbody>
</table>

260.5 UTL_CALL_STACK Data Structures

The UTL_CALL_STACK package defines a VARRAY type, UNIT_QUALIFIED_NAME.

VARRAY Type

- UNIT_QUALIFIED_NAME

260.5.1 UNIT_QUALIFIED_NAME

This data structure is a varray whose individual elements are, in order, the unit name, any lexical parents of the subprogram, and the subprogram name.

```
TYPE UNIT_QUALIFIED_NAME IS VARRAY(256) OF VARCHAR2(32767);
```

Example

Consider the following contrived PL/SQL procedure:

```
PROCEDURE topLevel IS
    FUNCTION localFunction(...) RETURNS VARCHAR2 IS
    FUNCTION innerFunction(...) RETURNS VARCHAR2 IS
        BEGIN
            DECLARE
                localVar PLS_INTEGER;
            BEGIN
                ... (1)
            END;
        END;
    END;
```
The unit qualified name at (1) would be

"["topLevel", "localFunction", "innerFunction"]"

If the unit were an anonymous block, the unit name would be "__anonymous_block"

260.6 Summary of UTL_CALL_STACK Subprograms

This table lists the subprograms in the UTL_CALL_STACK package.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKTRACE_DEPTH Function</td>
<td>Returns the number of backtrace items in the backtrace</td>
</tr>
<tr>
<td>BACKTRACE_LINE Function</td>
<td>Returns the line number of the unit at the specified backtrace depth</td>
</tr>
<tr>
<td>BACKTRACE_UNIT Function</td>
<td>Returns the name of the unit at the specified backtrace depth</td>
</tr>
<tr>
<td>CURRENT_EDITION Function</td>
<td>Returns the current edition name of the unit of the subprogram at dynamic depth</td>
</tr>
<tr>
<td>CONCATENATE_SUBPROGRAM Function</td>
<td>Returns a concatenated form of a unit-qualified name</td>
</tr>
<tr>
<td>DYNAMIC_DEPTH Function</td>
<td>Returns the number of subprograms on the call stack</td>
</tr>
<tr>
<td>ERROR_DEPTH Function</td>
<td>Returns the number of errors on the error stack</td>
</tr>
<tr>
<td>ERROR_MSG Function</td>
<td>Returns the error message of the error at the specified error depth</td>
</tr>
<tr>
<td>ERROR_NUMBER Function</td>
<td>Returns the error number of the error at the specified error depth</td>
</tr>
<tr>
<td>LEXICAL_DEPTH Function</td>
<td>Returns the lexical nesting level of the subprogram at specified depth</td>
</tr>
<tr>
<td>OWNER Function</td>
<td>Returns the owner name of the unit of the subprogram at specified depth</td>
</tr>
<tr>
<td>UNIT_LINE Function</td>
<td>Returns the line number of the unit of the subprogram at specified depth</td>
</tr>
<tr>
<td>SUBPROGRAM Function</td>
<td>Returns the unit-qualified name of the subprogram at specified depth</td>
</tr>
</tbody>
</table>

260.6.1 BACKTRACE_DEPTH Function

This function returns the number of backtrace items in the backtrace.

Syntax

```
UTL_CALL_STACK.BACKTRACE_DEPTH
RETURN PLS_INTEGER;
```
Return Values

The number of backtrace items in the backtrace, zero in the absence of an exception.

260.6.2 BACKTRACE_LINE Function

This function returns the line number of the unit at the specified backtrace depth.

Syntax

```sql
UTL_CALL_STACK.BACKTRACE_LINE (    
    backtrace_depth   IN   PLS_INTEGER)    
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtrace_depth</td>
<td>Depth in backtrace</td>
</tr>
</tbody>
</table>

Return Values

The line number of the unit at the specified backtrace depth

260.6.3 BACKTRACE_UNIT Function

This function returns the name of the unit at the specified backtrace depth.

Syntax

```sql
UTL_CALL_STACK.BACKTRACE_UNIT (    
    backtrace_depth   IN   PLS_INTEGER)    
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtrace_depth</td>
<td>Depth in backtrace</td>
</tr>
</tbody>
</table>

Return Values

The name of the unit at the specified backtrace depth
260.6.4 CURRENT_EDITION Function

This function returns the current edition name of the unit of the subprogram at the specified dynamic depth.

Syntax

```
UTL_CALL_STACK.CURRENT_EDITION (
    dynamic_depth    IN    PLS_INTEGER)
RETURN VARCHAR2;
```

Parameters

Table 260-5  CURRENT_EDITION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic_depth</td>
<td>Depth in the error stack</td>
</tr>
</tbody>
</table>

Return Values

The current edition name of the unit of the subprogram at the specified dynamic depth.

260.6.5 CONCATENATE_SUBPROGRAM Function

This function returns a concatenated form of a unit-qualified name.

Syntax

```
UTL_CALL_STACK.CONCATENATE_SUBPROGRAM (
    qualified_name    IN    UNIT_QUALIFIED_NAME)
RETURN VARCHAR2;
```

Parameters

Table 260-6  CONCATENATE_SUBPROGRAM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qualified_name</td>
<td>A unit-qualified name</td>
</tr>
</tbody>
</table>

Return Values

A string of the form `UNIT.SUBPROGRAM.LOCAL_SUBPROGRAM`

260.6.6 DYNAMIC_DEPTH Function

This function returns the number of subprograms on the call stack.

Syntax

```
UTL_CALL_STACK.DYNAMIC_DEPTH
RETURN PLS_INTEGER;
```
Return Values
The number of subprograms on the call stack

260.6.7 ERRORDEPTH Function
This function returns the number of errors on the error stack.

Syntax
UTL_CALL_STACK.ERRORDEPTH
RETURN PLS_INTEGER;

Return Values
The number of errors on the error stack

260.6.8 ERROR_MSG Function
This function returns the error message of the error at the specified error depth.

Syntax
UTL_CALL_STACK.ERROR_MSG (
    error_depth    IN    PLS_INTEGER)
RETURN VARCHAR2;

Parameters

Table 260-7  ERROR_MSG Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_depth</td>
<td>Depth in the error stack</td>
</tr>
</tbody>
</table>

Return Values
The error message of the error at the specified error depth.

260.6.9 ERROR_NUMBER Function
This function returns the error number of the error at the specified error depth.

Syntax
UTL_CALL_STACK.ERROR_NUMBER (
    error_depth    IN    PLS_INTEGER)
RETURN PLS_INTEGER;
### Parameters

**Table 260-8  ERROR_NUMBER Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_depth</td>
<td>Depth in the call stack</td>
</tr>
</tbody>
</table>

**Return Values**

The error number of the error at the specified error depth

#### 260.6.10 LEXICAL_DEPTH Function

This function returns the lexical nesting level of the subprogram at the specified dynamic depth.

**Syntax**

```sql
UTL_CALL_STACK.LEXICAL_DEPTH (  
    dynamic_depth    IN    PLS_INTEGER)  
RETURN PLS_INTEGER;
```

**Parameters**

**Table 260-9  LEXICAL_DEPTH Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic_depth</td>
<td>Depth in the call stack</td>
</tr>
</tbody>
</table>

**Return Values**

The lexical nesting level of the subprogram at the specified dynamic depth

#### 260.6.11 OWNER Function

This function returns the owner name of the unit of the subprogram at the specified dynamic depth.

**Syntax**

```sql
UTL_CALL_STACK.OWNER (  
    dynamic_depth    IN    PLS_INTEGER)  
RETURN VARCHAR2;
```

**Parameters**

**Table 260-10  OWNER Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic_depth</td>
<td>Depth in the call stack</td>
</tr>
</tbody>
</table>
260.6.12 UNIT_LINE Function

This function returns the line number of the unit of the subprogram at the specified dynamic depth.

Syntax

```plsql
UTL_CALL_STACK.UNIT_LINE (dynamic_depth IN PLS_INTEGER) RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic_depth</td>
<td>Depth in the call stack</td>
</tr>
</tbody>
</table>

Return Values

The line number of the unit of the subprogram at the specified dynamic depth

260.6.13 SUBPROGRAM Function

This function returns the unit-qualified name of the subprogram at the specified dynamic depth.

Syntax

```plsql
UTL_CALL_STACK.SUBPROGRAM (dynamic_depth IN PLS_INTEGER) RETURN UNIT_QUALIFIED_NAME;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic_depth</td>
<td>Depth in the call stack</td>
</tr>
</tbody>
</table>

Return Values

Returns the unit-qualified name of the subprogram at the specified dynamic depth.
261

UTL_COLL

The UTL_COLL package lets PL/SQL programs use collection locators to query and update.

This chapter contains the following topics:

• Summary of UTL_COLL Subprograms

261.1 Summary of UTL_COLL Subprograms

The UTL_COLL package has one subprogram, the IS_LOCATOR function.

Table 261-1  UTL_COLL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_LOCATOR</td>
<td>Function</td>
</tr>
<tr>
<td>Determines whether a collection item is actually a locator or not</td>
<td></td>
</tr>
</tbody>
</table>

261.1.1 IS_LOCATOR Function

This function determines whether a collection item is actually a locator or not.

Syntax

```plsql
UTL_COLL.IS_LOCATOR (          coln IN STANDARD)          RETURN Boolean;
```

Pragmas

Asserts WDNS, WNPS and RNPS pragmas

Parameters

Table 261-2  IS_LOCATOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coln</td>
<td>Nested table or varray item.</td>
</tr>
</tbody>
</table>

Return Values

Table 261-3  IS_LOCATOR Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Collection item is indeed a locator.</td>
</tr>
<tr>
<td>0</td>
<td>Collection item is not a locator.</td>
</tr>
</tbody>
</table>
Examples

CREATE OR REPLACE TYPE list_t as TABLE OF VARCHAR2(20);
/

CREATE OR REPLACE TYPE phone_book_t AS OBJECT (  
  pno  number,
  ph   list_t );
/

CREATE TABLE phone_book OF phone_book_t
  NESTED TABLE ph STORE AS nt_ph;
CREATE TABLE phone_book1 OF phone_book_t
  NESTED TABLE ph STORE AS nt_ph_1 RETURN LOCATOR;

INSERT INTO phone_book VALUES(1, list_t('650-633-5707','650-323-0953'));
INSERT INTO phone_book1 VALUES(1, list_t('415-555-1212'));

CREATE OR REPLACE PROCEDURE chk_coll IS
  plist list_t;
  plist1 list_t;
BEGIN
  SELECT ph INTO plist FROM phone_book WHERE pno=1;
  SELECT ph INTO plist1 FROM phone_book1 WHERE pno=1;

  IF (UTL_COLL.IS_LOCATOR(plist)) THEN
    DBMS_OUTPUT.PUT_LINE('plist is a locator');
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist is not a locator');
  END IF;

  IF (UTL_COLL.IS_LOCATOR(plist1)) THEN
    DBMS_OUTPUT.PUT_LINE('plist1 is a locator');
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist1 is not a locator');
  END IF;

END chk_coll;

SET SERVEROUTPUT ON
EXECUTE chk_coll;
The UTL_COMPRESS package provides a set of data compression utilities.

This chapter contains the following topics:

- Constants
- Exceptions
- Operational Notes
- Summary of UTL_COMPRESS Subprograms

### 262.1 UTL_COMPRESS Constants

The maximum number of handles for piecewise operations can be defined by a constant.

```plsql
UTLCOMP_MAX_HANDLE CONSTANT PLS_INTEGER := 5;
```

### 262.2 UTL_COMPRESS Exceptions

This table describes exceptions raised by UTL_COMPRESS subprograms.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER_TOO_SMALL</td>
<td>The compressed representation is too big.</td>
</tr>
<tr>
<td>DATA_ERROR</td>
<td>The input or output data stream was found to be an invalid format.</td>
</tr>
<tr>
<td>INVALID_ARGUMENT</td>
<td>One of the arguments was an invalid type or value.</td>
</tr>
<tr>
<td>INVALID_HANDLE</td>
<td>Invalid handle for piecewise compress or uncompress.</td>
</tr>
<tr>
<td>STREAM_ERROR</td>
<td>An error occurred during compression or uncompression of the data stream</td>
</tr>
</tbody>
</table>

### 262.3 UTL_COMPRESS Operational Notes

Certain operational notes apply to UTL_COMPRESS.

- It is the caller's responsibility to free the temporary LOB returned by the LZ* functions with DBMS_LOB.FREETEMPORARY call.
- A BFILE passed into LZ_COMPRESS* or LZ_UNCOMPRESS* has to be opened by DBMS_LOB.FILEOPEN.
- Under special circumstances (especially if the input has already been compressed) the output produced by one of the UTL_COMPRESS subprograms may be the same size, or even slightly larger than, the input.
262.4 Summary of UTL_COMPRESS Subprograms

This table lists the UTL_COMPRESS subprograms and briefly describes them.

### Table 262-2  UTL_COMPRESS Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOPEN Function</td>
<td>Checks to see if the handle to a piecewise (un)compress context is open or closed</td>
</tr>
<tr>
<td>LZ_COMPRESS Functions and Procedures</td>
<td>Compresses data using Lempel-Ziv compression algorithm</td>
</tr>
<tr>
<td>LZ_COMPRESS_ADD Procedure</td>
<td>Adds a piece of compressed data</td>
</tr>
<tr>
<td>LZ_COMPRESS_CLOSE</td>
<td>Closes and finishes piecewise compress operation</td>
</tr>
<tr>
<td>LZ_COMPRESS_OPEN</td>
<td>Initializes a piecewise context that maintains the compress state and data</td>
</tr>
<tr>
<td>LZ_UNCOMPRESS Functions and Procedures</td>
<td>Accepts compressed input, verifies it to be a valid and uncompresses it</td>
</tr>
<tr>
<td>LZ_UNCOMPRESS_EXTRACT Procedure</td>
<td>Extracts a piece of uncompressed data</td>
</tr>
<tr>
<td>LZ_UNCOMPRESS_OPEN Function</td>
<td>Initializes a piecewise context that maintains the uncompress state and data</td>
</tr>
<tr>
<td>LZ_UNCOMPRESS_CLOSE Procedure</td>
<td>Closes and finishes the piecewise uncompress</td>
</tr>
</tbody>
</table>

#### 262.4.1 ISOPEN Function

This function checks to see if the handle to a piecewise (un)compress context is open or closed.

**Syntax**

```
UTL_COMPRESS.ISOPEN(
    handle in binary_integer)
RETURN BOOLEAN;
```

**Parameters**

### Table 262-3  ISOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle to a piecewise uncompress context.</td>
</tr>
</tbody>
</table>

**Return Values**

TRUE if the given piecewise handle is opened, otherwise FALSE.
Examples

IF (UTL_COMPRESS.ISOPEN(myhandle) = TRUE) then
    UTL_COMPRESS.LZ_COMPRESS_CLOSE(myhandle, lob_1);
END IF;

Alternatively:

IF (UTL_COMPRESS.ISOPEN(myhandle) = TRUE) THEN
    UTL_COMPRESS.LZ_UNCOMPRESS_CLOSE(myhandle);
END IF;

262.4.2 LZ_COMPRESS Functions and Procedures

These functions and procedures compress data using Lempel-Ziv compression algorithm.

Syntax

This function accept a RAW as input, compress it and return the compressed RAW result and metadata:

```
UTL_COMPRESS.LZ_COMPRESS (  
    src  IN           RAW,  
    quality  IN           BINARY_INTEGER DEFAULT 6)  
RETURN RAW;
```

This function accept a BLOB as input, compress it and returns a temporary BLOB for the compressed data:

```
UTL_COMPRESS.LZ_COMPRESS (  
    src  IN           BLOB,  
    quality  IN           BINARY_INTEGER DEFAULT 6)  
RETURN BLOB;
```

This procedure returns the compressed data into the existing BLOB(dst) which is trimmed to the compressed data size:

```
UTL_COMPRESS.LZ_COMPRESS (  
    src  IN            BLOB,  
    dst  IN OUT NOCOPY BLOB,  
    quality  IN           BINARY_INTEGER DEFAULT 6);  
```

This function returns a temporary BLOB for the compressed data:

```
UTL_COMPRESS.LZ_COMPRESS (  
    src  IN            BFILE,  
    quality  IN           BINARY_INTEGER DEFAULT 6)  
RETURN BLOB;
```

This procedure will return the compressed data into the existing BLOB(dst) which is trimmed to the compressed data size:

```
UTL_COMPRESS.LZ_COMPRESS (  
    src  IN            BFILE,  
    dst  IN OUT NOCOPY BLOB,  
    quality  IN           BINARY_INTEGER DEFAULT 6);  
```
Parameters

Table 262-4  LZ_COMPRESS Function and Procedures Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Data (RAW, BLOB or BFILE) to be compressed.</td>
</tr>
<tr>
<td>dst</td>
<td>Destination for compressed data</td>
</tr>
<tr>
<td>quality</td>
<td>An integer in the range 1 to 9, 1=fast compression, 9=best compression, default=6</td>
</tr>
</tbody>
</table>

Usage Notes

- **quality** is an optional compression tuning value. It allows the UTL_COMPRESS user to choose between speed and compression quality, meaning the percentage of reduction in size. A faster compression speed will result in less compression of the data. A slower compression speed will result in more compression of the data. Valid values are [1..9], with 1=fastest and 9=slowest. The default 'quality' value is 6.

262.4.3 LZ_COMPRESS_ADD Procedure

This procedure adds a piece of compressed data.

Syntax

```sql
UTL_COMPRESS.LZ_COMPRESS_ADD (   handle IN             BINARY_INTEGER,   dst    IN OUT NOCOPY  BLOB,   src    IN             RAW);
```

Parameters

Table 262-5  LZ_COMPRESS_ADD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle to a piecewise compress context.</td>
</tr>
<tr>
<td>dst</td>
<td>The opened LOB from LZ_COMPRESS_OPEN to store compressed data.</td>
</tr>
<tr>
<td>src</td>
<td>The input data to be compressed.</td>
</tr>
</tbody>
</table>

Exceptions

- invalid_handle - out of range invalid or unopened handle.
- invalid_argument - NULL handle, src, dst, or invalid dst.
262.4.4 LZ_COMPRESS_CLOSE

This procedure closes and finishes piecewise compress operation.

Syntax

```sql
UTL_COMPRESS.LZ_COMPRESS_CLOSE (    handle IN             BINARY_INTEGER,    dst    IN OUT NOCOPY  BLOB);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle to a piecewise compress context.</td>
</tr>
<tr>
<td>dst</td>
<td>The opened LOB from LZ_COMPRESS_OPEN to store compressed data.</td>
</tr>
</tbody>
</table>

Exceptions

- invalid_handle - out of range invalid or uninitialized handle.
- invalid_argument - NULL handle, dst, or invalid dst.

262.4.5 LZ_COMPRESS_OPEN

This function initializes a piecewise context that maintains the compress state and data.

Syntax

```sql
UTL_COMPRESS.LZ_COMPRESS_OPEN (    dst       IN OUT NOCOPY BLOB,    quality   IN            BINARY_INTEGER DEFAULT 6)    RETURN BINARY_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>User supplied LOB to store compressed data.</td>
</tr>
<tr>
<td>quality</td>
<td>Speed versus efficiency of resulting compressed output.</td>
</tr>
<tr>
<td></td>
<td>Valid values are the range 1..9, with a default value of 6.</td>
</tr>
<tr>
<td></td>
<td>1=fastest compression, 9=slowest compression and best compressed file size.</td>
</tr>
</tbody>
</table>

Return Values

A handle to an initialized piecewise compress context.

Exceptions

- invalid_handle - invalid handle, too many open handles.
• invalid_argument - NULL dst or invalid quality specified.

Usage Notes
Close the opened handle with LZ_COMPRESS_CLOSE
• once the piecewise compress is completed
• in the event of an exception in the middle of process
because lack of doing so will cause these handles to leak.

262.4.6 LZ_UNCOMPRESS Functions and Procedures

This procedure accepts as input a RAW, BLOB or BFILE compressed string, verifies it to be a valid compressed value, uncompresses it using Lempel-Ziv compression algorithm, and returns the uncompressed RAW or BLOB result.

Syntax

This function returns uncompressed data as RAW:

UTL_COMPRESS.LZ_UNCOMPRESS(
    src  IN  RAW)
RETURN RAW;

This function returns uncompressed data as a temporary BLOB:

UTL_COMPRESS.LZ_UNCOMPRESS(
    src  IN  BLOB)
RETURN BLOB;

This procedure returns the uncompressed data into the existing BLOB(dst), which will be trimmed to the uncompressed data size:

UTL_COMPRESS.LZ_UNCOMPRESS(
    src  IN  BLOB,
    dst  IN  OUT NOCOPY BLOB);

This function returns a temporary BLOB for the uncompressed data:

UTL_COMPRESS.LZ_UNCOMPRESS(
    src  IN  BFILE)
RETURN BLOB;

This procedure returns the uncompressed data into the existing BLOB(dst). The original dst data will be overwritten.

UTL_COMPRESS.LZ_UNCOMPRESS(
    src  IN  BFILE,
    dst  IN  OUT NOCOPY BLOB);

Parameters

Table 262-8  LZ_UNCOMPRESS Function and Procedures Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Compressed data.</td>
</tr>
</tbody>
</table>
262.4.7 LZ_UNCOMPRESS_EXTRACT Procedure

This procedure extracts a piece of uncompressed data.

Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_EXTRACT(
    handle  IN  BINARY_INTEGER,
    dst    OUT NOCOPY  RAW);
```

Parameters

Table 262-9  LZ_UNCOMPRESS_EXTRACT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle to a piecewise uncompress context.</td>
</tr>
<tr>
<td>dst</td>
<td>The uncompressed data.</td>
</tr>
</tbody>
</table>

Exceptions

- **no_data_found** - finished uncompress.
- **invalid_handle** - out of range invalid or uninitialized handle.
- **invalid_argument** - NULL handle.

262.4.8 LZ_UNCOMPRESS_OPEN Function

This function initializes a piecewise context that maintains the uncompress state and data.

Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_OPEN(
    src  IN  BLOB)
RETURN BINARY_INTEGER;
```

Parameters

Table 262-10  LZ_UNCOMPRESS_OPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>The input data to be uncompressed.</td>
</tr>
</tbody>
</table>

Return Values

A handle to an initialized piecewise compress context.
Exceptions

- invalid_handle - invalid handle, too many open handles.
- invalid_argument - NULL src.

Usage Notes

Close the opened handle with LZ_UNCOMPRESS_CLOSE

- once the piecewise uncompress is completed
- in the event of an exception in the middle of process
because lack of doing so will cause these handles to leak.

262.4.9 LZ_UNCOMPRESS_CLOSE Procedure

This procedure closes and finishes the piecewise uncompress.

Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_CLOSE(
   handle  IN   BINARY_INTEGER);
```

Parameters

Table 262-11  LZ_UNCOMPRESS_CLOSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>The handle to a piecewise uncompress context.</td>
</tr>
</tbody>
</table>

Exceptions

- invalid_handle - out of range invalid or uninitialized handle.
- invalid_argument - NULL handle.
The `UTL_ENCODE` package provides functions that encode `RAW` data into a standard encoded format so that the data can be transported between hosts.

You can use `UTL_ENCODE` functions to encode the body of email text. The package also contains the decode counterpart functions of the encode functions. The functions follow published standards for encoding to accommodate non-Oracle utilities on the sending or receiving ends.

This chapter contains the following topic:

- Summary of `UTL_ENCODE` Subprograms

### 263.1 Summary of `UTL_ENCODE` Subprograms

This table lists the `UTL_ENCODE` subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BASE64_DECODE Function</code></td>
<td>Reads the base 64-encoded <code>RAW</code> input string and decodes it to its original <code>RAW</code> value</td>
</tr>
<tr>
<td><code>BASE64_ENCODE Function</code></td>
<td>Encodes the binary representation of the <code>RAW</code> value into base 64 elements and returns it in the form of a <code>RAW</code> string</td>
</tr>
<tr>
<td><code>MIMEHEADER_DECODE Function</code></td>
<td>Decodes a string from mime header format</td>
</tr>
<tr>
<td><code>MIMEHEADER_ENCODE Function</code></td>
<td>Encodes a string into mime header format</td>
</tr>
<tr>
<td><code>QUOTED_PRINTABLE_DECODE Function</code></td>
<td>Reads the <code>varchar2</code> quoted printable format input string and decodes it to the corresponding <code>RAW</code> string</td>
</tr>
<tr>
<td><code>QUOTED_PRINTABLE_ENCODE Function</code></td>
<td>Reads the <code>RAW</code> input string and encodes it to the corresponding quoted printable format string</td>
</tr>
<tr>
<td><code>TEXT_DECODE Function</code></td>
<td>Decodes a character set sensitive text string</td>
</tr>
<tr>
<td><code>TEXT_ENCODE Function</code></td>
<td>Encodes a character set sensitive text string</td>
</tr>
<tr>
<td><code>UUDECODE Function</code></td>
<td>Reads the <code>RAW</code> uuencode format input string and decodes it to the corresponding <code>RAW</code> string</td>
</tr>
<tr>
<td><code>UUENCODE Function</code></td>
<td>Reads the <code>RAW</code> input string and encodes it to the corresponding uuencode format string</td>
</tr>
</tbody>
</table>
263.1.1 BASE64_DECODE Function

This function reads the base 64-encoded RAW input string and decodes it to its original RAW value.

Syntax

UTL_ENCODE.BASE64_DECODE (r IN RAW) RETURN RAW;

Pragmas

pragma RESTRICT_REFERENCES(base64_decode, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 263-2   BASE64_DECODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The RAW string containing base 64-encoded data. There are no defaults or optional parameters.</td>
</tr>
</tbody>
</table>

Return Values

Table 263-3   BASE64_DECODE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the decoded string</td>
</tr>
</tbody>
</table>

263.1.2 BASE64_ENCODE Function

This function encodes the binary representation of the RAW value into base 64 elements and returns it in the form of a RAW string.

Syntax

UTL_ENCODE.BASE64_ENCODE (r IN RAW) RETURN RAW;

Pragmas

pragma RESTRICT_REFERENCES(base64_encode, WNDS, RNDS, WNPS, RNPS);
Parameters

Table 263-4  BASE64 ENCODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The RAW value to be encoded. There are no defaults or optional parameters.</td>
</tr>
</tbody>
</table>

Return Values

Table 263-5  BASE64 ENCODE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the encoded base 64 elements</td>
</tr>
</tbody>
</table>

263.1.3 MIMEHEADER DECODE Function

This function accepts as input an "encoded word."

It uses the form:

```sql
=?<charset>?<encoding>?<encoded text>?=
=?ISO-8859-1?Q?Here is some encoded text?= |
```

The `<encoded text>` is encapsulated in mime header tags which give the MIMEHEADER DECODE function information about how to decode the string. The mime header metadata tags are stripped from the input string and the `<encoded text>` is converted to the base database character set as follows:

- If this is a UTF16 platform, convert the encoded text from UTF16 to ASCII
- If this is an EBCDIC platform, convert the encoded text from EBCDIC to ASCII
- If this is an ASCII or UTF8 platform, no conversion needed

The string is decoded using either quoted-printable or base64 decoding, as specified by the `<encoding>` metadata tag in the encoded word. The resulting converted and decoded text is returned to the caller as a VARCHAR2 string.

Syntax

```sql
UTL_ENCODE.MIMEHEADER_DECODE ( |
    buf   IN   VARCHAR2 CHARACTER SET ANY_CS |
) RETURN data VARCHAR2 CHARACTER SET buf%CHARSET; |
```

Parameters

Table 263-6  MIMEHEADER DECODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>The encoded text data with mime header format tags.</td>
</tr>
</tbody>
</table>
Return Values

Table 263-7 MIMEHEADER_DECODE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The encoded text data with mime header format tags</td>
</tr>
</tbody>
</table>

Examples

v2:=utl_encode.mimeheader_decode('=?ISO-8859-1?Q?Here is some encoded text?=');

263.1.4 MIMEHEADER_ENCODE Function

This function returns an output an "encoded word".

The output is in the following form:

=?<charset>?<encoding>?<encoded text>?=
=?ISO-8859-1?Q?Here is some text?=  

The <encoded text> input parameter is the text to be encoded and becomes the <encoded text>.

The <encoding> value is either "Q" or "B" for quoted-printable encode or base64 encoding respectively. The ENCODING input parameter accepts as valid values UTL_ENCODE.QUOTED_PRINTABLE or UTL_ENCODE.BASE64 or NULL. If NULL, quoted-printable encoding is selected as a default value.

The <charset> value is specified as the input parameter encode_charset. If NULL, the database character set is selected as a default value.

The mimeheader encoding process includes conversion of the <encoded text> input string to the character set specified by the encode_charset parameter. The converted string is encoded to either quoted-printable or base64 encoded format. The mime header tags are appended and prepended.

Finally, the string is converted to the base character set of the database:

- If this is a UTF16 platform, convert the encoded text to UTF16
- If this is an EBCDIC platform, convert the encoded text to EBCDIC
- If this is an ASCII or UTF8 platform, no conversion needed.

Syntax

UTL_ENCODE.MIMEHEADER_ENCODE ( 
  buf            IN  VARCHAR2 CHARACTER SET ANY_CS, 
  encode_charset IN  VARCHAR2 DEFAULT NULL, 
  encoding       IN  PLS_INTEGER DEFAULT NULL) 
RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
Parameters

Table 263-8  MIMEHEADER_ENCODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>The text data.</td>
</tr>
<tr>
<td>encode_charset</td>
<td>The target character set.</td>
</tr>
<tr>
<td>encoding</td>
<td>The encoding format. Valid values are UTL_ENCODE.BASE64,</td>
</tr>
<tr>
<td></td>
<td>UTL_ENCODE.QUOTE_PRINTABLE and NULL</td>
</tr>
</tbody>
</table>

Return Values

Table 263-9  MIMEHEADER_ENCODE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A VARCHAR2 encoded string with mime header format tags.</td>
</tr>
</tbody>
</table>

263.1.5 QUOTED_PRINTABLE_DECODE Function

This function reads the varchar2 quoted printable format input string and decodes it to the corresponding RAW string.

Syntax

```sql
UTL_ENCODE.QUOTED_PRINTABLE_DECODE (r  IN RAW)
RETURN RAW;
```

Pragmas

```sql
pragma RESTRICT_REFERENCES(quoted_printable_decode, WNDS, RNDS, WNPS, RNPS); 
```

Parameters

Table 263-10  QUOTED_PRINTABLE_DECODE Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The RAW string containing a quoted printable data string. There are no defaults or optional parameters.</td>
</tr>
</tbody>
</table>

Return Values

Table 263-11  QUOTED_PRINTABLE_DECODE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>The decoded string</td>
</tr>
</tbody>
</table>
263.1.6 QUOTED_PRINTABLE_ENCODE Function

This function reads the RAW input string and encodes it to the corresponding quoted printable format string.

Syntax

```
UTL_ENCODE.QUOTED_PRINTABLE_ENCODE ( 
    r  IN RAW) 
RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(quoted_printable_encode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The RAW string. There are no defaults or optional parameters.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the quoted printable string</td>
</tr>
</tbody>
</table>

263.1.7 TEXT_DECODE Function

This function converts the input text to the target character set as specified by the encode_charset parameter, if not NULL.

The encoded text is converted to the base character set of database, as follows:

- If this is a UTF16 platform, convert the encoded text from UTF16 to ASCII
- If this is an EBCDIC platform, convert the encoded text from EBCDIC to ASCII
- If this is an ASCII or UTF8 platform, no conversion needed

You can decode from either quoted-printable or base64 format, with regard to each encoding parameter. If NULL, quoted-printable is selected as a default decoding format. If encode_charset is not NULL, you convert the string from the specified character set to the database character set. The resulting decoded and converted text string is returned to the caller.

Syntax

```
UTL_ENCODE.TEXT_DECODE( 
    buf            IN  VARCHAR2 CHARACTER SET ANY_CS, 
    encode_charset IN  VARCHAR2 DEFAULT NULL, 
```
263.1.8 TEXT_ENCODE Function

This function converts the input text to the target character set as specified by the encoding_charset parameter, if not NULL.

The text is encoded to either base64 or quoted-printable format, as specified by the encoding parameter. Quoted-printable is selected as a default if ENCODING is NULL.

The encoded text is converted to the base character set of the database:

- If this is a UTF16 platform, convert the encoded text to UTF16
- If this is an EBCDIC platform, convert the encoded text to EBCDIC
- If this is an ASCII or UTF8 platform, no conversion needed

The resulting encoded and converted text string is returned to the caller.

Syntax

```plaintext
UTL_ENCODE.TEXT_ENCODE (  
  buf            IN  VARCHAR2 CHARACTER SET ANY_CS,  
  encode_charset IN  VARCHAR2 DEFAULT NULL,  
  encoding       IN  PLS_INTEGER DEFAULT NULL)  
RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
```
Parameters

Table 263-16  TEXT_ENCODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>The text data.</td>
</tr>
<tr>
<td>encode_charset</td>
<td>The target character set.</td>
</tr>
<tr>
<td>encoding</td>
<td>The encoding format. Valid values are UTL_ENCODE.BASE64,</td>
</tr>
<tr>
<td></td>
<td>UTL_ENCODE.QUOTED_PRINTABLE and NULL</td>
</tr>
</tbody>
</table>

Return Values

Table 263-17  TEXT_ENCODE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A VARCHAR2 encoded string with mime header format tags.</td>
</tr>
</tbody>
</table>

Examples

v2:=utl_encode.text_encode(
    'Here is some text',
    'WE8ISO8859P1',
    UTL_ENCODE.BASE64);

263.1.9 UUDECODE Function

This function reads the RAW uuencode format input string and decodes it to the corresponding RAW string.

See "UUENCODE Function" for discussion of the cumulative nature of UUENCODE and UUDECODE for data streams.

Syntax

UTL_ENCODE.UUDECODE (r IN RAW) RETURN RAW;

Pragmas

pragma RESTRICT_REFERENCES(uudecode, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 263-18  UUDECODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The RAW string containing the uuencoded data string. There are no defaults or optional parameters.</td>
</tr>
</tbody>
</table>
263.1.10 UUENCODE Function

This function reads the RAW input string and encodes it to the corresponding uuencode format string.

The output of this function is cumulative, in that it can be used to encode large data streams, by splitting the data stream into acceptably sized RAW values, encoded, and concatenated into a single encoded string.

Syntax

```
UTL_ENCODE.UUENCODE (r          IN RAW,
    type       IN PLS_INTEGER DEFAULT 1,
    filename   IN VARCHAR2 DEFAULT NULL,
    permission IN VARCHAR2 DEFAULT NULL) RETURN RAW;
```

Pragmas

```
pragma RESTRICT_REFERENCES(uuencode, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 263-20  UUENCODE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW string</td>
</tr>
<tr>
<td>type</td>
<td>Optional number parameter containing the type of uuencoded output. Options: complete—a defined PL/SQL constant with a value of 1. (default) header_piece ...middle_piece ...end_piece</td>
</tr>
<tr>
<td>filename</td>
<td>Optional varchar2 parameter containing the uuencode filename; the default is uuencode.txt</td>
</tr>
<tr>
<td>permission</td>
<td>Optional varchar2 parameter containing the permission mode; the default is 0 (a text string zero).</td>
</tr>
</tbody>
</table>

Return Values

Table 263-21  UUENCODE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Contains the uuencode format string</td>
</tr>
</tbody>
</table>
With the UTL_FILE package, PL/SQL programs can read and write operating system text files. UTL_FILE provides a restricted version of operating system stream file I/O.

This chapter contains the following topics:

- Security Model
- Operational Notes
- Rules and Limits
- Exceptions
- Examples
- Data Structures
- Summary of UTL_FILE Subprograms

264.1 UTL_FILE Security Model

The set of files and directories that are accessible to the user through UTL_FILE is controlled by a number of factors and database parameters. Foremost of these is the set of directory objects that have been granted to the user.

The nature of directory objects is discussed in the Oracle Database SQL Language Reference.

Assuming the user has both READ and WRITE access to the directory object USER_DIR, the user can open a file located in the operating system directory described by USER_DIR, but not in subdirectories or parent directories of this directory.

Lastly, the client (text I/O) and server implementations are subject to operating system file permission checking.

UTL_FILE provides file access both on the client side and on the server side. When run on the server, UTL_FILE provides access to all operating system files that are accessible from the server. On the client side, as in the case for Forms applications, UTL_FILE provides access to operating system files that are accessible from the client.

Directory objects offer more flexibility and granular control to the UTL_FILE application administrator, can be maintained dynamically (that is, without shutting down the database), and are consistent with other Oracle tools. CREATE ANY DIRECTORY privilege is granted only to SYS and SYSTEM by default.

Note:

Use the CREATE DIRECTORY feature for directory access verification.
Note that neither hard nor symbolic links are supported.

On UNIX systems, the owner of a file created by the `FOPEN` function is the owner of the shadow process running the instance. Normally, this owner is `ORACLE`. Files created using `FOPEN` are always writable and readable using the `UTL_FILE` subprograms. However, non-privileged operating system users who need to read these files outside of PL/SQL may need access from a system administrator.

**WARNING:**

The privileges needed to access files in a directory object are operating system specific. `UTL_FILE` directory object privileges give you read and write access to all files within the specified directory.

### 264.2 UTL_FILE Operational Notes

Keep these notes in mind when using `UTL_FILE`.

The file location and file name parameters are supplied to the `FOPEN` function as separate strings, so that the file location can be checked against the list of accessible directories as specified by the `ALL_DIRECTORIES` view of accessible directory objects. Together, the file location and name must represent a legal filename on the system, and the directory must be accessible. A subdirectory of an accessible directory is not necessarily also accessible; it too must be specified using a complete path name matching an `ALL_DIRECTORIES` object.

`UTL_FILE` implicitly interprets line terminators on read requests, thereby affecting the number of bytes returned on a `GET_LINE` call. For example, the `len` parameter of `UTL_FILE.GET_LINE` specifies the requested number of bytes of character data. The number of bytes actually returned to the user will be the lesser of:

- The `GET_LINE len` parameter, or
- The number of bytes until the next line terminator character, or
- The `max_linesize` parameter specified by `UTL_FILE.FOPEN`

The `FOPEN max_linesize` parameter must be a number in the range 1 and 32767. If unspecified, Oracle supplies a default value of 1024. The `GET_LINE len` parameter must be a number in the range 1 and 32767. If unspecified, Oracle supplies the default value of `max_linesize`. If `max_linesize` and `len` are defined to be different values, then the lesser value takes precedence.

`UTL_FILE.GET_RAW` ignores line terminators.

`UTL_FILE` expects that files opened by `UTL_FILE.FOPEN` in text mode are encoded in the database character set. It expects that files opened by `UTL_FILE.FOPEN_NCHAR` in text mode are encoded in the UTF8 character set. If an opened file is not encoded in the expected character set, the result of an attempt to read the file is indeterminate. When data encoded in one character set is read and Globalization Support is told (such as by means of `NLS_LANG`) that it is encoded in another character set, the result is indeterminate. If `NLS_LANG` is set, it should therefore be the same as the database character set.
264.3 UTL_FILE Rules and Limits

Operating system-specific parameters, such as C-shell environment variables under UNIX, cannot be used in the file location or file name parameters.

UTL_FILE I/O capabilities are similar to standard operating system stream file I/O (OPEN, GET, PUT, CLOSE) capabilities, but with some limitations. For example, you call the FOPEN function to return a file handle, which you use in subsequent calls to GET_LINE or PUT to perform stream I/O to a file. When file I/O is done, you call FCLOSE to complete any output and free resources associated with the file.

Note:
The UTL_FILE package is similar to the client-side TEXT_IO package currently provided by Oracle Procedure Builder. Restrictions for a server implementation require some API differences between UTL_FILE and TEXT_IO. In PL/SQL file I/O, errors are returned using PL/SQL exceptions.

264.4 UTL_FILE Exceptions

This table describes exceptions raised by UTL_FILE subprograms.

Table 264-1  UTL_FILE Package Exceptions

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_PATH</td>
<td>File location is invalid.</td>
</tr>
<tr>
<td>INVALID_MODE</td>
<td>The open_mode parameter in FOPEN is invalid.</td>
</tr>
<tr>
<td>INVALID_FILEHANDLE</td>
<td>File handle is invalid.</td>
</tr>
<tr>
<td>INVALID_OPERATION</td>
<td>File could not be opened or operated on as requested.</td>
</tr>
<tr>
<td>READ_ERROR</td>
<td>Destination buffer too small, or operating system error occurred during the read operation</td>
</tr>
<tr>
<td>WRITE_ERROR</td>
<td>Operating system error occurred during the write operation.</td>
</tr>
<tr>
<td>INTERNAL_ERROR</td>
<td>Unspecified PL/SQL error</td>
</tr>
<tr>
<td>CHARSETMISMATCH</td>
<td>A file is opened using FOPEN_NCHAR, but later I/O operations use nonchar functions such as PUTF or GET_LINE.</td>
</tr>
<tr>
<td>FILE_OPEN</td>
<td>The requested operation failed because the file is open.</td>
</tr>
<tr>
<td>INVALID_MAXLINESIZE</td>
<td>The MAX_LINESIZE value for FOPEN() is invalid; it should be within the range 1 to 32767.</td>
</tr>
<tr>
<td>INVALID_FILENAME</td>
<td>The filename parameter is invalid.</td>
</tr>
<tr>
<td>ACCESS_DENIED</td>
<td>Permission to access to the file location is denied.</td>
</tr>
<tr>
<td>INVALID_OFFSET</td>
<td>Causes of the INVALID_OFFSET exception:</td>
</tr>
<tr>
<td></td>
<td>• ABSOLUTE_OFFSET = NULL and RELATIVE_OFFSET = NULL, or</td>
</tr>
<tr>
<td></td>
<td>• ABSOLUTE_OFFSET &lt; 0, or</td>
</tr>
<tr>
<td></td>
<td>• Either offset caused a seek past the end of the file</td>
</tr>
</tbody>
</table>
Table 264-1  (Cont.) UTL_FILE Package Exceptions

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE_FAILED</td>
<td>The requested file delete operation failed.</td>
</tr>
<tr>
<td>RENAME_FAILED</td>
<td>The requested file rename operation failed.</td>
</tr>
</tbody>
</table>

Procedures in UTL_FILE can also raise predefined PL/SQL exceptions such as NO_DATA_FOUND or VALUE_ERROR.

264.5 UTL_FILE Examples

These two examples show use of the procedure.

Example 1

Note: The examples are UNIX-specific.

Given the following:

```
SQL> CREATE DIRECTORY log_dir AS '/appl/gl/log';
SQL> GRANT READ ON DIRECTORY log_dir TO DBA;
SQL> GRANT WRITE ON DIRECTORY log_dir TO DBA;

SQL> CREATE DIRECTORY USER_DIR AS '/appl/gl/user'"
SQL> GRANT READ ON DIRECTORY USER_DIR TO PUBLIC;
SQL> GRANT WRITE ON DIRECTORY USER_DIR TO PUBLIC;
```

The following file locations and filenames are valid and accessible as follows:

<table>
<thead>
<tr>
<th>File Location</th>
<th>Filename</th>
<th>READ and WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>/appl/gl/log</td>
<td>L12345.log</td>
<td>Users with DBA privilege</td>
</tr>
<tr>
<td>/appl/gl/user</td>
<td>u12345.tmp</td>
<td>All users</td>
</tr>
</tbody>
</table>

The following file locations and filenames are invalid:

<table>
<thead>
<tr>
<th>File Location</th>
<th>Filename</th>
<th>Invalid Because</th>
</tr>
</thead>
<tbody>
<tr>
<td>/appl/gl/log/backup</td>
<td>L12345.log</td>
<td># subdirectories are not accessible</td>
</tr>
<tr>
<td>/APPL/gl/log</td>
<td>L12345.log</td>
<td># directory strings must follow case sensitivity rules as required by the O/S</td>
</tr>
<tr>
<td>/appl/gl/log</td>
<td>backup/L1234.log</td>
<td># filenames may not include portions of directory paths</td>
</tr>
</tbody>
</table>
Example 2

DECLARE
    V1 VARCHAR2(32767);
    F1 UTL_FILE.FILE_TYPE;
BEGIN
    -- In this example MAX_LINESIZE is less than GET_LINE's length request
    -- so the number of bytes returned will be 256 or less if a line terminator is seen.
    F1 := UTL_FILE.FOPEN('USER_DIR','u12345.tmp','R',256);
    UTL_FILE.GET_LINE(F1,V1,32767);
    UTL_FILE.FCLOSE(F1);

    -- In this example, FOPEN's MAX_LINESIZE is NULL and defaults to 1024,
    -- so the number of bytes returned will be 1024 or less if a line terminator is seen.
    F1 := UTL_FILE.FOPEN('USER_DIR','u12345.tmp','R');
    UTL_FILE.GET_LINE(F1,V1,32767);
    UTL_FILE.FCLOSE(F1);

    -- In this example, GET_LINE doesn't specify a number of bytes, so it defaults to
    -- the same value as FOPEN's MAX_LINESIZE which is NULL in this case and defaults to 1024.
    -- So the number of bytes returned will be 1024 or less if a line terminator is seen.
    F1 := UTL_FILE.FOPEN('USER_DIR','u12345.tmp','R');
    UTL_FILE.GET_LINE(F1,V1);
    UTL_FILE.FCLOSE(F1);
END;

264.6 UTL_FILE Data Structures

The UTL_FILE package defines a RECORD type.

Record Types

• FILETYPE Record Type

264.6.1 FILETYPE Record Type

The contents of FILE_TYPE are private to the UTL_FILE package. You should not reference or change components of this record.

TYPE file_type IS RECORD (  
id          BINARY_INTEGER,
    datatype   BINARY_INTEGER,
    byte_mode  BOOLEAN);
Fields

Table 264-2   FILE_TYPE Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>A numeric value indicating the internal file handle number</td>
</tr>
<tr>
<td>datatype</td>
<td>Indicates whether the file is a CHAR file, Nchar file or other (binary)</td>
</tr>
<tr>
<td>byte_mode</td>
<td>Indicates whether the file was open as a binary file, or as a text file</td>
</tr>
</tbody>
</table>

⚠️ Caution:

Oracle does not guarantee the persistence of FILE_TYPE values between database sessions or within a single session. Attempts to clone file handles or use dummy file handles may have indeterminate outcomes.

264.7 Summary of UTL_FILE Subprograms

This table lists the UTL_FILE subprograms and briefly describes them.

Table 264-3   UTL_FILE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCLOSE Procedure</td>
<td>Closes a file</td>
</tr>
<tr>
<td>FCLOSE_ALL Procedure</td>
<td>Closes all open file handles</td>
</tr>
<tr>
<td>FCOPY Procedure</td>
<td>Copies a contiguous portion of a file to a newly created file</td>
</tr>
<tr>
<td>FFLUSH Procedure</td>
<td>Physically writes all pending output to a file</td>
</tr>
<tr>
<td>FGETATTR Procedure</td>
<td>Reads and returns the attributes of a disk file</td>
</tr>
<tr>
<td>FGETPOS Function</td>
<td>Returns the current relative offset position within a file, in bytes</td>
</tr>
<tr>
<td>FOPEN Function</td>
<td>Opens a file for input or output</td>
</tr>
<tr>
<td>FOPEN_NCHAR Function</td>
<td>Opens a file in Unicode for input or output</td>
</tr>
<tr>
<td>FREMOVE Procedure</td>
<td>Deletes a disk file, assuming that you have sufficient privileges</td>
</tr>
<tr>
<td>FRENENAME Procedure</td>
<td>Renames an existing file to a new name, similar to the UNIX mv function</td>
</tr>
<tr>
<td>FSEEK Procedure</td>
<td>Adjusts the file pointer forward or backward within the file by the number of bytes specified</td>
</tr>
<tr>
<td>GET_LINE Procedure</td>
<td>Reads text from an open file</td>
</tr>
<tr>
<td>GET_LINE_NCHAR Procedure</td>
<td>Reads text in Unicode from an open file</td>
</tr>
<tr>
<td>GET_RAW Procedure</td>
<td>Reads a RAW string value from a file and adjusts the file pointer ahead by the number of bytes read</td>
</tr>
<tr>
<td>IS_OPEN Function</td>
<td>Determines if a file handle refers to an open file</td>
</tr>
</tbody>
</table>
Table 264-3  (Cont.) UTL_FILE Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_LINE Procedure</td>
<td>Writes one or more operating system-specific line terminators to a file</td>
</tr>
<tr>
<td>PUT Procedure</td>
<td>Writes a string to a file</td>
</tr>
<tr>
<td>PUT_LINE Procedure</td>
<td>Writes a line to a file, and so appends an operating system-specific line terminator</td>
</tr>
<tr>
<td>PUT_LINE_NCHAR Procedure</td>
<td>Writes a Unicode line to a file</td>
</tr>
<tr>
<td>PUT_NCHAR Procedure</td>
<td>Writes a Unicode string to a file</td>
</tr>
<tr>
<td>PUTF Procedure</td>
<td>A PUT procedure with formatting</td>
</tr>
<tr>
<td>PUTF_NCHAR Procedure</td>
<td>A PUTF_NCHAR procedure with formatting, and writes a Unicode string to a file, with formatting</td>
</tr>
<tr>
<td>PUT_RAW Procedure</td>
<td>Accepts as input a RAW data value and writes the value to the output buffer</td>
</tr>
</tbody>
</table>

264.7.1 FCLOSE Procedure

This procedure closes an open file identified by a file handle.

Syntax

```
UTL_FILE.FCLOSE (  
   file IN OUT FILE_TYPE);
```

Parameters

Table 264-4  FCLOSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN or FOPEN_NCHAR call</td>
</tr>
</tbody>
</table>

Usage Notes

If there is buffered data yet to be written when FCLOSE runs, then you may receive a WRITE_ERROR exception when closing a file.

Exceptions

WRITE_ERROR
INVALID_FILEHANDLE
264.7.2 FCLOSE_ALL Procedure

This procedure closes all open file handles for the session. This should be used as an emergency cleanup procedure, for example, when a PL/SQL program exits on an exception.

Syntax

UTL_FILE.FCLOSE_ALL;

Usage Notes

Note:

FCLOSE_ALL does not alter the state of the open file handles held by the user. This means that an IS_OPEN test on a file handle after an FCLOSE_ALL call still returns TRUE, even though the file has been closed. No further read or write operations can be performed on a file that was open before an FCLOSE_ALL.

Exceptions

WRITE_ERROR

264.7.3 FCOPY Procedure

This procedure copies a contiguous portion of a file to a newly created file.

By default, the whole file is copied if the start_line and end_line parameters are omitted. The source file is opened in read mode. The destination file is opened in write mode. A starting and ending line number can optionally be specified to select a portion from the center of the source file for copying.

Syntax

UTL_FILE.FCOPY (
    src_location    IN VARCHAR2,
    src_filename    IN VARCHAR2,
    dest_location   IN VARCHAR2,
    dest_filename   IN VARCHAR2,
    start_line      IN BINARY_INTEGER DEFAULT 1,
    end_line        IN BINARY_INTEGER DEFAULT NULL);

Parameters

Table 264-5  FCOPY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_location</td>
<td>Directory location of the source file, a DIRECTORY_NAME from the ALL.Directories view (case sensitive)</td>
</tr>
<tr>
<td>src_filename</td>
<td>Source file to be copied</td>
</tr>
</tbody>
</table>
### Table 264-5  (Cont.) FCOPY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest_location</td>
<td>Destination directory where the destination file is created</td>
</tr>
<tr>
<td>dest_filename</td>
<td>Destination file created from the source file</td>
</tr>
<tr>
<td>start_line</td>
<td>Line number at which to begin copying. The default is 1 for the first line</td>
</tr>
<tr>
<td>end_line</td>
<td>Line number at which to stop copying. The default is NULL, signifying end of file</td>
</tr>
</tbody>
</table>

#### Exceptions

- INVALID_FILENAME
- INVALID_PATH
- INVALID_OPERATION
- INVALID_OFFSET
- READ_ERROR
- WRITE_ERROR

### 264.7.4 FFLUSH Procedure

**FFLUSH** physically writes pending data to the file identified by the file handle. Normally, data being written to a file is buffered. The **FFLUSH** procedure forces the buffered data to be written to the file. The data must be terminated with a newline character.

Flushing is useful when the file must be read while still open. For example, debugging messages can be flushed to the file so that they can be read immediately.

#### Syntax

```sql
UTL_FILE.FFLUSH (file IN FILE_TYPE);
```

#### Parameters

**Table 264-6  FFLUSH Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an OPEN or OPEN_NCHAR call</td>
</tr>
</tbody>
</table>

#### Exceptions

- INVALID_FILENAME
- INVALID_MAXLINESIZE
- INVALID_OPERATION
- WRITE_ERROR
264.7.5 FGETATTR Procedure

This procedure reads and returns the attributes of a disk file.

Syntax

```
UTL_FILE.FGETATTR(
    location     IN VARCHAR2,
    filename     IN VARCHAR2,
    fexists      OUT BOOLEAN,
    file_length  OUT NUMBER,
    block_size   OUT BINARY_INTEGER);
```

Parameters

Table 264-7  FGETATTR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Directory location of the source file, a DIRECTORY_NAME from the ALL_DIRECTORIES view (case sensitive)</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the file to be examined</td>
</tr>
<tr>
<td>fexists</td>
<td>A BOOLEAN for whether or not the file exists</td>
</tr>
<tr>
<td>file_length</td>
<td>Length of the file in bytes. NULL if file does not exist.</td>
</tr>
<tr>
<td>block_size</td>
<td>File system block size in bytes. NULL if the file does not exist.</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_PATH
INVALID_FILENAME
INVALID_OPERATION
READ_ERROR
ACCESS_DENIED

264.7.6 FGETPOS Function

This function returns the current relative offset position within a file, in bytes.

Syntax

```
UTL_FILE.FGETPOS ( file IN FILE_TYPE) RETURN PLS_INTEGER;
```
Parameters

Table 264-8  FGETPOS Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Directory location of the source file</td>
</tr>
</tbody>
</table>

Return Values

FGETPOS returns the relative offset position for an open file, in bytes. It raises an exception if the file is not open. It returns 0 for the beginning of the file.

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
READ_ERROR

Usage Notes

If file is opened for byte mode operations, then the INVALID_OPERATION exception is raised.

264.7.7 FOPEN Function

This function opens a file. You can specify the maximum line size and have a maximum of 50 files open simultaneously.

See also FOPEN_NCHAR Function.

Syntax

UTL_FILE.FOPEN (  
    location     IN VARCHAR2,  
    filename     IN VARCHAR2,  
    open_mode    IN VARCHAR2,  
    max_linesize IN BINARY_INTEGER DEFAULT 1024)  
RETURN FILE_TYPE;

Parameters

Table 264-9  FOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Directory location of file. This string is a directory object name and must be specified in upper case. Read privileges must be granted on this directory object for the UTL_FILE user to run FOPEN.</td>
</tr>
</tbody>
</table>
| filename  | File name, including extension (file type), without directory path. If a directory path is given as a part of the filename, it is ignored by FOPEN. On Unix, the filename cannot end with /.

Table 264-9  (Cont.) FOPEN Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_mode</td>
<td>Specifies how the file is opened. Modes include:</td>
</tr>
<tr>
<td></td>
<td>· r -- read text</td>
</tr>
<tr>
<td></td>
<td>· w -- write text</td>
</tr>
<tr>
<td></td>
<td>· a -- append text</td>
</tr>
<tr>
<td></td>
<td>· rb -- read byte mode</td>
</tr>
<tr>
<td></td>
<td>· wb -- write byte mode</td>
</tr>
<tr>
<td></td>
<td>· ab -- append byte mode</td>
</tr>
<tr>
<td>max_linesize</td>
<td>Maximum number of bytes for each line, including the newline character, for this file (minimum value 1, maximum value 32767). If unspecified, Oracle supplies a default value of 1024.</td>
</tr>
</tbody>
</table>

Return Values

FOPEN returns a file handle, which must be passed to all subsequent procedures that operate on that file. The specific contents of the file handle are private to the UTL_FILE package, and individual components should not be referenced or changed by the UTL_FILE user.

Table 264-10  FOPEN Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_TYPE</td>
<td>Handle to open file</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_MAXILINEIZE
INVALID_MODE
INVALID_OPERATION
INVALID_PATH
INVALID_FILENAME

Usage Notes

The file location and file name parameters must be supplied to the FOPEN function as quoted strings so that the file location can be checked against the list of accessible directories as specified by the ALL_DIRECTORIES view of accessible directory objects.
264.7.8 FOPEN_NCHAR Function

This function opens a file in national character set mode for input or output, with the maximum line size specified. With this function, you can read or write a text file in Unicode instead of in the database character set.

You can have a maximum of 50 files open simultaneously.

Even though the contents of an NVARCHAR2 buffer may be AL16UTF16 or UTF8 (depending on the national character set of the database), the contents of the file are always read and written in UTF8. UTL_FILE converts between UTF8 and AL16UTF16 as necessary.

See also FOPEN Function.

Syntax

```
UTL_FILE.FOPEN_NCHAR (  
    location     IN VARCHAR2,  
    filename     IN VARCHAR2,  
    open_mode    IN VARCHAR2,  
    max_linesize IN BINARY_INTEGER DEFAULT 1024)  
RETURN FILE_TYPE;
```

Parameters

**Table 264-11  FOPEN_NCHAR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Directory location of file</td>
</tr>
<tr>
<td>filename</td>
<td>File name (including extension)</td>
</tr>
<tr>
<td>open_mode</td>
<td>Open mode (r,w,a,rb,wb,ab)</td>
</tr>
<tr>
<td>max_linesize</td>
<td>Maximum number of characters for each line, including the newline character, for this file (minimum value 1, maximum value 32767)</td>
</tr>
</tbody>
</table>

Return Values

FOPEN_NCHAR returns a file handle, which must be passed to all subsequent procedures that operate on that file. The specific contents of the file handle are private to the UTL_FILE package, and individual components should not be referenced or changed by the UTL_FILE user.

**Table 264-12  FOPEN_NCHAR Function Return Values**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_TYPE</td>
<td>Handle to open file</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_MAXILINESIZE

INVALID_MODE

INVALID_OPERATION
264.7.9 FREMOVE Procedure

This procedure deletes a disk file, assuming that you have sufficient privileges.

Syntax

```sql
UTL_FILE.FREMOVE (
    location IN VARCHAR2,
    filename IN VARCHAR2);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Directory location of the file, a DIRECTORY_NAME from ALL_DIRECTORIES (case sensitive)</td>
</tr>
<tr>
<td>filename</td>
<td>Name of the file to be deleted</td>
</tr>
</tbody>
</table>

Exceptions

ACCESS_DENIED
DELETE_FAILED
INVALID_FILENAME
INVALID_OPERATION
INVALID_PATH

Usage Notes

The FREMOVE procedure does not verify privileges before deleting a file. The O/S verifies file and directory permissions. An exception is returned on failure.

264.7.10 FRENAMe Procedure

This procedure renames an existing file to a new name, similar to the UNIX `mv` function.

Syntax

```sql
UTL_FILE.FRENAME (
    src_location     IN   VARCHAR2,
    src_filename     IN   VARCHAR2,
    dest_location    IN   VARCHAR2,
    dest_filename    IN   VARCHAR2,
    overwrite        IN   BOOLEAN DEFAULT FALSE);
```
### Parameters

#### Table 264-14  FRENAMIE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_location</td>
<td>Directory location of the source file, a DIRECTORY_NAME from the ALL_DIRECTORIES view (case sensitive)</td>
</tr>
<tr>
<td>src_filename</td>
<td>Source file to be renamed</td>
</tr>
<tr>
<td>dest_location</td>
<td>Destination directory of the destination file, a DIRECTORY_NAME from the ALL_DIRECTORIES view (case sensitive)</td>
</tr>
<tr>
<td>dest_filename</td>
<td>New name of the file</td>
</tr>
<tr>
<td>overwrite</td>
<td>Default is FALSE. Permission on both the source and destination directories must be granted. You can use the overwrite parameter to specify whether or not to overwrite a file if one exists in the destination directory. The default is FALSE for no overwrite.</td>
</tr>
</tbody>
</table>

### Exceptions

ACCESS_DENIED
INVALID_FILENAME
INVALID_PATH
RENAME_FAILED

### 264.7.11 FSEEK Procedure

This procedure adjusts the file pointer forward or backward within the file by the number of bytes specified.

#### Syntax

```sql
UTL_FILE.FSEEK (file             IN OUT  UTL_FILE.FILE_TYPE,
                absolute_offset  IN      PL_INTEGER DEFAULT NULL,
                relative_offset  IN      PLS_INTEGER DEFAULT NULL);
```

#### Parameters

#### Table 264-15  FSEEK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File handle</td>
</tr>
<tr>
<td>absolute_offset</td>
<td>Absolute location to which to seek; default = NULL</td>
</tr>
<tr>
<td>relative_offset</td>
<td>Number of bytes to seek forward or backward; positive = forward, negative integer = backward, zero = current position, default = NULL</td>
</tr>
</tbody>
</table>

### Exceptions

INVALID_FILEHANDLE
Usage Notes

- Using FSEEK, you can read previous lines in the file without first closing and reopening the file. You must know the number of bytes by which you want to navigate.

- If relative_offset, the procedure seeks forward. If relative_offset > 0, or backward, if relative_offset < 0, the procedure seeks through the file by the number of relative_offset bytes specified.

- If the beginning of the file is reached before the number of bytes specified, then the file pointer is placed at the beginning of the file. If the end of the file is reached before the number of bytes specified, then an INVALID_OFFSET error is raised.

- If absolute_offset, the procedure seeks to an absolute location specified in bytes.

- If file is opened for byte mode operations, then the INVALID_OPERATION exception is raised.

264.7.12 GET.getLine Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. Text is read up to, but not including, the line terminator, or up to the end of the file, or up to the end of the len parameter. It cannot exceed the max_linesize specified in FOPEN.

Syntax

UTL_FILE.GET_LINE ( 
    file        IN  FILE_TYPE, 
    buffer      OUT VARCHAR2, 
    len         IN  PLS_INTEGER DEFAULT NULL); 

Parameters

Table 264-16  GET_LINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN call. The file must be open for reading (mode r); otherwise an INVALID_OPERATION exception is raised.</td>
</tr>
<tr>
<td>buffer</td>
<td>Data buffer to receive the line read from the file</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes read from the file. Default is NULL. If NULL, Oracle supplies the value of max_linesize.</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_FILEHANDLE
Usage Notes

If the line does not fit in the buffer, a READ_ERROR exception is raised. If no text was read due to end of file, the NO_DATA_FOUND exception is raised. If the file is opened for byte mode operations, the INVALID_OPERATION exception is raised.

Because the line terminator character is not read into the buffer, reading blank lines returns empty strings.

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. See also "GET_LINE_NCHAR Procedure".

264.7.13 GET_LINE_NCHAR Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. With this function, you can read a text file in Unicode instead of in the database character set.

The file must be opened in national character set mode, and must be encoded in the UTF8 character set. The expected buffer datatype is NVARCHAR2. If a variable of another datatype, such as NCHAR, NCLB, or VARCHAR2 is specified, PL/SQL will perform standard implicit conversion from NVARCHAR2 after the text is read.

See also GET_LINE Procedure

Syntax

UTL_FILE.GET_LINE_NCHAR (  
    file    IN  FILE_TYPE,  
    buffer  OUT NVARCHAR2,  
    len     IN  PLS_INTEGER DEFAULT NULL);  

Parameters

Table 264-17    GET_LINE_NCHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.</td>
</tr>
<tr>
<td>buffer</td>
<td>Data buffer to receive the line read from the file</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes read from the file. Default is NULL. If NULL, Oracle supplies the value of max_linesize.</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_FILEHANDLE

INVALID_OPERATION
264.7.14 GET_RAW Procedure

This procedure reads a \texttt{RAW} string value from a file and adjusts the file pointer ahead by the number of bytes read. \texttt{UTL\_FILE\_GET\_RAW} ignores line terminators.

**Syntax**

\begin{verbatim}
UTL\_FILE\_GET\_RAW (  
   file       IN            UTL\_FILE\_FILE\_TYPE,  
   buffer     OUT NOCOPY    RAW,  
   len        IN            PLS\_INTEGER DEFAULT NULL);  
\end{verbatim}

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File handle</td>
</tr>
<tr>
<td>buffer</td>
<td>RAW data</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes read from the file. Default is NULL. If NULL, len is assumed to be the maximum length of RAW.</td>
</tr>
</tbody>
</table>

**Exceptions**

- INVALID\_FILE\_HANDLE
- INVALID\_OPERATION
- LENGTH\_MISMATCH
- NO\_DATA\_FOUND
- READ\_ERROR

**Usage Notes**

The subprogram will raise \texttt{No\_Data\_Found} when it attempts to read past the end of the file. Your application should allow for this by catching the exception in its processing loop.

```sql
PROCEDURE Sys.p (n IN VARCHAR2) IS  
h     UTL\_FILE\_FILE\_TYPE := UTL\_FILE\_FOPEN(‘D’, n, ‘r’, 32767);  
Buf   RAW(32767);  
Amnt  CONSTANT PLS\_INTEGER := 32767;  
BEGIN  
LOOP  
BEGIN  
   Utl\_File\_Get\_Raw(h, Buf, Amnt);  
   -- Do something with this chunk  
   EXCEPTION WHEN No\_Data\_Found THEN EXIT; END;  
END LOOP;  
UTL\_FILE\_FCLOSE (h);  
END;
```
264.7.15 IS_OPEN Function

This function tests a file handle to see if it identifies an open file.

**IS_OPEN** reports only whether a file handle represents a file that has been opened, but not yet closed. It does not guarantee that there will be no operating system errors when you attempt to use the file handle.

**Syntax**

```
UTL_FILE.IS_OPEN (  
    file  IN FILE_TYPE)  
RETURN BOOLEAN;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an OPEN or OPEN_NCHAR call</td>
</tr>
</tbody>
</table>

**Return Values**

TRUE or FALSE

**Exceptions**

INVALID_FILEHANDLE

264.7.16 NEW_LINE Procedure

This procedure writes one or more line terminators to the file identified by the input file handle.

This procedure is separate from **PUT** because the line terminator is a platform-specific character or sequence of characters.

**Syntax**

```
UTL_FILE.NEW_LINE (  
    file     IN FILE_TYPE,  
    lines    IN BINARY_INTEGER := 1);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an OPEN or OPEN_NCHAR call</td>
</tr>
<tr>
<td>lines</td>
<td>Number of line terminators to be written to the file</td>
</tr>
</tbody>
</table>
Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

264.7.17 PUT Procedure

PUT writes the text string stored in the buffer parameter to the open file identified by the file handle.

The file must be open for write operations. No line terminator is appended by PUT; use NEW_LINE to terminate the line or use PUT_LINE to write a complete line with a line terminator. See also "PUT_NCHAR Procedure".

Syntax

UTL_FILE.PUT (  
    file      IN FILE_TYPE,  
    buffer    IN VARCHAR2);  

Parameters

Table 264-21  PUT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN_NCHAR call. The file must be open for writing.</td>
</tr>
<tr>
<td>buffer</td>
<td>Buffer that contains the text to be written to the file. User must have opened the file using mode w or mode a; otherwise, an INVALID_OPERATION exception is raised.</td>
</tr>
</tbody>
</table>

Usage Notes

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

264.7.18 PUT_LINE Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle.

The file must be open for write operations. PUT_LINE terminates the line with the platform-specific line terminator character or characters.
See also "PUT_LINE_NCHAR Procedure".

Syntax

```
UTL_FILE.PUT_LINE (
    file    IN FILE_TYPE,
    buffer  IN VARCHAR2,
    autoflush IN BOOLEAN DEFAULT FALSE);
```

Parameters

**Table 264-22 PUT_LINE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an OPEN call</td>
</tr>
<tr>
<td>buffer</td>
<td>Text buffer that contains the lines to be written to the file</td>
</tr>
<tr>
<td>autoflush</td>
<td>Flushes the buffer to disk after the WRITE</td>
</tr>
</tbody>
</table>

Exceptions

- INVALID_FILEHANDLE
- INVALID_OPERATION
- WRITE_ERROR

Usage Notes

- The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in OPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.
- If file is opened for byte mode operations, then the INVALID_OPERATION exception is raised.

264.7.19 PUT_LINE_NCHAR Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. With this function, you can write a text file in Unicode instead of in the database character set.

This procedure is equivalent to the PUT_NCHAR Procedure, except that the line separator is appended to the written text. See also PUT_LINE Procedure.

Syntax

```
UTL_FILE.PUT_LINE_NCHAR (
    file    IN FILE_TYPE,
    buffer  IN NVARCHAR2);
```
Parameters

Table 264-23  PUT_LINE_NCHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN_NCHAR call. The file must be open for writing.</td>
</tr>
<tr>
<td>buffer</td>
<td>Text buffer that contains the lines to be written to the file</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

Usage Notes

- The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.
- If file is opened for byte mode operations, then the INVALID_OPERATION exception is raised.

264.7.20 PUT_NCHAR Procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle.

With this function, you can write a text file in Unicode instead of in the database character set. The file must be opened in the national character set mode. The text string will be written in the UTF8 character set. The expected buffer datatype is NVARCHAR2. If a variable of another datatype is specified, PL/SQL will perform implicit conversion to NVARCHAR2 before writing the text.

Syntax

UTL_FILE.PUT_NCHAR (  
      file      IN FILE_TYPE,  
      buffer    IN NVARCHAR2);  

Parameters

Table 264-24  PUT_NCHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN_NCHAR call. If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.</td>
</tr>
</tbody>
</table>
Table 264-24  (Cont.) PUT_NCHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>Buffer that contains the text to be written to the file. User must have opened the file using mode <code>w</code> or mode <code>a</code>; otherwise, an INVALID_OPERATION exception is raised.</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_FILEHANDLE

INVALID_OPERATION

WRITE_ERROR

Usage Notes

The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

Related Topics

- PUT Procedure
  - PUT writes the text string stored in the buffer parameter to the open file identified by the file handle.

264.7.21 PUTF Procedure

This procedure is a formatted PUT procedure.

It works like a limited printf().

Syntax

```sql
UTL_FILE.PUTF (  
    file     IN FILE_TYPE,  
    format   IN VARCHAR2,  
    [arg1    IN VARCHAR2 DEFAULT NULL,  
    . . .  
    arg5     IN VARCHAR2 DEFAULT NULL]);
```

Parameters

Table 264-25  PUTF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an OPEN call</td>
</tr>
<tr>
<td>format</td>
<td>Format string that can contain text as well as the formatting characters \n and %s</td>
</tr>
</tbody>
</table>
Table 264-25  (Cont.) PUTF Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1..arg5</td>
<td>From one to five operational argument strings. Argument strings are substituted, in order, for the %s formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each %s for which there is no argument.</td>
</tr>
</tbody>
</table>

Usage Notes

- If file is opened for byte mode operations, then the INVALID_OPERATION exception is raised.
- The format string can contain any text, but the character sequences %s and \n have special meaning.

<table>
<thead>
<tr>
<th>Character Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%s</td>
<td>Substitute this sequence with the string value of the next argument in the argument list.</td>
</tr>
<tr>
<td>\n</td>
<td>Substitute with the appropriate platform-specific line terminator.</td>
</tr>
</tbody>
</table>

Exceptions

- INVALID_FILEHANDLE
- INVALID_OPERATION
- WRITE_ERROR

Examples

The following example writes the lines:

Hello, world!
I come from Zork with greetings for all earthlings.

my_world  varchar2(4) := 'Zork';
...
PUTF(my_handle, 'Hello, world!\nI come from %s with %s.\n', my_world, 'greetings for all earthlings');

If there are more %s formatters in the format parameter than there are arguments, then an empty string is substituted for each %s for which there is no matching argument.

Related Topics

- PUTF_NCHAR Procedure
  This procedure is a formatted version of a PUT_NCHAR Procedure.
264.7.22 PUTF_NCHAR Procedure

This procedure is a formatted version of a PUT_NCHAR Procedure.

Using PUTF_NCHAR, you can write a text file in Unicode instead of in the database character set. It accepts a format string with formatting elements \n and %s, and up to five arguments to be substituted for consecutive instances of %s in the format string. The expected datatype of the format string and the arguments is NVARCHAR2.

If variables of another datatype are specified, PL/SQL will perform implicit conversion to NVARCHAR2 before formatting the text. Formatted text is written in the UTF8 character set to the file identified by the file handle. The file must be opened in the national character set mode.

Syntax

UTL_FILE.PUTF_NCHAR (  
    file IN FILE_TYPE,  
    format IN NVARCHAR2,  
    [arg1 IN NVARCHAR2 DEFAULT NULL,  
    ...  
    arg5 IN NVARCHAR2 DEFAULT NULL]);

Parameters

Table 264-26  PUTF_NCHAR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.</td>
</tr>
<tr>
<td>format</td>
<td>Format string that can contain text as well as the formatting characters \n and %s</td>
</tr>
<tr>
<td>arg1..arg5</td>
<td>From one to five operational argument strings. Argument strings are substituted, in order, for the %s formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each %s for which there is no argument.</td>
</tr>
</tbody>
</table>

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

Usage Notes

- The maximum size of the buffer parameter is 32767 bytes unless you specify a smaller size in FOPEN. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.
• If file is opened for byte mode operations, then the `INVALID_OPERATION` exception is raised.

Related Topics
• `PUT_NCHAR Procedure`
  This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle.

264.7.23 `PUT_RAW Procedure`

This procedure accepts as input a `RAW` data value and writes the value to the output buffer.

**Syntax**

```plaintext
UTL_FILE.PUT_RAW (  
  file IN UTL_FILE.FILE_TYPE,  
  buffer IN RAW,  
  autoflush IN BOOLEAN DEFAULT FALSE);  
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File handle</td>
</tr>
<tr>
<td>buffer</td>
<td>The <code>RAW</code> data written to the buffer</td>
</tr>
<tr>
<td>autoflush</td>
<td>If <code>TRUE</code>, then performs a flush after writing the value to the output buffer; default is <code>FALSE</code>.</td>
</tr>
</tbody>
</table>

**Exceptions**

- `INVALID_FILEHANDLE`
- `INVALID_OPERATION`
- `WRITE_ERROR`

**Usage Notes**

You can request an automatic flush of the buffer by setting the third argument to `TRUE`.

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.
The **UTL_HTTP** package makes Hypertext Transfer Protocol (HTTP) callouts from SQL and PL/SQL. You can use it to access data on the Internet over HTTP.

When the package fetches data from a Web site using HTTPS, it requires Oracle Wallet Manager which can be created by either Oracle Wallet Manager or the orapki utility. Non-HTTPS fetches do not require an Oracle wallet.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Constants**
- **Exceptions**
- **Examples**
- **Data Structures**
- **Operations**
- **Subprogram Groups**
- **Summary of UTL_HTTP Subprograms**

![See Also:

- UTL_URL
- UTL_SMTP
- Oracle Database Enterprise User Security Administrator's Guide for more information on Wallet Manager](https://www.oracle.com)

### 265.1 UTL_HTTP Overview

With the **UTL_HTTP** package, you can write PL/SQL programs that communicate with Web (HTTP) servers. **UTL_HTTP** also contains a function that can be used in SQL queries.

The package supports HTTP over the Secured Socket Layer protocol (SSL), also known as HTTPS. It also supports SSL client authentication by sending the client-certificate in a wallet to authenticate with the remote Web server.

Other Internet-related data-access protocols (such as the File Transfer Protocol (FTP) or the Gopher protocol) are also supported using an HTTP proxy server that supports those protocols.
265.2 UTL_HTTP Security Model

This package is an invoker's rights package. The invoking user will need the `connect` privilege granted in the access control list assigned to the remote network host to which he wants to connect, as well as the `use-client-certificates` or the `use-passwords` privilege to authenticate himself with the remote Web server using the credentials stored in an Oracle wallet.

Note:

For more information about managing fine-grained access, see *Oracle Database Security Guide*.

265.3 UTL_HTTP Constants

The `UTL_HTTP` package defines several constants to use when specifying parameter values.

These are shown in following tables.

- Table 265-1
- Table 265-2
- Table 265-3

<table>
<thead>
<tr>
<th>Table 265-1</th>
<th>UTL_HTTP Constants - HTTP Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type</td>
</tr>
<tr>
<td>HTTP_VERSION_1_0</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>HTTP_VERSION_1_1</td>
<td>VARCHAR2(10)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 265-2</th>
<th>UTL_HTTP Constants - Default Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type</td>
</tr>
<tr>
<td>DEFAULT_HTTP_PORT</td>
<td>PLS_INTEGER</td>
</tr>
<tr>
<td>DEFAULT_HTTPS_PORT</td>
<td>PLS_INTEGER</td>
</tr>
</tbody>
</table>
### Table 265-3  UTL_HTTP Constants - HTTP 1.1 Status Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_CONTINUE</td>
<td>PLS_INTEGER</td>
<td>100</td>
<td>The client should continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server.</td>
</tr>
<tr>
<td>HTTP_SWITCHING_PROTOCOLS</td>
<td>PLS_INTEGER</td>
<td>101</td>
<td>The server understands and is willing to comply with the client's request, through the Upgrade message header field, for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.</td>
</tr>
<tr>
<td>HTTP_OK</td>
<td>PLS_INTEGER</td>
<td>200</td>
<td>The request has succeeded. The information returned with the response is dependent on the method used in the request</td>
</tr>
<tr>
<td>HTTP_CREATED CONSTANT</td>
<td>PLS_INTEGER</td>
<td>201</td>
<td>The request has been fulfilled and resulted in a new resource being created.</td>
</tr>
<tr>
<td>HTTP_ACCEPTED</td>
<td>PLS_INTEGER</td>
<td>202</td>
<td>The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place.</td>
</tr>
<tr>
<td>HTTP_NON_AUTHENTICATIVE_INFO</td>
<td>PLS_INTEGER</td>
<td>203</td>
<td>The returned metainformation in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy.</td>
</tr>
<tr>
<td>HTTP_NO_CONTENT</td>
<td>PLS_INTEGER</td>
<td>204</td>
<td>The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation.</td>
</tr>
<tr>
<td>HTTP_RESET_CONTENT</td>
<td>PLS_INTEGER</td>
<td>205</td>
<td>The server has fulfilled the request and the user agent should reset the document view which caused the request to be sent. The response must not include an entity.</td>
</tr>
<tr>
<td>HTTP_PARTIAL_CONTENT</td>
<td>PLS_INTEGER</td>
<td>206</td>
<td>The server has fulfilled the partial GET request for the resource.</td>
</tr>
<tr>
<td>HTTP_MULTIPLE_CHOICES</td>
<td>PLS_INTEGER</td>
<td>300</td>
<td>The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent-driven negotiation information is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.</td>
</tr>
<tr>
<td>HTTP_MOVED_PERMANENTLY</td>
<td>PLS_INTEGER</td>
<td>301</td>
<td>The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs.</td>
</tr>
<tr>
<td>HTTP_FOUND CONSTANT</td>
<td>PLS_INTEGER</td>
<td>302</td>
<td>The requested resource resides temporarily under a different URI.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HTTP_SEE_OTHER</td>
<td>PLS_INTEGER</td>
<td>303</td>
<td>The response to the request can be found under a different URI and should be retrieved using a GET method on that resource.</td>
</tr>
<tr>
<td>HTTP_NOT_MODIFIED</td>
<td>PLS_INTEGER</td>
<td>304</td>
<td>If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server responds with this status code.</td>
</tr>
<tr>
<td>HTTP_USE_PROXY</td>
<td>PLS_INTEGER</td>
<td>305</td>
<td>The requested resource must be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy.</td>
</tr>
<tr>
<td>HTTP_TEMPORARY_REDIRECT</td>
<td>PLS_INTEGER</td>
<td>307</td>
<td>The requested resource resides temporarily under a different URI.</td>
</tr>
<tr>
<td>HTTP_BAD_REQUEST</td>
<td>PLS_INTEGER</td>
<td>400</td>
<td>The request could not be understood by the server due to malformed syntax.</td>
</tr>
<tr>
<td>HTTP_UNAUTHORIZED</td>
<td>PLS_INTEGER</td>
<td>401</td>
<td>The request requires user authentication. The client may repeat the request with a suitable Authorization header field. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.</td>
</tr>
<tr>
<td>HTTP_PAYMENT_REQUIRED</td>
<td>PLS_INTEGER</td>
<td>402</td>
<td>This code is reserved for future use.</td>
</tr>
<tr>
<td>HTTP_FORBIDDEN</td>
<td>PLS_INTEGER</td>
<td>403</td>
<td>The server understood the request, but is refusing to fulfill it.</td>
</tr>
<tr>
<td>HTTP_NOT_FOUND</td>
<td>PLS_INTEGER</td>
<td>404</td>
<td>The server has not found anything matching the Request-URI.</td>
</tr>
<tr>
<td>HTTP_NOT_ACCEPTABLE</td>
<td>PLS_INTEGER</td>
<td>406</td>
<td>The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.</td>
</tr>
<tr>
<td>HTTP_PROXY_AUTH</td>
<td>PLS_INTEGER</td>
<td>407</td>
<td>This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy.</td>
</tr>
<tr>
<td>HTTP_REQUEST_TIMEOUT</td>
<td>PLS_INTEGER</td>
<td>408</td>
<td>The client did not produce a request within the time that the server was prepared to wait.</td>
</tr>
<tr>
<td>HTTP_CONFLICT</td>
<td>PLS_INTEGER</td>
<td>409</td>
<td>The request could not be completed due to a conflict with the current state of the resource.</td>
</tr>
<tr>
<td>HTTP_GONE</td>
<td>PLS_INTEGER</td>
<td>410</td>
<td>The requested resource is no longer available at the server and no forwarding address is known.</td>
</tr>
<tr>
<td>HTTP_LENGTH_REQUIRED</td>
<td>PLS_INTEGER</td>
<td>411</td>
<td>The server refuses to accept the request without a defined Content-Length.</td>
</tr>
<tr>
<td>HTTP_PRECONDITION</td>
<td>PLS_INTEGER</td>
<td>412</td>
<td>The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.</td>
</tr>
</tbody>
</table>
Table 265-3  (Cont.) UTL_HTTP Constants - HTTP 1.1 Status Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_REQUEST_ENUM</td>
<td>PLS_INTEGER</td>
<td>413</td>
<td>The server is refusing to process a request because the request entity is larger than the server is willing or able to process.</td>
</tr>
<tr>
<td>HTTP_REQUEST_ENUM_TOO_LARGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP_REQUEST_URI</td>
<td>PLS_INTEGER</td>
<td>414</td>
<td>The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.</td>
</tr>
<tr>
<td>HTTP_REQUEST_URI_TOO_LARGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP_UNSUPPORTED_MEDIA_TYPE</td>
<td>PLS_INTEGER</td>
<td>415</td>
<td>The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.</td>
</tr>
<tr>
<td>HTTP_UNSUPPORTED_MEDIA_TYPE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP_REQUEST_ENUM_RANGE_NOT_SATISFIABLE</td>
<td>PLS_INTEGER</td>
<td>416</td>
<td>A server returns a response with this status code if a request included a Range request-header field, and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field.</td>
</tr>
<tr>
<td>HTTP_REQUEST_ENUM_RANGE_NOT_SATISFIABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP_EXPECTATION</td>
<td>PLS_INTEGER</td>
<td>417</td>
<td>The expectation given in an Expect request-header field could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.</td>
</tr>
<tr>
<td>HTTP_EXPECTATION_FAILED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP_NOT_IMPLEMENTED</td>
<td>PLS_INTEGER</td>
<td>501</td>
<td>The server does not support the functionality required to fulfill the request.</td>
</tr>
<tr>
<td>HTTP_BAD_GATEWAY</td>
<td>PLS_INTEGER</td>
<td>502</td>
<td>The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request</td>
</tr>
<tr>
<td>HTTP_BAD_GATEWAY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP_SERVICEUnavailable</td>
<td>PLS_INTEGER</td>
<td>503</td>
<td>The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.</td>
</tr>
<tr>
<td>HTTP_GATEWAY_TIMEOUT</td>
<td>PLS_INTEGER</td>
<td>504</td>
<td>The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (for example, HTTP, FTP, LDAP) or some other auxiliary server (for example, DNS) it needed to access in attempting to complete the request.</td>
</tr>
<tr>
<td>HTTP_VERSION_NOT_SUPPORTED</td>
<td>PLS_INTEGER</td>
<td>505</td>
<td>The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.</td>
</tr>
<tr>
<td>HTTP_VERSION_NOT_SUPPORTED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

265.4 UTL_HTTP Exceptions

Exceptions indicate that the UTL_HTTP package encountered issues.

The following table lists these exceptions. By default, UTL_HTTP raises the exception request_failed when a request fails to execute. If the package is set to raise a detailed exception by set_detailed_excp_support, the rest of the exceptions will be raised directly...
(except for the exception end_of_body, which will be raised by READ_TEXT, READ_LINE, and READ_RAW regardless of the setting).

**Table 265-4  UTL_HTTP Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Reason</th>
<th>Where Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ARGUMENT</td>
<td>29261</td>
<td>The argument passed to the interface is bad</td>
<td>Any HTTP request or response interface when detailed_exception is enabled</td>
</tr>
<tr>
<td>BAD_URL</td>
<td>29262</td>
<td>The requested URL is badly formed</td>
<td>BEGIN_REQUEST, when detailed_exception is enabled</td>
</tr>
<tr>
<td>END_OF_BODY</td>
<td>29266</td>
<td>The end of HTTP response body is reached</td>
<td>READ_RAW, READ_TEXT, and READ_LINE, when detailed_exception is enabled</td>
</tr>
<tr>
<td>HEADER_NOT_FOUND</td>
<td>29265</td>
<td>The header is not found</td>
<td>GET_HEADER, GET_HEADER_BY_NAME, when detailed_exception is enabled</td>
</tr>
<tr>
<td>HTTP_CLIENT_ERROR</td>
<td>29268</td>
<td>From GET_RESPONSE, the response status code indicates that a client error has occurred (status code in 4xx range). Or from begin_request, the HTTP proxy returns a status code in the 4xx range when making an HTTPS request through the proxy.</td>
<td>GET_RESPONSE, BEGIN_REQUEST when detailed_exception is enabled</td>
</tr>
<tr>
<td>HTTP_SERVER_ERROR</td>
<td>29269</td>
<td>From GET_RESPONSE, the response status code indicates that a client error has occurred (status code in 5xx range). Or from begin_request, the HTTP proxy returns a status code in the 5xx range when making an HTTPS request through the proxy.</td>
<td>GET_RESPONSE, BEGIN_REQUEST when detailed_exception is enabled</td>
</tr>
<tr>
<td>NETWORK_ACCESS_DENIED</td>
<td>24247</td>
<td>Access to the remote network host or credentials in an Oracle wallet is denied</td>
<td>BEGIN_REQUEST and SET_AUTHENTICATION_FROM_WALLET when detailed_exception is enabled</td>
</tr>
<tr>
<td>ILLEGAL_CALL</td>
<td>29267</td>
<td>The call to UTL_HTTP is illegal at the current state of the HTTP request</td>
<td>SET_HEADER, SET_AUTHENTICATION, and SET_PERSISTENT_CONN_SUPPORT, when detailed_exception is enabled</td>
</tr>
<tr>
<td>PARTIAL_MULTIBYTE_EXCEPTION</td>
<td>29275</td>
<td>No complete character is read and a partial multibyte character is found at the end of the response body</td>
<td>READ_TEXT and READ_LINE, when detailed_exception is enabled</td>
</tr>
<tr>
<td>PROTOCOL_ERROR</td>
<td>29263</td>
<td>An HTTP protocol error occurs when communicating with the Web server</td>
<td>SET_HEADER, GET_RESPONSE, READ_RAW, READ_TEXT, and READ_LINE, when detailed_exception is enabled</td>
</tr>
</tbody>
</table>
Table 265-4 (Cont.) UTL_HTTP Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Reason</th>
<th>Where Raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_FAILED</td>
<td>29273</td>
<td>The request fails to executes</td>
<td>Any HTTP request or response interface when detailed_exception is disabled</td>
</tr>
<tr>
<td>TOO_MANY_REQUESTS</td>
<td>29270</td>
<td>Too many requests or responses are open</td>
<td>BEGIN_REQUEST, when detailed_exception is enabled</td>
</tr>
<tr>
<td>TRANSFER_TIMEOUT</td>
<td>29276</td>
<td>No data is read and a read timeout occurred</td>
<td>READ_TEXT and READ_LINE, when detailed_exception is enabled</td>
</tr>
<tr>
<td>UNKNOWN_SCHEME</td>
<td>29264</td>
<td>The scheme of the requested URL is unknown</td>
<td>BEGIN_REQUEST and GET_RESPONSE, when detailed_exception is enabled</td>
</tr>
</tbody>
</table>

Note:
The partial_multibyte_char and transfer_timeout exceptions are duplicates of the same exceptions defined in UTL_TCP. They are defined in this package so that the use of this package does not require the knowledge of the UTL_TCP. As those exceptions are duplicates, an exception handle that catches the partial_multibyte_char and transfer_timeout exceptions in this package also catch the exceptions in the UTL_TCP.

For REQUEST and REQUEST_PIECES, the request_failed exception is raised when any exception occurs and detailed_exception is disabled.

265.5 UTL_HTTP Examples

These five examples demonstrate how to use UTL_HTTP.

- General Usage
- Retrieving HTTP Response Headers
- Handling HTTP Authentication
- Retrieving and Restoring Cookies
- Making HTTP Request with Private Wallet and Cookie Table

265.5.1 UTL_HTTP General Usage

This is an general example of UTL_HTTP usage.

```
SET SERVEROUTPUT ON SIZE 40000

DECLARE
    req   UTL_HTTP.REQ;
    resp  UTL_HTTP.RESP;
    value VARCHAR2(1024);
```
BEGIN
    UTL_HTTP.SET_PROXY('proxy.my-company.com', 'corp.my-company.com');
    req := UTL_HTTP.BEGIN_REQUEST('http://www-hr.corp.my-company.com');
    UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
    resp := UTL_HTTP.GET_RESPONSE(req);
    LOOP
        UTL_HTTP.READ_LINE(resp, value, TRUE);
        DBMS_OUTPUT.PUT_LINE(value);
    END LOOP;
    UTL_HTTP.END_RESPONSE(resp);
END;

EXCEPTION
    WHEN UTL_HTTP.END_OF_BODY THEN
        UTL_HTTP.END_RESPONSE(resp);
END;

265.5.2 UTL_HTTP Retrieving HTTP Response Headers

This example shows how UTL_HTTP retrieves HTTP response headers.

SET SERVEROUTPUT ON SIZE 40000

DECLARE
    req UTL_HTTP.REQ;
    resp UTL_HTTP.RESP;
    name VARCHAR2(256);
    value VARCHAR2(1024);
BEGIN
    UTL_HTTP.SET_PROXY('proxy.my-company.com', 'corp.my-company.com');
    req := UTL_HTTP.BEGIN_REQUEST('http://www-hr.corp.my-company.com');
    UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
    resp := UTL_HTTP.GET_RESPONSE(req);
    DBMS_OUTPUT.PUT_LINE('HTTP response status code: ' || resp.status_code);
    DBMS_OUTPUT.PUT_LINE('HTTP response reason phrase: ' || resp.reason_phrase);
    FOR i IN 1..UTL_HTTP.GET_HEADER_COUNT(resp) LOOP
        UTL_HTTP.GET_HEADER(resp, i, name, value);
        DBMS_OUTPUT.PUT_LINE(name || ': ' || value);
    END LOOP;
    UTL_HTTP.END_RESPONSE(resp);
END;

265.5.3 UTL_HTTP Handling HTTP Authentication

This code sample indicates how UTL_HTTP handles HTTP authentication.

SET SERVEROUTPUT ON SIZE 40000

CREATE OR REPLACE PROCEDURE get_page (url IN VARCHAR2,
                                        username IN VARCHAR2 DEFAULT NULL,
                                        password IN VARCHAR2 DEFAULT NULL,
                                        realm IN VARCHAR2 DEFAULT NULL) AS

    req UTL_HTTP.REQ;
    resp UTL_HTTP.RESP;
    my_scheme VARCHAR2(256);
    my_realm VARCHAR2(256);
    name VARCHAR2(256);
    value VARCHAR2(256);
BEGIN
    -- Turn off checking of status code. We will check it by ourselves.
    UTL_HTTP.SET_RESPONSE_ERROR_CHECK(FALSE);
    req := UTL_HTTP.BEGIN_REQUEST(url);

IF (username IS NOT NULL) THEN
    UTL_HTTP.SET_AUTHENTICATION(req, username, password); -- Use HTTP Basic Authen.
END IF;

resp := UTL_HTTP.GET_RESPONSE(req);

IF (resp.status_code = UTL_HTTP.HTTP_UNAUTHORIZED) THEN
    UTL_HTTP.GET_AUTHENTICATION(resp, my_scheme, my_realm, FALSE);
    DBMS_OUTPUT.PUT_LINE('Web proxy server is protected.');
    DBMS_OUTPUT.PUT('Please provide the required ' || my_scheme || ' authentication
    username/password for realm ' || my_realm || ' for the proxy server.');
    UTL_HTTP.END_RESPONSE(resp);
    RETURN;
ELSIF (resp.status_code = UTL_HTTP.HTTP_PROXY_AUTH_REQUIRED) THEN
    UTL_HTTP.GET_AUTHENTICATION(resp, my_scheme, my_realm, TRUE);
    DBMS_OUTPUT.PUT_LINE('Web page ' || url || ' is protected.';
    DBMS_OUTPUT.PUT('Please provide the required ' || my_scheme || ' authentication
    username/password for realm ' || my_realm || ' for the Web page.';
    UTL_HTTP.END_RESPONSE(resp);
    RETURN;
END IF;
FOR i IN 1..UTL_HTTP.GET_HEADER_COUNT(resp) LOOP
    UTL_HTTP.GET_HEADER(resp, i, name, value);
    DBMS_OUTPUT.PUT_LINE(name || ': ' || value);
END LOOP;
UTL_HTTP.END_RESPONSE(resp);
END;

Chapter 265
UTL_HTTP Examples

265.5.4 UTL_HTTP Retrieving and Restoring Cookies

This example show how UTL_HTTP can be used to retrieve and restore cookies.

CREATE TABLE my_cookies (  
session_id  INTEGER,  
name        VARCHAR2(256),  
value       VARCHAR2(1024),  
domain      VARCHAR2(256),  
expire      DATE,  
path        VARCHAR2(1024),  
secure      VARCHAR2(1),  
version     INTEGER);

CREATE SEQUENCE session_id;
SET SERVEROUTPUT ON SIZE 40000

REM Retrieve cookies from UTL_HTTP
CREATE OR REPLACE FUNCTION save_cookies RETURN PLS_INTEGER AS  
cookies        UTL_HTTP.COOKIE_TABLE;  
my_session_id  PLS_INTEGER;  
secure         VARCHAR2(1);  
BEGIN  
/* assume that some cookies have been set in previous HTTP requests. */  
UTL_HTTP.GET_COOKIE_TABLE(cookies);
SELECT session_id.nextval INTO my_session_id FROM DUAL;
FOR i in 1..cookies.count LOOP
    IF (cookies(i).secure) THEN
        secure := 'Y';
    ELSE
        secure := 'N';
    END IF;
    DBMS_OUTPUT.PUT_LINE('Session ID: ' || my_session_id || ', Secure: ' || secure);
    END LOOP;
RETURN;
END;
END IF;
INSERT INTO my_cookies
VALUES (my_session_id, cookies(i).name, cookies(i).value,
        cookies(i).domain,
        cookies(i).expire, cookies(i).path, secure, cookies(i).version);
END LOOP;
RETURN my_session_id;
END;
/

REM Retrieve cookies from UTL_HTTP
CREATE OR REPLACE PROCEDURE restore_cookies (this_session_id IN PLS_INTEGER)
AS
    cookies        UTL_HTTP.COOKIE_TABLE;
    cookie         UTL_HTTP.COOKIE;
    i              PLS_INTEGER := 0;
    CURSOR c (c_session_id PLS_INTEGER) IS
        SELECT * FROM my_cookies WHERE session_id = c_session_id;
BEGIN
    FOR r IN c(this_session_id) LOOP
        i := i + 1;
        cookie.name     := r.name;
        cookie.value    := r.value;
        cookie.domain   := r.domain;
        cookie.expire   := r.expire;
        cookie.path     := r.path;
        IF (r.secure = 'Y') THEN
            cookie.secure := TRUE;
        ELSE
            cookie.secure := FALSE;
        END IF;
        cookie.version := r.version;
        cookies(i) := cookie;
    END LOOP;
    UTL_HTTP.CLEAR_COOKIES;
    UTL_HTTP.ADD_COOKIES(cookies);
END;
/

265.5.5 UTL_HTTP Making HTTP Request with Private Wallet and Cookie Table

This example shows how UTL_HTTP creates a request context with a wallet and cookie table, then makes an HTTP Request using that wallet and cookie table.

SET SERVEROUTPUT ON SIZE 40000
CREATE OR REPLACE PROCEDURE DISPLAY_PAGE(url IN VARCHAR2) AS
    request_context UTL_HTTP.REQUEST_CONTEXT_KEY;
    req             UTL_HTTP.REQ;
    resp            UTL_HTTP.RESP;
    data            VARCHAR2(1024);
BEGIN
    -- Create a request context with its wallet and cookie table
    request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(
        wallet_path => 'file:/oracle/wallets/test/
        wallet',
        url
    );
    ...
wallet_password => '******',
enable_cookies => TRUE,
max_cookies => 300,
max_cookies_per_site => 20);

-- Make a HTTP request using the private wallet and cookie table in the request context
req := UTL_HTTP.BEGIN_REQUEST(
    url => url,
    request_context => request_context);
resp := UTL_HTTP.GET_RESPONSE(req);
BEGIN
    LOOP
        UTL_HTTP.READ_TEXT(resp, data);
        DBMS_OUTPUT.PUT(data);
    END LOOP;
EXCEPTION
    WHEN UTL_HTTP.END_OF_BODY THEN
        UTL_HTTP.END_RESPONSE(resp);
END;

-- Destroy the request context
UTL_HTTP.DESTROY_REQUEST_CONTEXT(request_context);
END;
BEGIN
    DISPLAY_PAGE('https://www.example.com/');
END;
/

265.6 UTL_HTTP Data Structures

Data structures are used to represent requests, responses, cookies, connections, and request context.

- **REQ Type**
- **RESP Type**
- **COOKIE and COOKIE_TABLE Types**
- **CONNECTION Type**
- **REQUEST_CONTEXT_KEY Type**

265.6.1 REQ Type

Use this PL/SQL record type to represent an HTTP request.

**Syntax**

```
TYPE req IS RECORD (
    url VARCHAR2(32767),
    method VARCHAR2(64),
    http_version VARCHAR2(64));
```
Parameters

Table 265-5    REQ Type Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the HTTP request. It is set after the request is created by BEGIN_REQUEST.</td>
</tr>
<tr>
<td>method</td>
<td>The method to be performed on the resource identified by the URL. It is set after the request is created by BEGIN_REQUEST.</td>
</tr>
<tr>
<td>http_version</td>
<td>The HTTP protocol version used to send the request. It is set after the request is created by BEGIN_REQUEST.</td>
</tr>
</tbody>
</table>

Usage Notes

The information returned in REQ from the interface begin_request is for read-only. Changing the field values in the record has no effect on the request.

There are other fields in REQ record type whose names begin with the prefix private_. The fields are private and are intended for use by implementation of the UTL_HTTP package. You should not modify the fields.

265.6.2 REQUEST_CONTEXT_KEY Type

This type is used to represent the key to a request context.

A request context is a context that holds a private wallet and cookie table to make a HTTP request. This private wallet and cookie table, unlike the session-wide ones maintained in the package, will not be shared with other HTTP requests within the database session.

Syntax

SUBTYPE request_context_key IS PLS_INTEGER;

Usage Notes

To provide enhanced security, UTL_HTTP allows PL/SQL programs to create request contexts. A request context is a private context that holds a wallet and a cookie table that will not be shared with other programs in the same database session when making HTTP requests and receiving HTTP responses. PL/SQL programs should use request contexts when they need to use wallets or cookies that contain sensitive information such as authentication credentials.

265.6.3 RESP Type

This PL/SQL record type is used to represent an HTTP response.

Syntax

TYPE resp IS RECORD {
  status_code   PLS_INTEGER,
  reason_phrase VARCHAR2(256),
  http_version  VARCHAR2(64));
Parameters

Table 265-6  RESP Type Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status_code</td>
<td>The status code returned by the Web server. It is a 3-digit integer that</td>
</tr>
<tr>
<td></td>
<td>indicates the results of the HTTP request as handled by the Web server. It</td>
</tr>
<tr>
<td></td>
<td>is set after the response is processed by GET_RESPONSE.</td>
</tr>
<tr>
<td>reason_phrase</td>
<td>The short textual message returned by the Web server that describe the</td>
</tr>
<tr>
<td></td>
<td>status code. It gives a brief description of the results of the HTTP</td>
</tr>
<tr>
<td></td>
<td>request as handled by the Web server. It is set after the response is</td>
</tr>
<tr>
<td></td>
<td>processed by GET_RESPONSE.</td>
</tr>
<tr>
<td>http_version</td>
<td>The HTTP protocol version used in the HTTP response. It is set after the</td>
</tr>
<tr>
<td></td>
<td>response is processed by GET_RESPONSE.</td>
</tr>
</tbody>
</table>

Usage Notes

The information returned in RESP from the interface GET_RESPONSE is read-only. There are other fields in the RESP record type whose names begin with the prefix private_. The fields are private and are intended for use by implementation of the UTL_HTTP package. You should not modify the fields.

265.6.4 COOKIE and COOKIE_TABLE Types

The COOKIE type is the PL/SQL record type that represents an HTTP cookie. The COOKIE_TABLE type is a PL/SQL index-by-table type that represents a collection of HTTP cookies.

Syntax

```
TYPE cookie IS RECORD (
   name  VARCHAR2(256),
   value VARCHAR2(1024),
   domain VARCHAR2(256),
   expire TIMESTAMP WITH TIME ZONE,
   path  VARCHAR2(1024),
   secure BOOLEAN,
   version PLS_INTEGER,
   comment VARCHAR2(1024));
```

```
TYPE cookie_table IS TABLE OF cookie INDEX BY binary_integer;
```

Fields of COOKIE Record Type

Table 265-7 shows the fields for the COOKIE and COOKIE_TABLE record types.

Table 265-7  Fields of COOKIE and COOKIE_TABLE Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the HTTP cookie</td>
</tr>
<tr>
<td>value</td>
<td>The value of the cookie</td>
</tr>
<tr>
<td>domain</td>
<td>The domain for which the cookie is valid</td>
</tr>
</tbody>
</table>
### Table 265-7 (Cont.) Fields of COOKIE and COOKIE_TABLE Type

| Field     | Description                                                                emaximize
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>expire</td>
<td>The time by which the cookie will expire</td>
</tr>
<tr>
<td>path</td>
<td>The subset of URLs to which the cookie applies</td>
</tr>
<tr>
<td>secure</td>
<td>Should the cookie be returned to the Web server using secured means only.</td>
</tr>
<tr>
<td>version</td>
<td>The version of the HTTP cookie specification the cookie conforms.</td>
</tr>
<tr>
<td>comment</td>
<td>The comment that describes the intended use of the cookie.</td>
</tr>
</tbody>
</table>

**Usage Notes**

PL/SQL programs do not usually examine or change the cookie information stored in the UTL_HTTP package. The cookies are maintained by the package transparently. They are maintained inside the UTL_HTTP package, and they last for the duration of the database session only. PL/SQL applications that require cookies to be maintained beyond the lifetime of a database session can read the cookies using `GET_COOKIES`, store them persistently in a database table, and re-store the cookies back in the package using `ADD_COOKIES` in the next database session. All the fields in the cookie record, except for the comment field, must be stored. Do not alter the cookie information, which can result in an application error in the Web server or compromise the security of the PL/SQL and the Web server applications. See "Retrieving and Restoring Cookies".

### 265.6.5 CONNECTION Type

Use the PL/SQL record type to represent the remote hosts and TCP/IP ports of a network connection that is kept persistent after an HTTP request is completed, according to the HTTP 1.1 protocol specification. The persistent network connection may be reused by a subsequent HTTP request to the same host and port. The subsequent HTTP request may be completed faster because the network connection latency is avoided. `connection_table` is a PL/SQL table of `connection`.

For a direct HTTP persistent connection to a Web server, the `host` and `port` fields contain the host name and TCP/IP port number of the Web server. The `proxy_host` and `proxy_port` fields are not set. For an HTTP persistent connection that was previously used to connect to a Web server using a proxy, the `proxy_host` and `proxy_port` fields contain the host name and TCP/IP port number of the proxy server. The host and port fields are not set, which indicates that the persistent connection, while connected to a proxy server, is not bound to any particular target Web server. An HTTP persistent connection to a proxy server can be used to access any target Web server that is using a proxy.

The `SSL` field indicates if Secured Socket Layer (SSL) is being used in an HTTP persistent connection. An HTTPS request is an HTTP request made over SSL. For an HTTPS (SSL) persistent connection connected using a proxy, the host and port fields contain the host name and TCP/IP port number of the target HTTPS Web server and the fields will always be set. An HTTPS persistent connection to an HTTPS Web...
server using a proxy server can only be reused to make another request to the same target Web server.

**Syntax**

```plaintext
TYPE connection IS RECORD (
    host  VARCHAR2(256),
    port  PLS_INTEGER,
    proxy_host  VARCHAR2(256),
    proxy_port  PLS_INTEGER,
    ssl  BOOLEAN);
```

```plaintext
TYPE connection_table IS TABLE OF connection INDEX BY BINARY_INTEGER;
```

### 265.7 UTL_HTTP Operations

These topics provide information about how **UTL_HTTP** makes HTTP requests from SQL and PL/SQL.

- Operational Flow
- Simple HTTP Fetches
- HTTP Requests
- HTTP Responses
- HTTP Persistent Connections
- Error Conditions
- Session Settings
- Request Context
- External Password Store

#### 265.7.1 UTL_HTTP Operational Flow

The **UTL_HTTP** package provides access to the HTTP protocol.

The interfaces must be called in the order shown in the following illustration, or an exception will be raised.
The following can be called at any time:
• Non-protocol interfaces that manipulate cookies
  – GET_COOKIE_COUNT
  – GET_COOKIES
  – ADD_COOKIES
  – CLEAR.Cookies

• Persistent connections
  – GET_PERSISTENT_CONN_COUNT
  – GET_PERSISTENT_CONNS
  – CLOSE_PERSISTENT_CONN
  – CLOSE_PERSISTENT_CONNS

• Interfaces that manipulate attributes and configurations of the UTL_HTTP package in the current session
  – SET_PROXY
  – GET_PROXY
  – SET_COOKIE_SUPPORT
  – GET_COOKIE_SUPPORT
  – SET_FOLLOW_REDIRECT
  – GET_FOLLOW_REDIRECT
  – SET_BODY_CHARSET
  – GET_BODY_CHARSET
  – SET_PERSISTENT_CONN_SUPPORT
  – GET_PERSISTENT_CONN_SUPPORT
  – SET_DETAILED_EXCP_SUPPORT
  – GET_DETAILED_EXCP_SUPPORT
  – SET_WALLET
  – SET_TRANSFER_TIMEOUT
  – GET_TRANSFER_TIMEOUT

• Interfaces that retrieve the last detailed exception code and message UTL_HTTP package in the current session
  – GET_DETAILED_SQLCODE
  – GET_DETAILED_SQLERRM
265.7.2 UTL_HTTP Simple HTTP Fetches

REQUEST and REQUEST_PIECES take a string uniform resource locator (URL), contact that site, and return the data (typically HTML) obtained from that site.

You should not expect REQUEST or REQUEST_PIECES to succeed in contacting a URL unless you can contact that URL by using a browser on the same machine (and with the same privileges, environment variables, and so on.)

If REQUEST or REQUEST_PIECES fails (for example, if it raises an exception, or if it returns an HTML-formatted error message, but you believe that the URL argument is correct), then try contacting that same URL with a browser to verify network availability from your machine. You may have a proxy server set in your browser that needs to be set with each REQUEST or REQUEST_PIECES call using the optional proxy parameter.

Note:

UTL_HTTP can also use environment variables to specify its proxy behavior. For example, on UNIX, setting the environment variable http_proxy to a URL uses that service as the proxy server for HTTP requests. Setting the environment variable no_proxy to a domain name does not use the HTTP proxy server for URLs in that domain. When the UTL_HTTP package is executed in the Oracle database server, the environment variables are the ones that are set when the database instance is started.

See Also:

Simple HTTP Fetches in a Single Call Subprograms

265.7.3 UTL_HTTP HTTP Requests

The HTTP Requests group of subprograms begin an HTTP request, manipulate attributes, and send the request information to the Web server. When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect,
body character set, persistent-connection support, and transfer timeout of the current session. The settings can be changed by calling the request interface.

See Also:
HTTP Requests Subprograms

265.7.4 UTL_HTTP HTTP Responses

The HTTP Responses group of subprograms manipulate an HTTP response obtained from GET_RESPONSE and receive response information from the Web server.

When a response is created for a request, it inherits settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout from the request. Only the body character set can be changed by calling the response interface.

See Also:
HTTP Responses Subprograms

265.7.5 UTL_HTTP HTTP Cookies

The UTL_HTTP package provides subprograms to manipulate HTTP cookies.

See Also:
HTTP Cookies Subprograms

265.7.6 UTL_HTTP HTTP Persistent Connections

The UTL_HTTP package provides subprograms to manipulate persistent connections.

See Also:
HTTP Persistent Connections Subprograms
265.7.7 UTL_HTTP Error Conditions

The UTL_HTTP package provides subprograms to retrieve error information.

See Also:
Error Conditions Subprograms

265.7.8 UTL_HTTP Session Settings

Session settings manipulate the configuration and default behavior of UTL_HTTP when HTTP requests are executed within a database user session.

When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout of the current session. Those settings can be changed later by calling the request interface. When a response is created for a request, it inherits those settings from the request. Only the body character set can be changed later by calling the response interface.

See Also:
Session Settings Subprograms

265.7.9 UTL_HTTP Request Context

The UTL_HTTP package maintains a common wallet and cookie table within the database session that all HTTP requests and responses share. This makes it easy for users to share the wallet or to maintain application state in the cookies within the session. However, if an application stores private information in the wallet or in the cookies that it does not want to share with other applications in the same database session, it may define a request context to hold its own wallet and cookie table and use this request context to make HTTP requests.

See Also:
HTTP Requests Subprograms

265.7.10 UTL_HTTP External Password Store

The UTL_HTTP package allows HTTP password credentials to be stored in an Oracle wallet's external password store. The external password store provides an easy but
secure storage for passwords and frees the application developers from the need to maintain their own storage.

See Also:

SET_AUTHENTICATION_FROM_WALLET Procedure

265.8 UTL_HTTP Subprogram Groups

This section describes the UTL_HTTP subprograms. They are grouped by function.

- Simple HTTP Fetches in a Single Call Subprograms
- Session Settings Subprograms
- HTTP Requests Subprograms
- HTTP Request Contexts Subprograms
- HTTP Responses Subprograms
- HTTP Cookies Subprograms
- HTTP Persistent Connections Subprograms
- Error Conditions Subprograms

265.8.1 UTL_HTTP Simple HTTP Fetches in a Single Call Subprograms

REQUEST and REQUESTPieces take a string uniform resource locator (URL), contact that site, and return the data (typically HTML) obtained from that site.

Table 265-8 UTL_HTTP Subprograms—Simple HTTP Fetches in a Single Call

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST Function</td>
<td>Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.</td>
</tr>
<tr>
<td>REQUEST_PIECES Function</td>
<td>Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL</td>
</tr>
</tbody>
</table>

265.8.2 UTL_HTTP Session Settings Subprograms

This table lists and briefly describes the UTL_HTTP Session Settings Subprograms.

Table 265-9 UTL_HTTP Subprograms—Session Settings

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_BODY_CHARSET Procedure</td>
<td>Retrieves the default character set of the body of all future HTTP requests</td>
</tr>
<tr>
<td>GET_COOKIE_SUPPORT Procedure</td>
<td>Retrieves the current cookie support settings</td>
</tr>
</tbody>
</table>
Table 265-9  (Cont.) UTL_HTTP Subprograms—Session Settings

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETDetalleDEXCP_SUPPORT Procedure</td>
<td>Checks if the UTL_HTTP package will raise a detailed exception or not</td>
</tr>
<tr>
<td>GET_FOLLOW_REDIRECT Procedure</td>
<td>Retrieves the follow-redirect setting in the current session</td>
</tr>
<tr>
<td>GET_PERSISTENT_CONN_SUPPORT Procedure</td>
<td>Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session</td>
</tr>
<tr>
<td>GET_PROXY Procedure</td>
<td>Retrieves the current proxy settings</td>
</tr>
<tr>
<td>GET_RESPONSE_ERROR_CHECK Procedure</td>
<td>Checks if the response error check is set or not</td>
</tr>
<tr>
<td>GET_TRANSFER_TIMEOUT Procedure</td>
<td>Sets the default character set of the body of all future HTTP requests when the media type is text and the character set is not specified in the Content-Type header</td>
</tr>
<tr>
<td>SET_COOKIE_SUPPORT Procedures</td>
<td>Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session</td>
</tr>
<tr>
<td>SET_DETAILED_EXCP_SUPPORT Procedure</td>
<td>Sets the UTL_HTTP package to raise a detailed exception</td>
</tr>
<tr>
<td>SET_FOLLOW_REDIRECT Procedures</td>
<td>Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP responses to future requests in the GET_RESPONSE function</td>
</tr>
<tr>
<td>SET_PERSISTENT_CONN_SUPPORT Procedure</td>
<td>Sets whether or not future HTTP requests will support the HTTP 1.1 persistent connection; sets the maximum number of persistent connections maintained in the current database user session</td>
</tr>
<tr>
<td>SET_PROXY Procedure</td>
<td>Sets the proxy to be used for requests of HTTP or other protocols</td>
</tr>
<tr>
<td>SET_RESPONSE_ERROR_CHECK Procedure</td>
<td>Sets whether or not GET_RESPONSE raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges</td>
</tr>
<tr>
<td>SET_TRANSFER_TIMEOUT Procedure</td>
<td>Sets the timeout value for UTL_HTTP to read the HTTP response from the Web server or proxy server</td>
</tr>
<tr>
<td>SET_WALLET Procedure</td>
<td>Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS</td>
</tr>
</tbody>
</table>

265.8.3 UTL_HTTP HTTP Requests Subprograms

This table lists and briefly describes the UTL_HTTP HTTP Requests.

Table 265-10  UTL_HTTP Subprograms—HTTP Requests

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_REQUEST Function</td>
<td>Begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line.</td>
</tr>
<tr>
<td>SET_HEADER Procedure</td>
<td>Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.</td>
</tr>
</tbody>
</table>
### Table 265-10  (Cont.) UTL_HTTP Subprograms—HTTP Requests

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_AUTHENTICATION Procedure</td>
<td>Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.</td>
</tr>
<tr>
<td>SET_AUTHENTICATION_FROM_WALLET Procedure</td>
<td>Sets the HTTP authentication information in the HTTP request header needed for the request to be authorized by the Web server using the username and password credential stored in the Oracle wallet.</td>
</tr>
<tr>
<td>SET_BODY_CHARSET Procedures</td>
<td>Sets the character set of the request body when the media type is text but the character set is not specified in the Content-Type header</td>
</tr>
<tr>
<td>SET_COOKIE_SUPPORT Procedures</td>
<td>Enables or disables support for the HTTP cookies in the request</td>
</tr>
<tr>
<td>SET_FOLLOW_REDIRECT Procedures</td>
<td>Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP response to this request in the GET_RESPONSE Function</td>
</tr>
<tr>
<td>SET_PERSISTENT_CONN_SUPPORT Procedure</td>
<td>Enables or disables support for the HTTP 1.1 persistent-connection in the request</td>
</tr>
<tr>
<td>SET_PROXY Procedure</td>
<td>Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP)</td>
</tr>
<tr>
<td>WRITE_RAW Procedure</td>
<td>Writes some binary data in the HTTP request body</td>
</tr>
<tr>
<td>WRITE_TEXT Procedure</td>
<td>Writes some text data in the HTTP request body</td>
</tr>
</tbody>
</table>

#### 265.8.4 UTL_HTTP HTTP Request Contexts Subprograms

UTL_HTTP HTTP Request Contexts subprograms create or destroy a request context. The following table lists and briefly describes the UTL_HTTP HTTP Request Contexts.

### Table 265-11  UTL_HTTP Subprograms—HTTP Request Contexts

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_REQUEST_CONTEXT Function</td>
<td>Creates a request context in UTL_HTTP for a wallet and a cookie table</td>
</tr>
<tr>
<td>DESTROY_REQUEST_CONTEXT Procedure</td>
<td>Destroys a request context in UTL_HTTP</td>
</tr>
</tbody>
</table>

#### 265.8.5 UTL_HTTP HTTP Responses Subprograms

This table lists and briefly describes the HTTP Responses Subprograms of UTL_HTTP.

### Table 265-12  UTL_HTTP Subprograms—HTTP Responses

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_RESPONSE Procedure</td>
<td>Ends the HTTP response. It completes the HTTP request and response.</td>
</tr>
<tr>
<td>GET_AUTHENTICATION Procedure</td>
<td>Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header</td>
</tr>
</tbody>
</table>
Table 265-12 (Cont.) UTL_HTTP Subprograms—HTTP Responses

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_HEADER Procedure</td>
<td>Returns the n&lt;sup&gt;th&lt;/sup&gt; HTTP response header name and value returned in the response</td>
</tr>
<tr>
<td>GET_HEADER_BY_NAME Procedure</td>
<td>Returns the HTTP response header value returned in the response given the name of the header</td>
</tr>
<tr>
<td>GET_HEADER_COUNT Function</td>
<td>Returns the number of HTTP response headers returned in the response</td>
</tr>
<tr>
<td>GET_RESPONSE Function</td>
<td>Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed.</td>
</tr>
<tr>
<td>READ_LINE Procedure</td>
<td>Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer</td>
</tr>
<tr>
<td>READ_RAW Procedure</td>
<td>Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer</td>
</tr>
<tr>
<td>READ_TEXT Procedure</td>
<td>Reads the HTTP response body in text form and returns the output in the caller-supplied buffer</td>
</tr>
<tr>
<td>SET_BODY_CHARSET Procedures</td>
<td>Sets the character set of the response body when the media type is &quot;text&quot; but the character set is not specified in the Content-Type header</td>
</tr>
</tbody>
</table>

265.8.6 UTL_HTTP HTTP Cookies Subprograms

The HTTP cookies subprograms manages cookies in the UTL_HTTP package.

The following table lists and briefly describes the HTTP cookies subprograms of UTL_HTTP.

Table 265-13 UTL_HTTP Subprograms—HTTP Cookies

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_COOKIES Procedure</td>
<td>Add the cookies either to a request context or to the UTL_HTTP package's session state</td>
</tr>
<tr>
<td>CLEAR_COOKIES Procedure</td>
<td>Clears all the cookies maintained either in a request context or in the UTL_HTTP package's session state</td>
</tr>
<tr>
<td>GET_COOKIE_COUNT Function</td>
<td>Returns the number of cookies maintained either in a request context or in the UTL_HTTP package's session states</td>
</tr>
<tr>
<td>GET_COOKIES Function</td>
<td>Returns all the cookies maintained either in a request context or in the UTL_HTTP package's session state</td>
</tr>
</tbody>
</table>

265.8.7 UTL_HTTP HTTP Persistent Connections Subprograms

This table lists and briefly describes the UTL_HTTP HTTP Persistent Connections subprograms.
Table 265-14  UTL_HTTP Subprograms—HTTP Persistent Connections

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE_PERSISTENT_CONN</td>
<td>Procedure Closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session</td>
</tr>
<tr>
<td>CLOSE_PERSISTENT_CONNS</td>
<td>Procedure Closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session</td>
</tr>
<tr>
<td>GET_PERSISTENT_CONN_COUNT</td>
<td>Function Returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers</td>
</tr>
<tr>
<td>GET_PERSISTENT_CONNS</td>
<td>Procedure Returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers</td>
</tr>
</tbody>
</table>

265.8.8 UTL_HTTP Error Conditions Subprograms

This table lists and briefly describes error conditions subprograms of UTL_HTTP.

Table 265-15  UTL_HTTP Subprograms—Error Conditions

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_DETAILED_SQLCODE Function</td>
<td>Retrieves the detailed SQLCODE of the last exception raised</td>
</tr>
<tr>
<td>GET_DETAILED_SQLERRM Function</td>
<td>Retrieves the detailed SQLERRM of the last exception raised</td>
</tr>
</tbody>
</table>

265.9 Summary of UTL_HTTP Subprograms

This table lists the UTL_HTTP subprograms and briefly describes them.

Table 265-16  UTL_HTTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_COOKIES Procedure</td>
<td>Add the cookies either to a request context or to the UTL_HTTP package's session state</td>
<td>HTTP Cookies Subprograms</td>
</tr>
<tr>
<td>BEGIN_REQUEST Function</td>
<td>Begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>CLEAR_COOKIES Procedure</td>
<td>Clears all the cookies maintained either in a request context or in the UTL_HTTP package's session state</td>
<td>HTTP Cookies Subprograms</td>
</tr>
<tr>
<td>CLOSE_PERSISTENT_CONN Procedure</td>
<td>Closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session</td>
<td>HTTP Persistent Connections Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>CLOSE_PERSISTENT_CONNS</td>
<td>Closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session</td>
<td>HTTP Persistent Connections Subprograms</td>
</tr>
<tr>
<td>CREATE_REQUEST_CONTEXT</td>
<td>Creates a request context in UTL_HTTP for a wallet and a cookie table</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>DESTROY_REQUEST_CONTEXT</td>
<td>Destroys a request context in UTL_HTTP for a wallet and a cookie table</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>END_REQUEST</td>
<td>Ends the HTTP request</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>END_RESPONSE</td>
<td>Ends the HTTP response. It completes the HTTP request and response</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>GET_AUTHENTICATION</td>
<td>Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>GET_BODY_CHARSET</td>
<td>Retrieves the default character set of the body of all future HTTP requests</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>GET_COOKIE_COUNT</td>
<td>Returns the number of cookies currently maintained by the UTL_HTTP package set by all Web servers</td>
<td>HTTP Cookies Subprograms</td>
</tr>
<tr>
<td>GET_COOKIE_SUPPORT</td>
<td>Retrieves the current cookie support settings</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>GET_COOKIES</td>
<td>Returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers</td>
<td>HTTP Cookies Subprograms</td>
</tr>
<tr>
<td>GET_DETAILED_EXCP_SUPPORT</td>
<td>Checks if the UTL_HTTP package will raise a detailed exception or not</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>GET_DETAILED_SQLCODE</td>
<td>Retrieves the detailed SQLCODE of the last exception raised</td>
<td>Error Conditions Subprograms</td>
</tr>
<tr>
<td>GET_DETAILED_SQLERRM</td>
<td>Retrieves the detailed SQLERRM of the last exception raised</td>
<td>Error Conditions Subprograms</td>
</tr>
<tr>
<td>GET_FOLLOW_REDIRECT</td>
<td>Retrieves the follow-redirect setting in the current session</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>GET_HEADER</td>
<td>Returns the n&lt;sup&gt;th&lt;/sup&gt; HTTP response header name and value returned in the response</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>GET_HEADER_BY_NAME</td>
<td>Returns the HTTP response header value returned in the response given the name of the header</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>GET_HEADER_COUNT Function</td>
<td>Returns the number of HTTP response headers returned in the response</td>
<td>HTTP Responses and HTTP Responses Subprograms</td>
</tr>
<tr>
<td>GET_PERSISTENT_CONN_COUNT Function</td>
<td>Returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers</td>
<td>HTTP Persistent Connections Subprograms</td>
</tr>
<tr>
<td>GET_HEADER_COUNT Function</td>
<td>Sees whether or not future HTTP requests will support the HTTP 1.1 persistent connection; sets the maximum number of persistent connections maintained in the current database user session</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>GET_PERSISTENT_CONN_SUPPORT Procedure</td>
<td>Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session (see Session Settings Subprograms)</td>
<td>HTTP Persistent Connections Subprograms</td>
</tr>
<tr>
<td>GET_PERSISTENT_CONNS Procedure</td>
<td>Returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers</td>
<td>HTTP Persistent Connections Subprograms</td>
</tr>
<tr>
<td>GET_PROXY Procedure</td>
<td>Retrieves the current proxy settings</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>GET_RESPONSE Function</td>
<td>Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>GET_RESPONSE_ERROR_CHECK Procedure</td>
<td>Checks if the response error check is set or no</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>GET_TRANSFER_TIMEOUT Procedure</td>
<td>Retrieves the current network transfer timeout value</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>READ_LINE Procedure</td>
<td>Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>READ_RAW Procedure</td>
<td>Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>READ_TEXT Procedure</td>
<td>Reads the HTTP response body in text form and returns the output in the caller-supplied buffer</td>
<td>HTTP Responses Subprograms</td>
</tr>
<tr>
<td>REQUEST Function</td>
<td>Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.</td>
<td>Simple HTTP Fetches in a Single Call Subprograms</td>
</tr>
</tbody>
</table>
### Table 265-16  (Cont.) UTL_HTTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_Pieces Function</td>
<td>Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL</td>
<td>Simple HTTP Fetches in a Single Call Subprograms</td>
</tr>
<tr>
<td>SET_AUTHENTICATION Procedure</td>
<td>Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>SET_AUTHENTICATION_FROM_WALLET Procedure</td>
<td>Sets the HTTP authentication information in the HTTP request header needed for the request to be authorized by the Web server using the username and password credential stored in the Oracle wallet.</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>SET_BODY_CHARSET Procedures</td>
<td>Sets the default character set of the body of all future HTTP requests when the media type is text and the character set is not specified in the Content-Type header</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>SET_BODY_CHARSET Procedures</td>
<td>Sets the character set of the request body when the media type is text but the character set is not specified in the Content-Type header</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>SET_BODY_CHARSET Procedures</td>
<td>Sets the character set of the response body when the media type is “text” but the character set is not specified in the Content-Type header</td>
<td>HTTP Responses Subprograms and Session Settings Subprograms</td>
</tr>
<tr>
<td>SET_COOKIE_SUPPORT Procedures</td>
<td>Enables or disables support for the HTTP cookies in the request</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>SET_DETAILED_EXCP_SUPPORT Procedure</td>
<td>Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>SET_DETAILED_EXCP_SUPPORT Procedure</td>
<td>Sets the UTL_HTTP package to raise a detailed exception</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>SET_FOLLOW_REDIRECT Procedures</td>
<td>Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP response to this request in the GET_RESPONSE function</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>SET_HEADER Procedure</td>
<td>Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP responses to future requests in the GET_RESPONSE function</td>
<td>Session Settings Subprograms</td>
</tr>
</tbody>
</table>
Table 265-16   (Cont.) UTL_HTTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_HEADER Procedure</td>
<td>Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>SET_PERSISTENT_CONN_SUPPORT Procedure</td>
<td>Enables or disables support for the HTTP 1.1 persistent-connection in the request</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>SET_PROXY Procedure</td>
<td>Sets the proxy to be used for requests of HTTP or other protocols</td>
<td>Session Settings and Session Settings Subprograms</td>
</tr>
<tr>
<td>SET_RESPONSE_ERROR_CHECK Procedure</td>
<td>Sets whether or not \texttt{GET_RESPONSE} raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>SET_TRANSFER_TIMEOUT Procedure</td>
<td>Sets the timeout value for \texttt{UTL_HTTP} to read the HTTP response from the Web server or proxy server</td>
<td>Session Settings and Session Settings Subprograms</td>
</tr>
<tr>
<td>SET_WALLET Procedure</td>
<td>Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS</td>
<td>Session Settings Subprograms</td>
</tr>
<tr>
<td>WRITE_LINE Procedure</td>
<td>Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP)</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>WRITE_RAW Procedure</td>
<td>Writes some binary data in the HTTP request body</td>
<td>HTTP Requests Subprograms</td>
</tr>
<tr>
<td>WRITE_TEXT Procedure</td>
<td>Writes some text data in the HTTP request body</td>
<td>HTTP Requests Subprograms</td>
</tr>
</tbody>
</table>

265.9.1 ADD_COOKIES Procedure

This procedure adds the cookies either to a request context or to the \texttt{UTL\_HTTP} package’s session state.

\begin{itemize}
\item \textbf{See Also:}
\item \texttt{HTTP Cookies} and \texttt{HTTP Cookies Subprograms}
\end{itemize}

Syntax

```
UTL_HTTP.ADD_COOKIES (  
    cookies        IN  cookie_table,  
    request_context IN request_context_key DEFAULT NULL);
```
Parameters

Table 265-17    ADD_COOKIES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookies</td>
<td>The cookies to be added</td>
</tr>
<tr>
<td>request_context</td>
<td>Request context to add the cookies. If NULL, the cookies will be added to the UTL_HTTP package's session state instead.</td>
</tr>
</tbody>
</table>

Usage Notes

The cookies that the package currently maintains are not cleared before new cookies are added.

265.9.2 BEGIN_REQUEST Function

This function begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line. The PL/SQL program continues the request by calling some other interface to complete the request.

The URL may contain the username and password needed to authenticate the request to the server. The format is:

```
scheme://[user[:password]@]host[:port]/[...]
```

See Also:

HTTP Requests and HTTP Requests Subprograms

Syntax

```
UTL_HTTP.BEGIN_REQUEST (  
    url               IN  VARCHAR2,  
    method            IN  VARCHAR2 DEFAULT 'GET',  
    http_version      IN  VARCHAR2 DEFAULT NULL,  
    request_context   IN  request_context_key DEFAULT NULL,  
    https_host        IN  VARCHAR2 DEFAULT NULL)  
RETURN req;
```

Parameters

Table 265-18    BEGIN_REQUEST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the HTTP request</td>
</tr>
<tr>
<td>method</td>
<td>The method performed on the resource identified by the URL</td>
</tr>
</tbody>
</table>
Table 265-18  (Cont.) BEGIN_REQUEST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>http_version</td>
<td>The HTTP protocol version that sends the request. The format of the protocol version is HTTP/major-version.minor-version, where major-version and minor-version are positive numbers. If this parameter is set to NULL, UTL_HTTP uses the latest HTTP protocol version that it supports to send the request. The latest version that the package supports is 1.1 and it can be upgraded to a later version. The default is NULL.</td>
</tr>
<tr>
<td>request_context</td>
<td>Request context that holds the private wallet and the cookie table to use in this HTTP request. If this parameter is NULL, the wallet and cookie table shared in the current database session will be used instead.</td>
</tr>
<tr>
<td>https_host</td>
<td>A string representing the host name. If the string does not begin with a wildcard, the string will be used as the host name for server name indication (SNI). If the string begins with a wildcard, the string will be used to match against the common name (CN) of the remote server's certificate for an HTTPS request. If NULL, the host name in the given URL will be used for SNI.</td>
</tr>
</tbody>
</table>

Usage Notes

• The URL passed as an argument to this function is not examined for illegal characters, such as spaces, according to URL specification RFC 2396. You should escape those characters with the UTL_URL package to return illegal and reserved characters. URLs should consist of US-ASCII characters only. See UTL_URL for a list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

• BEGIN_REQUEST can send a URL whose length is up to 32767 bytes. However, different Web servers impose different limits on the length of the URL they can accept. This limit is often about 4000 bytes. If this limit is exceeded, the outcome will depend on the Web server. For example, a Web server might simply drop the HTTP connection without returning a response of any kind. If this happens, a subsequent invocation of the GET_RESPONSE Function will raise the PROTOCOL_ERROR exception.

A URL will be long when its QUERY_STRING (that is, the information that follows the question mark (?)) is long. In general, it is better to send this parameterization in the body of the request using the POST method.

```plsql
req := UTL_HTTP.BEGIN_REQUEST (url=>the_url, method=>'POST');
UTL_HTTP.SET_HEADER (r      =>  req,
  name   =>  'Content-Type',
  value  =>  'application/x-www-form-urlencoded');
UTL_HTTP.SET_HEADER (r      =>   req,
  name   =>   'Content-Length',
  value  =>'  <length of data posted in bytes>');
UTL_HTTP.WRITE_TEXT (r      =>   req,
  data   =>   'p1 = value1&p2=value2...');
resp := UTL_HTTP.GET_RESPONSE (r     =>   req);
...
```
The programmer must determine whether a particular Web server may, or may not, accept data provided in this way.

- An Oracle wallet must be set before accessing Web servers over HTTPS. See the `SET_WALLET Procedure` procedure on how to set up an Oracle wallet. To use SSL client authentication, the client certificate should be stored in the wallet and the caller must have the `use-client-certificates` privilege on the wallet. See "Managing Fine-grained Access to External Network Services" in the *Oracle Database Security Guide* to grant the privilege.

- To connect to the remote Web server directly, or indirectly through a HTTP proxy, the `UTL_HTTP` must have the `connect` ACL privilege to the remote Web server host or the proxy host respectively.

265.9.3 CLEAR_COOKIES Procedure

This procedure clears all the cookies maintained either in a request context or in the `UTL_HTTP` package's session state.

` See Also:  
HTTP Cookies and HTTP Cookies Subprograms`

**Syntax**

```
UTL_HTTP.CLEAR_COOKIES (  
    request_context  IN  request_context_key DEFAULT NULL);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_context</td>
<td>Request context to clear the cookies. If NULL, the cookies maintained in the UTL_HTTP package's session state will be cleared instead.</td>
</tr>
</tbody>
</table>

265.9.4 CLOSE_PERSISTENT_CONN Procedure

This procedure closes an HTTP persistent connection maintained by the `UTL_HTTP` package in the current database session.

` See Also:  
HTTP Persistent Connections and HTTP Persistent Connections Subprograms`
Syntax

UTL_HTTP.CLOSE_PERSISTENT_CONN (conn IN connection);

Parameters

Table 265-20 CLOSE_PERSISTENT_CONN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn</td>
<td>The HTTP persistent connection to close</td>
</tr>
</tbody>
</table>

265.9.5 CLOSE_PERSISTENT_CONNS Procedure

This procedure closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session. This procedure uses a pattern-match approach to decide which persistent connections to close.

To close a group of HTTP persistent connection that share a common property (for example, all connections to a particular host, or all SSL connections), set the particular parameters and leave the rest of the parameters NULL. If a particular parameter is set to NULL when this procedure is called, that parameter will not be used to decide which connections to close.

For example, the following call to the procedure closes all persistent connections to foobar:

UTL_HTTP.CLOSE_PERSISTENT_CONNS(host => 'foobar');

And the following call to the procedure closes all persistent connections through the foobar at TCP/IP port 80:

UTL_HTTP.CLOSE_PERSISTENT_CONNS(proxy_host => 'foobar',
                                 proxy_port => 80);

And the following call to the procedure closes all persistent connections:

UTL_HTTP.CLOSE_PERSISTENT_CONNS;

See Also:

HTTP Persistent Connections and HTTP Persistent Connections Subprograms

Syntax

UTL_HTTP.CLOSE_PERSISTENT_CONNS (host IN VARCHAR2 DEFAULT NULL,
                                 port IN PLS_INTEGER DEFAULT NULL,
                                 proxy_host IN VARCHAR2 DEFAULT NULL,
                                 proxy_port IN PLS INTEGER DEFAULT NULL,
                                 ssl IN BOOLEAN DEFAULT NULL);
Parameters

Table 265-21  CLOSE_PERSISTENT_CONNS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The host for which persistent connections are to be closed</td>
</tr>
<tr>
<td>port</td>
<td>The port number for which persistent connections are to be closed</td>
</tr>
<tr>
<td>proxy_host</td>
<td>The proxy host for which persistent connections are to be closed</td>
</tr>
<tr>
<td>proxy_port</td>
<td>The proxy port for which persistent connections are to be closed</td>
</tr>
<tr>
<td>ssl</td>
<td>Close persistent SSL connection</td>
</tr>
</tbody>
</table>

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

Note that the use of a NULL value in a parameter when this procedure is called means that the caller does not care about its value when the package decides which persistent connection to close. If you want a NULL value in a parameter to match only a NULL value of the parameter of a persistent connection (which is when you want to close a specific persistent connection), you should use the CLOSE_PERSISTENT_CONN procedure that closes a specific persistent connection.

265.9.6 CREATE_REQUEST_CONTEXT Function

This function creates a request context. A request context is a context that holds a wallet and a cookie for private use in making a HTTP request. This allows the HTTP request to use a wallet and a cookie table that will not be shared with other applications making HTTP requests in the same database session.

See Also:

Request Context and HTTP Request Contexts Subprograms

Syntax

```sql
UTL_HTTP.CREATE_REQUEST_CONTEXT (  
    wallet_path          IN VARCHAR2 DEFAULT NULL,  
    wallet_password      IN VARCHAR2 DEFAULT NULL,  
    enable_cookies       IN BOOLEAN  DEFAULT TRUE,  
    max_cookies          IN PLS_INTEGER DEFAULT 300,  
    max_cookies_per_site IN PLS_INTEGER DEFAULT 20)  
RETURN request_context_key;
```
Parameters

Table 265-22  CREATE_REQUEST_CONTEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_path</td>
<td>Directory path that contains the Oracle wallet. The format is file:directory-path.</td>
</tr>
<tr>
<td>wallet_password</td>
<td>The password needed to open the wallet. If the wallet is auto-login enabled, the password may be omitted and should be set to NULL. See the Oracle Database Enterprise User Security Administrator's Guide for detailed information about wallets.</td>
</tr>
<tr>
<td>enable_cookies</td>
<td>Sets whether HTTP requests using this request context should support HTTP cookies or not: TRUE to enable the support, FALSE to disable it.</td>
</tr>
<tr>
<td>max_cookies</td>
<td>Sets the maximum total number of cookies that will be maintained in this request context.</td>
</tr>
<tr>
<td>max_cookies_per_site</td>
<td>Sets the maximum number of cookies per each Web site that will be maintained in this request context.</td>
</tr>
</tbody>
</table>

Return Values

The request context created.

Examples

DECLARE
    request_context  UTL_HTTP.REQUEST_CONTEXT_KEY;
    req              utl_http.req;
BEGIN
  request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(
    wallet_path          => 'file:/oracle/wallets/test_wallets',
    wallet_password      => NULL,
    enable_cookies       => TRUE,
    max_cookies          => 300,
    max_cookies_per_site => 20);
  req := UTL_HTTP.BEGIN_REQUEST(
    url                  => 'http://www.example.com/',
    request_context      => request_context);
END;

265.9.7 DESTROY_REQUEST_CONTEXT Procedure

This procedure destroys a request context in UTL_HTTP. A request context cannot be destroyed when it is in use by a HTTP request or response.

See Also:
- Request Context and HTTP Request Contexts Subprograms
Syntax

UTL_HTTP.DESTROY_REQUEST_CONTEXT (  
    request_context    request_context_key);

Parameters

Table 265-23  DESTROY_REQUEST_CONTEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_context</td>
<td>Request context to destroy</td>
</tr>
</tbody>
</table>

Examples

DECLARE  
    request_context  UTL_HTTP.REQUEST_CONTEXT_KEY;  
BEGIN  
    request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(...);  
    ...  
    UTL_HTTP.DESTROY_REQUEST_CONTEXT(request_context);  
END;

265.9.8 END_REQUEST Procedure

This procedure ends the HTTP request. To terminate the HTTP request without completing the request and waiting for the response, the program can call this procedure. Otherwise, the program should go through the normal sequence of beginning a request, getting the response, and closing the response. The network connection will always be closed and will not be reused.

See Also:

HTTP Requests and HTTP Requests Subprograms

Syntax

UTL_HTTP.END_REQUEST (  
    r  IN OUT NOCOPY req);

Parameters

Table 265-24  END_REQUEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
</tbody>
</table>
265.9.9 END_RESPONSE Procedure

This procedure ends the HTTP response. It completes the HTTP request and response. Unless HTTP 1.1 persistent connection is used in this request, the network connection is also closed.

![See Also:](HTTP Responses and HTTP Responses Subprograms)

Syntax

```
UTL_HTTP.END_RESPONSE (  
    r  IN OUT NOCOPY resp);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
</tbody>
</table>

265.9.10 GET_AUTHENTICATION Procedure

This procedure retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header.

![See Also:](HTTP Responses and HTTP Responses Subprograms)

Syntax

```
UTL_HTTP.GET_AUTHENTICATION(  
    r          IN OUT NOCOPY resp,  
    scheme     OUT VARCHAR2,  
    realm      OUT VARCHAR2,  
    for_proxy  IN BOOLEAN  DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
<tr>
<td>scheme</td>
<td>The scheme for the required HTTP authentication</td>
</tr>
</tbody>
</table>
Table 265-26  (Cont.) GET_AUTHENTICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realm</td>
<td>The realm for the required HTTP authentication</td>
</tr>
<tr>
<td>for_proxy</td>
<td>Returns the HTTP authentication information required for the access to the HTTP proxy server instead of the Web server? Default is FALSE</td>
</tr>
</tbody>
</table>

Usage Notes

When a Web client is unaware that a document is protected, at least two HTTP requests are required for the document to be retrieved. In the first HTTP request, the Web client makes the request without supplying required authentication information; so the request is denied. The Web client can determine the authentication information required for the request to be authorized by calling GET_AUTHENTICATION. The Web client makes the second request and supplies the required authentication information with SET_AUTHORIZATION. If the authentication information can be verified by the Web server, the request will succeed and the requested document is returned. Before making the request, if the Web client knows that authentication information is required, it can supply the required authentication information in the first request, thus saving an extra request.

265.9.11 GET_BODY_CHARSET Procedure

This procedure retrieves the default character set of the body of all future HTTP requests.

Syntax

```
UTL_HTTP.GET_BODY_CHARSET (.charset OUT NOCOPY VARCHAR2);
```

Parameters

Table 265-27  GET_BODY_CHARSET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>The default character set of the body of all future HTTP requests</td>
</tr>
</tbody>
</table>

See Also:

Session Settings and Session Settings Subprograms
265.9.12 GET_COOKIE_COUNT Function

This function returns the number of cookies maintained either in a request context or in the UTL_HTTP package's session state.

**See Also:**
- HTTP Cookies and HTTP Cookies Subprograms

**Syntax**

```sql
UTL_HTTP.GET_COOKIE_COUNT (  
    request_context  IN  request_context_key DEFAULT NULL)  
RETURN PLS_INTEGER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_context</td>
<td>Request context to return the cookie count for. If NULL, the cookie count maintained in the UTL_HTTP package's session state will be returned instead.</td>
</tr>
</tbody>
</table>

265.9.13 GET_COOKIE_SUPPORT Procedure

This procedure retrieves the current cookie support settings.

**See Also:**
- Session Settings and Session Settings Subprograms

**Syntax**

```sql
UTL_HTTP.GET_COOKIE_SUPPORT (  
    enable                OUT BOOLEAN,  
    max_cookies           OUT PLS_INTEGER,  
    max_cookies_per_site  OUT PLS_INTEGER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Indicates whether future HTTP requests should support HTTP cookies (TRUE) or not (FALSE)</td>
</tr>
</tbody>
</table>
Table 265-29   (Cont.) GET_COOKIE_SUPPORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_cookies</td>
<td>Indicates the maximum total number of cookies maintained in the current session</td>
</tr>
<tr>
<td>max_cookies_per_site</td>
<td>Indicates the maximum number of cookies maintained in the current session for each Web site</td>
</tr>
</tbody>
</table>

265.9.14 GET_COOKIES Function

This function returns all the cookies maintained either in a request context or in the UTL_HTTP package's session state.

See Also:
HTTP Cookies and HTTP Cookies Subprograms

Syntax

```sql
UTL_HTTP.GET_COOKIES (  
    cookies          IN  OUT NOCOPY cookie_table,  
    request_context  IN             request_context_key DEFAULT NULL);
```

Parameters

Table 265-30   GET_COOKIES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookies</td>
<td>The cookies returned</td>
</tr>
<tr>
<td>request_context</td>
<td>Request context to return the cookies for. If NULL, the cookies maintained in the UTL_HTTP package's session state will be returned instead.</td>
</tr>
</tbody>
</table>

265.9.15 GET_DETAILED_EXCP_SUPPORT Procedure

This procedure checks if the UTL_HTTP package will raise a detailed exception or not.

See Also:
Session Settings and Session Settings Subprograms

Syntax

```sql
UTL_HTTP.GET_DETAILED_EXCP_SUPPORT (  
    enable  OUT BOOLEAN);
```
Parameters

Table 265-31  GET_DETAILED_EXCP_SUPPORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>TRUE if UTL_HTTP raises a detailed exception; otherwise FALSE</td>
</tr>
</tbody>
</table>

265.9.16 GET_DETAILED_SQLCODE Function

This function retrieves the detailed SQLCODE of the last exception raised.

See Also:
Error Conditions and Error Conditions Subprograms

Syntax

```
UTL_HTTP.GET_DETAILED_SQLCODE
RETURN PLS_INTEGER;
```

265.9.17 GET_DETAILED_SQLERRM Function

This function retrieves the detailed SQLERRM of the last exception raised.

See Also:
Error Conditions and Error Conditions Subprograms

Syntax

```
UTL_HTTP.GET_DETAILED_SQLERRM
RETURN VARCHAR2;
```

265.9.18 GET_FOLLOW_REDIRECT Procedure

This procedure retrieves the follow-redirect setting in the current session

See Also:
Session Settings and Session Settings Subprograms

Syntax

```
UTL_HTTP.GET_FOLLOW_REDIRECT (  
   max_redirects OUT PLS_INTEGER);
```
265.9.19 GET_HEADER Procedure

This procedure returns the n\textsuperscript{th} HTTP response header name and value returned in the response.

\textbf{See Also:}

- HTTP Responses
- HTTP Responses Subprograms

\textbf{Syntax}

\begin{verbatim}
UTL_HTTP.GET_HEADER ( \
    r      IN OUT NOCOPY resp, \
    n      IN PLS_INTEGER, \
    name   OUT NOCOPY VARCHAR2, \
    value  OUT NOCOPY VARCHAR2);
\end{verbatim}

\textbf{Parameters}

\textbf{Table 265-33  GET_HEADER Procedure Parameters}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
<tr>
<td>n</td>
<td>The n\textsuperscript{th} header to return</td>
</tr>
<tr>
<td>name</td>
<td>The name of the HTTP response header</td>
</tr>
<tr>
<td>value</td>
<td>The value of the HTTP response header</td>
</tr>
</tbody>
</table>

\textbf{Usage Notes}

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.
265.9.20 GET_HEADER_BY_NAME Procedure

This procedure returns the HTTP response header value returned in the response given the name of the header.

See Also:
HTTP Responses and HTTP Responses Subprograms

Syntax

UTL_HTTP.GET_HEADER_BY_NAME(
    r      IN OUT NOCOPY resp,
    name   IN VARCHAR2,
    value  OUT NOCOPY VARCHAR2,
    n      IN PLS_INTEGER DEFAULT 1);

Parameters

Table 265-34  GET_HEADER_BY_NAME Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
<tr>
<td>name</td>
<td>The name of the HTTP response header for which the value is to return</td>
</tr>
<tr>
<td>value</td>
<td>The value of the HTTP response header</td>
</tr>
<tr>
<td>n</td>
<td>The n\textsuperscript{th} occurrence of an HTTP response header by the specified name to return. The default is 1.</td>
</tr>
</tbody>
</table>

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

265.9.21 GET_HEADER_COUNT Function

This function returns the number of HTTP response headers returned in the response.

See Also:
HTTP Responses and HTTP Responses Subprograms
Syntax

UTL_HTTP.GET_HEADER_COUNT ( 
    r  IN OUT NOCOPY resp) 
RETURN PLS_INTEGER;

Parameters

Table 265-35  GET_HEADER_COUNT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
</tbody>
</table>

Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

265.9.22 GET_PERSISTENT_CONN_COUNT Function

This function returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers.

See Also:

HTTP Persistent Connections and HTTP Persistent Connections Subprograms

Syntax

UTL_HTTP.GET_PERSISTENT_CONN_COUNT 
RETURN PLS_INTEGER;

Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.
265.9.23 GET_PERSISTENT_CONN_SUPPORT Procedure

This procedure checks if the persistent connection support is enabled, and gets the maximum number of persistent connections in the current session.

See Also:
Session Settings and Session Settings Subprograms

Syntax

```
UTL_HTTP.GET_PERSISTENT_CONN_SUPPORT (    enable OUT BOOLEAN,    max_conns OUT PLS_INTEGER);
```

Parameters

Table 265-36  GET_PERSISTENT_CONN_SUPPORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>TRUE if persistent connection support is enabled; otherwise FALSE</td>
</tr>
<tr>
<td>max_conns</td>
<td>the maximum number of persistent connections maintained in the current session</td>
</tr>
</tbody>
</table>

265.9.24 GET_PERSISTENT_CONNS Procedure

This procedure returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers.

See Also:
HTTP Persistent Connections and HTTP Persistent Connections Subprograms

Syntax

```
UTL_HTTP.get_persistent_conns (    connections IN OUT NOCOPY connection_table);
```

Parameters

Table 265-37  GET_PERSISTENT_CONNS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connections</td>
<td>The network connections kept persistent</td>
</tr>
</tbody>
</table>
Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

265.9.25 GET_PROXY Procedure

This procedure retrieves the current proxy settings.

See Also:

Session Settings and Session Settings Subprograms

Syntax

```
UTL_HTTP.GET_PROXY (
    proxy             OUT NOCOPY VARCHAR2,
    no_proxy_domains  OUT NOCOPY VARCHAR2);
```

Parameters

Table 265-38   GET_PROXY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>The proxy (host and an optional port number) currently used by the UTL_HTTP package</td>
</tr>
<tr>
<td>no_proxy_domains</td>
<td>The list of hosts and domains for which no proxy is used for all requests</td>
</tr>
</tbody>
</table>

265.9.26 GET_RESPONSE Function

This function reads the HTTP response.

When the function returns, the status line and the HTTP response headers have been read and processed. The status code, reason phrase, and the HTTP protocol version are stored in the response record. This function completes the HTTP headers section.

See Also:

HTTP Responses and HTTP Responses Subprograms

Syntax

```
UTL_HTTP.GET_RESPONSE (
    r                       IN OUT NOCOPY req,
```

Chapter 265

Summary of UTL_HTTP Subprograms

265-46
return_info_response  IN BOOLEAN DEFAULT FALSE)
RETURN resp;

Parameters

Table 265-39  GET_RESPONSE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
<tr>
<td>return_info_response</td>
<td>Return 100 informational response or not.</td>
</tr>
<tr>
<td></td>
<td>• TRUE means get_response should return 100 informational response when it is received from the HTTP server. The request will not be ended if a 100 response is returned.</td>
</tr>
<tr>
<td></td>
<td>• FALSE means the API should ignore any 100 informational response received from the HTTP server and should return the following non-100 response instead. The default is FALSE.</td>
</tr>
</tbody>
</table>

Exceptions

• When detailed-exception is disabled:
  ORA-29273_REQUEST_FAILED - the request fails to execute. Use the GET_DETAILED_EXCP_SUPPORT Procedure and the GET_DETAILED_SQLERRM Function to get the detailed error message.
  
• When detailed-exception is enabled:
  ORA-29261_BAD_ARGUMENT - some arguments passed are not valid
  
• When response error check is enabled:
  ORA-29268_HTTP_CLIENT_ERROR - the response code is in 400 range
  ORA-29269_HTTP_SERVER_ERROR - the response code is in 500 range

Usage Notes

• The request will be ended when this functions returns regardless of whether an exception is raised or not. There is no need to invoke the END_REQUEST Procedure.

• If URL redirection occurs, the URL and method fields in the req record will be updated to the last redirected URL and the method used to access the URL.

Examples

In certain situations (initiated by the HTTP client or not), the HTTP server may return a 1xx informational response. The user who does not expect such a response may indicate to GET_RESPONSE to ignore the response and proceed to receive the regular response. In the case when the user expects such a response, the user can indicate to GET_RESPONSE to return the response.

For example, when a user is issuing a HTTP POST request with a large request body, the user may want to check with the HTTP server to ensure that the server will accept the request before sending the data. To do so, the user will send the additional EXPECT: 100-CONTINUE request header, and check for 100 CONTINUE response from the server before proceeding to send the request body. Then, the user will get the regular HTTP response.

The following code example illustrates this:
DECLARE
  data  VARCHAR2(1024) := '...';
  req   utl_http.req;
  resp  utl_http.resp;
BEGIN

  req := utl_http.begin_request('http://www.acme.com/receiver', 'POST');
  utl_http.set_header(req, 'Content-Length', length(data));
  -- Ask HTTP server to return "100 Continue" response
  utl_http.set_header(req, 'Expect', '100-continue');
  resp := utl_http.get_response(req, TRUE);

  -- Check for and dispose "100 Continue" response
  IF (resp.status_code <> 100) THEN
    utl_http.end_response(resp);
    raise_application_error(20000, 'Request rejected');
  END IF;
  utl_http.end_response(resp);

  -- Now, send the request body
  utl_http.write_text(req, data);

  -- Get the regular response
  resp := utl_http.get_response(req);
  utl_http.read_text(resp, data);
  utl_http.end_response(resp);
END;

265.9.27 GET_RESPONSE_ERROR_CHECK Procedure

This procedure checks if the response error check is set or not.

See Also:
Session Settings and Session Settings Subprograms

Syntax

UTL_HTTP.GET_RESPONSE_ERROR_CHECK (enable  OUT BOOLEAN);

Parameters

Table 265-40  GET_RESPONSE_ERROR_CHECK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>TRUE if the response error check is set; otherwise FALSE</td>
</tr>
</tbody>
</table>
265.9.28 GET_TRANSFER_TIMEOUT Procedure

This procedure retrieves the default timeout value for all future HTTP requests.

See Also:
Session Settings and Session Settings Subprograms

Syntax

```
UTL_HTTP.GET_TRANSFER_TIMEOUT (  
    timeout  OUT PLS_INTEGER);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The network transfer timeout value in seconds</td>
</tr>
</tbody>
</table>

265.9.29 READ_LINE Procedure

This procedure reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer.

The end of line is as defined in the function `read_line` of UTL_TCP. The end_of_body exception will be raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

See Also:
HTTP Responses and HTTP Responses Subprograms

Syntax

```
UTL_HTTP.READ_LINE(  
    r            IN OUT NOCOPY resp,  
    data         OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,  
    remove_crlf  IN  BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
</tbody>
</table>
Table 265-42  (Cont.) READ_LINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The HTTP response body in text form</td>
</tr>
<tr>
<td>remove_crlf</td>
<td>Removes the newline characters if set to TRUE</td>
</tr>
</tbody>
</table>

Usage Notes

The UTL_HTTP package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, read_line waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, read_line stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the partial_multibyte_char exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the read_raw procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the transfer_timeout exception is raised instead. The exception can be handled and the read operation can be retried later.

When the Content-Type response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "ORA-01482: unsupported character set" exception is raised if you try to read the response body as text. You can either read the response body as binary using the READ_RAW procedure, or set the character set of the response body explicitly using the SET_BODY_CHARSET procedure and read the response body as text again.

265.9.30 READ_RAW Procedure

This procedure reads the HTTP response body in binary form and returns the output in the caller-supplied buffer.

The end_of_body exception is raised if the end of the HTTP response body is reached.

See Also:

HTTP Responses and HTTP Responses Subprograms

Syntax

UTL_HTTP.READ_RAW(
    r IN OUT NOCOPY resp,
Parameters

Table 265-43  READ_RAW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response</td>
</tr>
<tr>
<td>data</td>
<td>The HTTP response body in binary form</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes of data to read. If len is NULL, this procedure will read as much input as possible to fill the buffer allocated in data. The actual amount of data returned may be less than that specified if not much data is available before the end of the HTTP response body is reached or the transfer_timeout amount of time has elapsed. The default is NULL</td>
</tr>
</tbody>
</table>

Usage Notes

The UTL_HTTP package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, read_raw waits for each data packet to be ready to read until timeout occurs. If it occurs, read_raw stops reading and returns all the data read successfully. If no data is read successfully, the transfer_timeout exception is raised. The exception can be handled and the read operation can be retried later.

265.9.31 READ_TEXT Procedure

This procedure reads the HTTP response body in text form and returns the output in the caller-supplied buffer.

The end_of_body exception is raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

See Also:

HTTP Responses and HTTP Responses Subprograms

Syntax

UTL_HTTP.READ_TEXT(
    r     IN OUT NOCOPY resp,
    data  OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
    len   IN PLS_INTEGER DEFAULT NULL);
Parameters

Table 265-44  READ_TEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>The HTTP response data</td>
</tr>
<tr>
<td><code>data</code></td>
<td>The HTTP response body in text form</td>
</tr>
<tr>
<td><code>len</code></td>
<td>The maximum number of characters of data to read. If <code>len</code> is NULL, this procedure will read as much input as possible to fill the buffer allocated in <code>data</code>. The actual amount of data returned may be less than that specified if little data is available before the end of the HTTP response body is reached or the <code>transfer_timeout</code> amount of time has elapsed. The default is NULL.</td>
</tr>
</tbody>
</table>

Usage Notes

The `UTL_HTTP` package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, `read_text` waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, `read_text` stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `read_raw` procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

When the Content-Type response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "ORA-01482: unsupported character set" exception is raised if you try to read the response body as text. You can either read the response body as binary using the `READ_RAW` procedure, or set the character set of the response body explicitly using the `SET_BODY_CHARSET` procedure and read the response body as text again.

265.9.32 REQUEST Function

This function returns up to the first 2000 bytes of data retrieved from the given URL.

This function can be used directly in SQL queries. The URL may contain the username and password needed to authenticate the request to the server. The format is `scheme://[user[:password]@]host[:port]/...`
You can define a username/password for the proxy to be specified in the proxy string. The format is

'http://[user[:password]@[host[:port]//

See Also:
Simple HTTP Fetches and Simple HTTP Fetches in a Single Call Subprograms

Syntax

UTL_HTTP.REQUEST (  
  url              IN VARCHAR2,  
  proxy            IN VARCHAR2 DEFAULT NULL,  
  wallet_path      IN VARCHAR2 DEFAULT NULL,  
  wallet_password  IN VARCHAR2 DEFAULT NULL,  
  https_host       IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;

Pragmas
pragma restrict_references (request, wnds, rnds, wnps, rnps);

Parameters

Table 265-45  REQUEST Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Uniform resource locator</td>
</tr>
<tr>
<td>proxy</td>
<td>(Optional) Specifies a proxy server to use when making the HTTP request. See SET_PROXY for the full format of the proxy setting.</td>
</tr>
<tr>
<td>wallet_path</td>
<td>(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request. The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\username\WALLETS, and in Unix is, for example, file:/home/username/wallets.</td>
</tr>
<tr>
<td>wallet_password</td>
<td>(Optional) Specifies the password required to open the wallet</td>
</tr>
<tr>
<td>https_host</td>
<td>A string representing the host name. If the string does not begin with a wildcard, the string will be used as the host name for server name indication (SNI). If the string begins with a wildcard, the string will be used to match against the common name (CN) of the remote server’s certificate for an HTTPS request. If NULL, the host name in the given URL will be used for SNI.</td>
</tr>
</tbody>
</table>
Return Values

The return type is a string of length 2000 or less, which contains up to the first 2000 bytes of the HTML result returned from the HTTP request to the argument URL.

Exceptions

INIT_FAILED
REQUEST_FAILED

Usage Notes

The URL passed as an argument to this function is not examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Please see the documentation of the function SET_WALLET on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server:

```html
<html>
<head><title>Error Message</title></head>
<body>
<H1>Fatal Error 500</H1>
Reason: Can't locate remote host: home.nothing.comm.<p>
</body>
</html>
```

Examples

SQL> SELECT UTL_HTTP.REQUEST('http://www.my-company.com/') FROM DUAL;
UTL_HTTP.REQUEST('HTTP://WWW.MY-COMPANY.COM/')

If you are behind a firewall, include the proxy parameter. For example, from within the Oracle firewall, where there might be a proxy server named www-proxy.my-company.com:

```sql
```
265.9.33 REQUEST_PIECES Function

This function returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL.

You can define a username/password for the proxy to be specified in the proxy string. The format is

```
[http://][user[:password]@]host[:port][/]
```

See Also:

Simple HTTP Fetches and Simple HTTP Fetches in a Single Call Subprograms

Syntax

```
TYPE html_pieces IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

UTL_HTTP.REQUEST_PIECES (  
  url             IN VARCHAR2,  
  max_pieces      IN NATURAL DEFAULT 32767,  
  proxy           IN VARCHAR2 DEFAULT NULL,  
  wallet_path     IN VARCHAR2 DEFAULT NULL,  
  wallet_password IN VARCHAR2 DEFAULT NULL,  
  https_host      IN VARCHAR2 DEFAULT NULL)  
RETURN html_pieces;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES (request_pieces, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 265-46  REQUEST_PIECES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Uniform resource locator</td>
</tr>
<tr>
<td>max_pieces</td>
<td>(Optional) The maximum number of pieces (each 2000 characters in length, except for the last, which may be shorter), that REQUEST_PIECES should return. If provided, then that argument should be a positive integer.</td>
</tr>
<tr>
<td>proxy</td>
<td>(Optional) Specifies a proxy server to use when making the HTTP request. See SET_PROXY for the full format of the proxy setting.</td>
</tr>
</tbody>
</table>
Table 265-46 (Cont.) REQUEST_PIECES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_path</td>
<td>(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request.</td>
</tr>
<tr>
<td></td>
<td>The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\username\WALLETS, and in Unix is, for example, file:/home/username/wallets. When the</td>
</tr>
<tr>
<td></td>
<td>UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible</td>
</tr>
<tr>
<td></td>
<td>from the database server. See SET_WALLET for the description on how to set up an Oracle wallet. Non-HTTPS requests do not require an Oracle wallet.</td>
</tr>
<tr>
<td>wallet_password</td>
<td>(Optional) Specifies the password required to open the wallet</td>
</tr>
<tr>
<td>https_host</td>
<td>A string representing the host name.</td>
</tr>
<tr>
<td></td>
<td>If the string does not begin with a wildcard, the string will be used as the host name for server name indication (SNI).</td>
</tr>
<tr>
<td></td>
<td>If the string begins with a wildcard, the string will be used to match against the common name (CN) of the remote server's certificate for an HTTPS</td>
</tr>
<tr>
<td></td>
<td>request.</td>
</tr>
<tr>
<td></td>
<td>If NULL, the host name in the given URL will be used for SNI.</td>
</tr>
</tbody>
</table>

Return Values

REQUEST_PIECES returns a PL/SQL table of type UTL_HTTP.HTML_PIECES. Each element of that PL/SQL table is a string of maximum length 2000. The elements of the PL/SQL table returned by REQUEST_PIECES are successive pieces of the data obtained from the HTTP request to that URL.

Exceptions

INIT_FAILED
REQUEST_FAILED

Usage Notes

The URL passed as an argument to this function will not be examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Each entry of the PL/SQL table (the "pieces") returned by this function may not be filled to their fullest capacity. The function may start filling the data in the next piece before the previous "piece" is totally full.

Please see the documentation of the function SET_WALLET on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server.
Examples

SET SERVEROUTPUT ON

DECLARE
  x UTL_HTTP.HTML_PIECES;
  len PLS_INTEGER;
BEGIN
  x := UTL_HTTP.REQUEST_PIECES('http://www.oracle.com/', 100);
  DBMS_OUTPUT.PUT_LINE(x.count || ' pieces were retrieved. ');
  DBMS_OUTPUT.PUT_LINE('with total length ');
  IF x.count < 1 THEN
    DBMS_OUTPUT.PUT_LINE('0');
  ELSE
    len := 0;
    FOR i in 1..x.count LOOP
      len := len + length(x(i));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(len);
  END IF;
END;
/
-- Output
Statement processed.
4 pieces were retrieved.
with total length
7687

265.9.34 SET_AUTHENTICATION Procedure

This procedure sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.

See Also:

HTTP Requests and HTTP Requests Subprograms

Syntax

UTL_HTTP.SET_AUTHENTICATION(
  r IN OUT NOCOPY req,
username IN VARCHAR2,
password IN VARCHAR2,
scheme IN VARCHAR2 DEFAULT 'Basic',
for_proxy IN BOOLEAN DEFAULT FALSE);

Parameters

Table 265-47  SET_AUTHENTICATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>HTTP request</td>
</tr>
<tr>
<td>username</td>
<td>Username for the HTTP authentication</td>
</tr>
<tr>
<td>password</td>
<td>Password for the HTTP authentication</td>
</tr>
<tr>
<td>scheme</td>
<td>HTTP authentication scheme. Either Basic for the HTTP basic or AWS for Amazon S3 authentication scheme. Default is basic.</td>
</tr>
<tr>
<td>for_proxy</td>
<td>Identifies if the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. Default is FALSE.</td>
</tr>
</tbody>
</table>

Usage Notes
The supported authentication schemes are HTTP basic and Amazon S3 authentication.

265.9.35 SET_AUTHENTICATION_FROM_WALLET Procedure

This procedure sets the HTTP authentication information in the HTTP request header needed for the request to be authorized by the Web server using the username and password credential stored in the Oracle wallet.

![See Also:](https://example.com)

External Password Store on , and HTTP Requests Subprograms

Syntax

```sql
UTL_HTTP.SET_AUTHENTICATION_FROM_WALLET(
    r IN OUT NOCOPY req,
    alias IN VARCHAR2,
    scheme IN VARCHAR2 DEFAULT 'Basic',
    for_proxy IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 265-48  SET_AUTHENTICATION_FROM_WALLET Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
</tbody>
</table>
Table 265-48  (Cont.) SET_AUTHENTICATION_FROM_WALLET Procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Alias to identify and retrieve the username and password credential stored in the Oracle wallet</td>
</tr>
<tr>
<td>scheme</td>
<td>HTTP authentication scheme. Either Basic for the HTTP basic or AWS for Amazon S3 authentication scheme. Default is basic.</td>
</tr>
<tr>
<td>for_proxy</td>
<td>Identifies if the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. Default is FALSE.</td>
</tr>
</tbody>
</table>

Usage Notes

- To use the password credentials in a wallet, the UTL_HTTP user must have the use-passwords privilege on the wallet.
- The supported authentication schemes are HTTP basic and Amazon S3 authentication schemes.

Examples

Creating a wallet and entering username and password in the wallet

```bash
> mkstore -wrl /oracle/wallets/test_wallet -create
Enter password: ******
Enter password again: ******
> mkstore -wrl /oracle/wallets/test_wallet -createCredential hr-access jsmith
Your secret/Password is missing in the command line
Enter your secret/Password: ****
Re-enter your secret/Password: ****
Enter wallet password: ******
```

Granting the use-passwords privilege on the wallet to a user by the database administrator

```sql
BEGIN
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
    acl         => 'wallet-acl.xml',
    description => 'Wallet ACL',
    principal   => 'SCOTT',
    is_grant    => TRUE,
    privilege   => 'use-passwords');
  DBMS_NETWORK_ACL_ADMIN.ASSIGN_WALLET_acl(
    acl         => 'wallet-acl.xml',
    wallet_path => 'file: /oracle/wallets/test_wallet');
END;
```

Using username and password from the wallet

```sql
DECLARE
  req  UTL_HTTP.req;
BEGIN
  UTL_HTTP.SET_WALLET(path => 'file:/oracle/wallets/test_wallet');
  req := UTL_HTTP.BEGIN_REQUEST(...);
  UTL_HTTP.SET_AUTHENTICATION_FROM_WALLET(req, 'hr-access');
  ...
END;
```
265.9.36 SET_BODY_CHARSET Procedures

This procedure is overloaded. The description of different functionality is located alongside the syntax declarations.

### See Also:
- HTTP Responses and HTTP Responses Subprograms
- Session Settings and Session Settings Subprograms

### Syntax

Sets the default character set of the body of all future HTTP requests when the media type is `text` and the character set is not specified in the `Content-Type` header.

Following the HTTP protocol standard specification, if the media type of a request or a response is `text`, but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to `ISO-8859-1`. A response created for a request inherits the default body character set of the request instead of the body character set of the current session. The default body character set is `ISO-8859-1` in a database user session. The default body character set setting affects only future requests and has no effect on existing requests. After a request is created, the body character set can be changed by using the other `SET_BODY_CHARSET` procedure that operates on a request:

```sql
UTL_HTTP.SET_BODY_CHARSET (charset IN VARCHAR2 DEFAULT NULL);
```

Sets the character set of the request body when the media type is `text` but the character set is not specified in the `Content-Type` header. According to the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to "ISO-8859-1". Use this procedure to change the default body character set a request inherits from the session default setting:

```sql
UTL_HTTP.SET_BODY_CHARSET(    r IN OUT NOCOPY req,    charset IN VARCHAR2 DEFAULT NULL);
```

Sets the character set of the response body when the media type is "text" but the character set is not specified in the `Content-Type` header. For each the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to "ISO-8859-1". Use this procedure to change the default body character set a response inherits from the request:

```sql
UTL_HTTP.SET_BODY_CHARSET(    r IN OUT NOCOPY resp,    charset IN VARCHAR2 DEFAULT NULL);
```
### Parameters

**Table 265-49  SET_BODY_CHARSET Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP response.</td>
</tr>
<tr>
<td>charset</td>
<td>The default character set of the response body. The character set can be in Oracle or Internet Assigned Numbers Authority (IANA) naming convention. If <code>charset</code> is <code>NULL</code>, the database character set is assumed.</td>
</tr>
</tbody>
</table>

---

### 265.9.37 SET_COOKIE_SUPPORT Procedures

This overloaded procedure handles cookie support. The description of different functionality is located alongside the syntax declarations.

#### See Also:

- HTTP Requests and HTTP Requests Subprograms
- Session Settings and Session Settings Subprograms

#### Syntax

Enables or disables support for the HTTP cookies in the request. Use this procedure to change the cookie support setting a request inherits from the session default setting:

```sql
UTL_HTTP.SET_COOKIE_SUPPORT(
    r       IN OUT NOCOPY REQ,
    enable  IN BOOLEAN DEFAULT TRUE);
```

Sets whether or not future HTTP requests will support HTTP cookies, and the maximum number of cookies maintained in the current database user session:

```sql
UTL_HTTP.SET_COOKIE_SUPPORT (
    enable       IN BOOLEAN,
    max_cookies  IN PLS_INTEGER DEFAULT 300,
    max_cookies_per_site  IN PLS_INTEGER DEFAULT 20);
```

#### Parameters

**Table 265-50  SET_COOKIE_SUPPORT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>enable</td>
<td>Set <code>enable</code> to <code>TRUE</code> to enable HTTP cookie support; <code>FALSE</code> to disable</td>
</tr>
<tr>
<td>max_cookies</td>
<td>Sets the maximum total number of cookies maintained in the current session</td>
</tr>
</tbody>
</table>
Table 265-50  (Cont.) SET_COOKIE_SUPPORT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_cookies_per_site</td>
<td>Sets the maximum number of cookies maintained in the current session for each Web site</td>
</tr>
</tbody>
</table>

Usage Notes

If cookie support is enabled for an HTTP request, all cookies saved in the current session and applicable to the request are returned to the Web server in the request in accordance with HTTP cookie specification standards. Cookies set in the response to the request are saved in the current session for return to the Web server in the subsequent requests if cookie support is enabled for those requests. If the cookie support is disabled for an HTTP request, no cookies are returned to the Web server in the request and the cookies set in the response to the request are not saved in the current session, although the Set-Cookie HTTP headers can still be retrieved from the response.

Cookie support is enabled by default for all HTTP requests in a database user session. The default setting of the cookie support (enabled versus disabled) affects only the future requests and has no effect on the existing ones. After your request is created, the cookie support setting may be changed by using the other SET_COOKIE_SUPPORT procedure that operates on a request.

The default maximum number of cookies saved in the current session is 20 for each site and 300 total.

If you lower the maximum total number of cookies or the maximum number of cookies for each Web site, the oldest cookies will be purged first to reduce the number of cookies to the lowered maximum. HTTP cookies saved in the current session last for the duration of the database session only; there is no persistent storage for the cookies. Cookies saved in the current session are not cleared if you disable cookie support.

See "Examples" for how to use GET_COOKIES and ADD_COOKIES to retrieve, save, and restore cookies.

265.9.38 SET_DETAILED_EXCP_SUPPORT Procedure

This procedure sets the UTL_HTTP package to raise a detailed exception.

By default, UTL_HTTP raises the request_failed exception when an HTTP request fails. Use GET_DETAILED_SQLCODE and GET_DETAILED_SQLEERM for more detailed information about the error.

See Also:

Session Settings and Session Settings Subprograms
Syntax

```sql
UTL_HTTP.SET_DETAILED_EXCP_SUPPORT (  
   enable IN BOOLEAN DEFAULT FALSE);
```

Parameters

**Table 265-51  SET_DETAILED_EXCP_SUPPORT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Asks UTL_HTTP to raise a detailed exception directly if set to TRUE; otherwise FALSE</td>
</tr>
</tbody>
</table>

### 265.9.39 SET_FOLLOW_REDIRECT Procedures

This procedure sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP response to this request, or future requests, in the `GET_RESPONSE` function.

**See Also:**
- HTTP Requests and HTTP Requests Subprograms
- Session Settings and Session Settings Subprograms

Syntax

Use this procedure to set the maximum number of redirections:

```sql
UTL_HTTP.SET_FOLLOW_REDIRECT (  
   max_redirects IN PLS_INTEGER DEFAULT 3);
```

Use this procedure to change the maximum number of redirections a request inherits from the session default setting:

```sql
UTL_HTTP.SET_FOLLOW_REDIRECT(  
   r              IN OUT NOCOPY req,  
   max_redirects IN PLS_INTEGER DEFAULT 3);
```

Parameters

**Table 265-52  SET_FOLLOW_REDIRECT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>max_redirects</td>
<td>The maximum number of redirects. Set to zero to disable redirects.</td>
</tr>
</tbody>
</table>
Usage Notes

If `max redirects` is set to a positive number, the `GET RESPONSE Function` will automatically follow the redirected URL for the HTTP response status code 301, 302, and 307 for the HTTP HEAD and GET methods, and 303 for all HTTP methods, and retry the HTTP request (the request method will be changed to HTTP GET for the status code 303) at the new location. It follows the redirection until the final, non-redirect location is reached, or an error occurs, or the maximum number of redirections has been reached (to prevent an infinite loop). The URL and method fields in the `REQ` record will be updated to the last redirected URL and the method used to access the URL. Set the maximum number of redirects to zero to disable automatic redirection.

While it is set not to follow redirect automatically in the current session, it is possible to specify individual HTTP requests to follow redirect instructions the function `FOLLOW_REDIRECT` and vice versa.

The default maximum number of redirections in a database user session is 3. The default value affects only future requests and has no effect on existing requests.

The `SET_FOLLOW_REDIRECT` procedure must be called before `GET_RESPONSE` for any redirection to take effect.

265.9.40 SET_HEADER Procedure

This procedure sets an HTTP request header. The request header is sent to the Web server as soon as it is set.

See Also:

HTTP Requests and HTTP Requests Subprograms

Syntax

```
UTL_HTTP.SET_HEADER (  
  r      IN OUT NOCOPY req,  
  name   IN VARCHAR2,  
  value  IN VARCHAR2);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>name</td>
<td>The name of the HTTP request header</td>
</tr>
<tr>
<td>value</td>
<td>The value of the HTTP request header</td>
</tr>
</tbody>
</table>
Usage Notes

Multiple HTTP headers with the same name are allowed in the HTTP protocol standard. Therefore, setting a header does not replace a prior header with the same name.

If the request is made using HTTP 1.1, `UTL_HTTP` sets the Host header automatically for you.

When you set the Content-Type header with this procedure, `UTL_HTTP` looks for the character set information in the header value. If the character set information is present, it is set as the character set of the request body. It can be overridden later by using the `SET_BODY_CHARSET` procedure.

When you set the Transfer-Encoding header with the value chunked, `UTL_HTTP` automatically encodes the request body written by the `WRITE_TEXT`, `WRITE_LINE` and `WRITE_RAW` procedures. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format.

265.9.41 SET_PERSISTENT_CONN_SUPPORT Procedure

This overloaded procedure provides persistent connection support. Descriptions of the different functionality are given in the syntax declarations.

Syntax

Sets whether future HTTP requests should support the HTTP 1.1 persistent connection or not, and the maximum numbers of persistent connections to be maintained in the current database user session.

```
UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(
    enable      IN BOOLEAN DEFAULT FALSE,
    max_conns   IN PLS_INTEGER DEFAULT 0);
```

Enables or disables support for the HTTP 1.1 persistent-connection in the request.

```
UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(
    r           IN OUT NOCOPY req,
    enable      IN BOOLEAN DEFAULT FALSE);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>TRUE to keep the network connection persistent. FALSE otherwise.</td>
</tr>
<tr>
<td>maximum_conns</td>
<td>Maximum number of connections</td>
</tr>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
</tbody>
</table>
Usage Notes

If the persistent-connection support is enabled for an HTTP request, the package will keep the network connections to a Web server or the proxy server open in the package after the request is completed properly for a subsequent request to the same server to reuse for each HTTP 1.1 protocol specification. With the persistent connection support, subsequent HTTP requests may be completed faster because the network connection latency is avoided. If the persistent-connection support is disabled for a request, the package will always send the HTTP header "Connection: close" automatically in the HTTP request and close the network connection when the request is completed. This setting has no effect on HTTP requests that follows HTTP 1.0 protocol, for which the network connections will always be closed after the requests are completed.

When a request is being made, the package attempts to reuse an existing persistent connection to the target Web server (or proxy server) if one is available. If none is available, a new network connection will be initiated. The persistent-connection support setting for a request affects only whether the network connection should be closed after a request completes.

Use this procedure to change the persistent-connection support setting a request inherits from the session default setting.

Users should note that while the use of persistent connections in UTL_HTTP may reduce the time it takes to fetch multiple Web pages from the same server, it consumes precious system resources (network connections) in the database server. Also, excessive use of persistent connections may reduce the scalability of the database server when too many network connections are kept open in the database server. Network connections should be kept open only if they will be used immediately by subsequent requests and should be closed immediately when they are no longer needed. Set the default persistent connection support as disabled in the session, and enable persistent connection in individual HTTP requests as shown in "Examples".

The default value of the maximum number of persistent connections in a database session is zero. To truly enable persistent connections, you must also set the maximum number of persistent connections to a positive value or no connections will be kept persistent.

Note that if you want to use persistent connections, you must call the overload that takes the maximum_conns parameter prior to calling the BEGIN_REQUEST Function, otherwise persistent connections will not be enabled for the current request even if the other form of SET_PERSISTENT_CONN_SUPPORT is called.

Examples

Using SET_PERSISTENT_CONN_SUPPORT in http requests at the session level, showing the active persistent connection after each request

```sql
DECLARE
  pieces utl_http.html_pieces;
  conns utl_http.connection_table;
BEGIN

  -- Turns on persistent connection support for the request_pieces call.
  utl_http.set_persistent_conn_support(true, 1);

  FOR i IN 1..10 LOOP

```

---

265-66
pieces := utl_http.request_pieces('http://www.example.com/');

-- Shows the active persistent connection
utl_http.get_persistent_conns(conns);
FOR j IN 1..conns.count LOOP
  dbms_output.put_line('Persistent connection '||j||': '||conns(j).host||':'||
    conns(j).port);
END LOOP;
END LOOP;

-- Turns off persistent connection support. Set active max persistent connection to 0 to close all active connections.
utl_http.set_persistent_conn_support(false, 0);
END;
/

Using SET_PERSISTENT_CONN_SUPPORT in HTTP requests showing how to use persistent connection individually in each request to fetch multiple URLs at the same host

DECLARE
  -- Table to store the URLs
  TYPE vc2_table IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
  paths VC2_TABLE;

PROCEDURE fetch_pages(paths IN vc2_table) AS
  req UTL_HTTP.REQ;
  resp UTL_HTTP.RESP;
  data VARCHAR2(1024);
BEGIN
  -- Set the proxy server
  UTL_HTTP.SET_PROXY('www-proxy.example.com:80', '');
  FOR i IN 1..paths.count LOOP
    req := UTL_HTTP.BEGIN_REQUEST(paths(i));
    -- Use persistent connections except for the last request
    IF (i < paths.count) THEN
      -- Use a persistent connection for the current request
      UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(req, TRUE);
    END IF;
    resp := UTL_HTTP.GET_RESPONSE(req); 
    -- Display the results of the response
    DBMS_OUTPUT.PUT_LINE('-');
    DBMS_OUTPUT.PUT_LINE('URL: ' || paths(i));
    DBMS_OUTPUT.PUT_LINE('HTTP Response Status Code: ' || resp.status_code);
    DBMS_OUTPUT.PUT_LINE('HTTP Response Reason Phrase: ' || resp.reason_phrase);
    DBMS_OUTPUT.PUT_LINE('HTTP Response Version: ' || resp.http_version);
    BEGIN
      LOOP
        UTL_HTTP.READ_TEXT(resp, data);
        -- do something with the data
      END LOOP;
  END LOOP;
END;
BEGIN
-- Set a maximum of 1 persistent connection, but start with persistent connections
-- off
UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(FALSE, 1);

-- Create a list of URLs
paths(1) := 'http://www.example.com/technetwork/index.html';
paths(2) := 'http://www.example.com/us/products/index.html';

fetch_pages(paths);
END;
/

265.9.42 SET_PROXY Procedure

This procedure sets the proxy to be used for requests of the HTTP or other protocols, excluding those for hosts that belong to the domain specified in no_proxy_domains.

no_proxy_domains is a comma-, semi-colon-, or space-separated list of domains or hosts for which HTTP requests should be sent directly to the destination HTTP server instead of going through a proxy server.

See Also:
- Session Settings and Session Settings Subprograms

Syntax

UTL_HTTP.SET_PROXY (proxy IN VARCHAR2,
                      no_proxy_domains IN VARCHAR2);

Parameters

Table 265-55 SET_PROXY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>The proxy (host and an optional port number) to be used by the UTL_HTTP package</td>
</tr>
<tr>
<td>no_proxy_domains</td>
<td>The list of hosts and domains for which no proxy should be used for all requests</td>
</tr>
</tbody>
</table>
Usage Notes

The proxy may include an optional TCP/IP port number at which the proxy server listens. The syntax is [http://]host[:port][/], for example, www-proxy.my-company.com:80. If the port is not specified for the proxy, port 80 is assumed.

Optionally, a port number can be specified for each domain or host. If the port number is specified, the no-proxy restriction is only applied to the request at the port of the particular domain or host, for example, corp.my-company.com, eng.my-company.com:80. When no_proxy_domains is NULL and the proxy is set, all requests go through the proxy. When the proxy is not set, UTL_HTTP sends requests to the target Web servers directly.

You can define a username/password for the proxy to be specified in the proxy string. The format is

[http://][user[:password]@]host[:port][/]

If proxy settings are set when the database server instance is started, the proxy settings in the environment variables http_proxy and no_proxy are assumed. Proxy settings set by this procedure override the initial settings.

265.9.43 SET_RESPONSE_ERROR_CHECK Procedure

This procedure sets whether or not GET_RESPONSE raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges.

For example, when the requested URL is not found in the destination Web server, a 404 (document not found) response status code is returned.

See Also:
Session Settings and Session Settings Subprograms

Syntax

UTL_HTTP.SET_RESPONSE_ERROR_CHECK (enable IN BOOLEAN DEFAULT FALSE);

Parameters

Table 265-56 SET_RESPONSE_ERROR_CHECK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>TRUE to check for response errors; otherwise FALSE</td>
</tr>
</tbody>
</table>

Usage Notes

If the status code indicates an error—a 4xx or 5xx code—and this procedure is enabled, GET_RESPONSE will raise the HTTP_CLIENT_ERROR or HTTP_SERVER_ERROR exception. If SET_RESPONSE_ERROR_CHECK is set to FALSE, GET_RESPONSE will not raise an exception when the status code indicates an error.
Response error check is turned off by default.

The `GET_RESPONSE` function can raise other exceptions when `SET_RESPONSE_ERROR_CHECK` is set to `FALSE`.

### 265.9.44 SET_TRANSFER_TIMEOUT Procedure

This procedure sets the default time out value for all future HTTP requests that the `UTL_HTTP` package should attempt while reading the HTTP response from the Web server or proxy server.

This timeout value may be used to avoid the PL/SQL programs from being blocked by busy Web servers or heavy network traffic while retrieving Web pages from the Web servers.

*See Also:*
- [Session Settings and Session Settings Subprograms](#)

#### Syntax

```plsql
UTL_HTTP.SET_TRANSFER_TIMEOUT (timeout IN PLS_INTEGER DEFAULT 60);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The network transfer timeout value in seconds.</td>
</tr>
</tbody>
</table>

#### Usage Notes

The default value of the timeout is 60 seconds.

### 265.9.45 SET_WALLET Procedure

This procedure sets the Oracle wallet used for all HTTP requests over Secured Socket Layer (SSL), namely HTTPS.

When the `UTL_HTTP` package communicates with an HTTP server over SSL, the HTTP server presents its digital certificate, which is signed by a certificate authority, to the `UTL_HTTP` package for identification purpose. The Oracle wallet contains the list of certificate authorities that are trusted by the user of the `UTL_HTTP` package. An Oracle wallet is required to make an HTTPS request.
See Also:

- Session Settings and Session Settings Subprograms
- Oracle Database Security Guide managing fine-grained access

Syntax

```sql
UTL_HTTP.SET_WALLET (
    path      IN VARCHAR2,
    password  IN VARCHAR2 DEFAULT NULL);
```

Parameters

**Table 265-58  SET_WALLET Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The directory path that contains the Oracle wallet. The format is <code>file:directory-path</code>. The format of wallet_path on a PC is, for example, <code>file:c:\WINNT\Profiles\username\WALLETS</code>, and in Unix is, for example, <code>file:/home/username/wallets</code>. When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server.</td>
</tr>
<tr>
<td>password</td>
<td>The password needed to open the wallet. If the wallet is auto-login enabled, the password may be omitted and should be set to NULL. See Oracle Database Advanced Security Guide for information about using Oracle Wallet Manager and the ORAPKI utility to create an auto-login wallet.</td>
</tr>
</tbody>
</table>

Usage Notes

To set up an Oracle wallet, use the Oracle Wallet Manager to create a wallet. In order for the HTTPS request to succeed, the certificate authority that signs the certificate of the remote HTTPS Web server must be a trust point set in the wallet.

When a wallet is created, it is populated with a set of well-known certificate authorities as trust points. If the certificate authority that signs the certificate of the remote HTTPS Web server is not among the trust points, or the certificate authority has new root certificates, you should obtain the root certificate of that certificate authority and install it as a trust point in the wallet using Oracle Wallet Manager

See Also:

Oracle Database Advanced Security Guide for more information on Wallet Manager
265.9.46 WRITE_LINE Procedure

This procedure writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP).

As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.

See Also:
HTTP Requests and HTTP Requests Subprograms

Syntax

UTL_HTTP.WRITE_LINE(
    r     IN OUT NOCOPY req,
    data  IN VARCHAR2 CHARACTER SET ANY_CS);

Parameters

Table 265-59  WRITE_LINE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>data</td>
<td>The text line to send in the HTTP request body</td>
</tr>
</tbody>
</table>

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. The UTL_HTTP package performs chunked transfer-encoding on the request body transparently when the Transfer-Encoding: chunked header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the SET_HEADER procedure for details.

If you send the Content-Length header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the Content-Length header, and the results using the WRITE_RAW procedure to avoid the automatic character set conversion. Or, if the remove Web server or CGI programs allow, you can send the request body using the
HTTP 1.1 chunked transfer-encoding format, where UTL_HTTP handles the length of the chunks transparently.

### 265.9.47 WRITE_RAW Procedure

This procedure writes some binary data in the HTTP request body. As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed.

**See Also:**

HTTP Requests and HTTP Requests Subprograms

**Syntax**

```
UTL_HTTP.WRITE_RAW(
  r IN OUT NOCOPY REQ,
  data IN RAW);
```

**Parameters**

**Table 265-60  WRITE_RAW Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>data</td>
<td>The binary data to send in the HTTP request body</td>
</tr>
</tbody>
</table>

**Usage Notes**

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the `Content-Length` header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. UTL_HTTP performs chunked transfer-encoding on the request body transparently when the `Transfer-Encoding:chunked` header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the `SET_HEADER` procedure for details.

### 265.9.48 WRITE_TEXT Procedure

This procedure writes some text data in the HTTP request body.

As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.
Syntax

UTL_HTTP.WRITE_TEXT(
    r     IN OUT NOCOPY REQ,
    data  IN            VARCHAR2 CHARACTER SET ANY_CS);

Parameters

Table 265-61  WRITE_TEXT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The HTTP request</td>
</tr>
<tr>
<td>data</td>
<td>The text data to send in the HTTP request body</td>
</tr>
</tbody>
</table>

Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the Content-Length header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. UTL_HTTP performs chunked transfer-encoding on the request body transparently when the Transfer-Encoding: chunked header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the SET_HEADER procedure for details.

If you send the Content-Length header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the Content-Length header, and the results using the WRITE_RAW procedure to avoid the automatic character set conversion. Or, if the remove Web server or CGI programs allow, you can send the request body using the HTTP 1.1 chunked transfer-encoding format, where UTL_HTTP handles the length of the chunks transparently.

See Also:

HTTP Requests and HTTP Requests Subprograms
UTL_I18N is a set of services that provides additional globalization functionality for applications written in PL/SQL.

### See Also:

*Oracle Database Globalization Support Guide*

The chapter contains the following topics:

- Overview
- Security Model
- Constants
- Summary of UTL_I18N Subprograms

### 266.1 UTL_I18N Overview

UTL_I18N services provide additional globalization functionality for applications written in PL/SQL.

The UTL_I18N PL/SQL package consists of the following categories of services:

- String conversion functions for various datatypes.
- Functions that convert a text string to character references and vice versa.
- Functions that map between Oracle, Java, and ISO languages and territories.
- Functions that map between Oracle, Internet Assigned Numbers Authority (IANA), and e-mail safe character sets.
- A function that returns the Oracle character set name from an Oracle language name.
- A function that returns the maximum number of bytes for a character of an Oracle character set.
- A function that performs script transliteration.
- Functions that return the ISO currency code, local time zones, and local languages supported for a given territory.
- Functions that return the most appropriate linguistic sort, a listing of all the applicable linguistic sorts, and the local territories supported for a given language.
- Functions that map between the Oracle full and short language names.
- A function that returns the language translation of a given language and territory name.
- A function that returns a listing of the most commonly used time zones.
266.2 UTL_I18N Security Model

The functions of the UTL_I18N package neither read database contents nor modify them. The functions operate on their arguments only and/or they retrieve static internationalization information from NLS Data files. The execution privilege for the package is granted to PUBLIC by default.

266.3 UTL_I18N Constants

UTL_I18N defines constants to use when specifying parameter values.

These constants are shown in the following table.

Table 266-1   UTL_I18N Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC_CONTEXT</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>Returns the default character set for general cases.</td>
</tr>
<tr>
<td>MAIL_GENERIC</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>Map from an Oracle character set name to an email safe character set name on a non-Windows platform.</td>
</tr>
<tr>
<td>ORACLE_TO_IANA</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>Map from an Oracle character set name to an IANA character set name.</td>
</tr>
<tr>
<td>SHIFT_IN</td>
<td>PLS_INTEGER</td>
<td>0</td>
<td>Used with shift_status. Must be set the first time it is called in piecewise conversion.</td>
</tr>
<tr>
<td>IANA_TO_ORACLE</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Map from an IANA character set name to an Oracle character set name.</td>
</tr>
<tr>
<td>MAIL_CONTEXT</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>The mapping is between an Oracle character set name and an email safe character set name.</td>
</tr>
<tr>
<td>MAIL_WINDOWS</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td>Map from an Oracle character set name to an email safe character set name on a Windows platform.</td>
</tr>
<tr>
<td>SHIFT_OUT</td>
<td>PLS_INTEGER</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>FWKATAKANA_HIRA</td>
<td>VARCHAR2(30)</td>
<td>'fwkatakana_hiragana'</td>
<td>Converts only fullwidth Katakana characters to fullwidth Hiragana characters.</td>
</tr>
<tr>
<td>FWKATAKANA_HWKA</td>
<td>VARCHAR2(30)</td>
<td>'fwkatakana_hwkatakana'</td>
<td>Converts only fullwidth Katakana characters to halfwidth Katakana characters.</td>
</tr>
<tr>
<td>HIRAGANA_FWKATA</td>
<td>VARCHAR2(30)</td>
<td>'hiragana_fwkatakana'</td>
<td>Converts only fullwidth Hiragana characters to fullwidth Katakana characters.</td>
</tr>
<tr>
<td>HIRAGANA_HWKATA</td>
<td>VARCHAR2(30)</td>
<td>'hiragana_hwkatakana'</td>
<td>Converts only fullwidth Hiragana characters to halfwidth Katakana characters.</td>
</tr>
</tbody>
</table>
### Table 266-1 (Cont.) UTL_I18N Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWKATAKANA_FWKA</td>
<td>VARCHAR2(30)</td>
<td>'hwkatakana-fwkatakana'</td>
<td>Converts only halfwidth Katakana characters to fullwidth Katakana characters.</td>
</tr>
<tr>
<td>HWKATAKANA_HIRA</td>
<td>VARCHAR2(30)</td>
<td>'hwkatakana_hiragana'</td>
<td>Converts only halfwidth Katakana characters to fullwidth Hiragana characters.</td>
</tr>
<tr>
<td>KANA_FWKATAKANA</td>
<td>VARCHAR2(30)</td>
<td>'kana_fwkatakana'</td>
<td>Converts any type of Kana character to a fullwidth Katakana character.</td>
</tr>
<tr>
<td>KANA_HIRAGANA</td>
<td>VARCHAR2(30)</td>
<td>'kana_hiragana'</td>
<td>Converts any type of Kana character to a fullwidth Hiragana character.</td>
</tr>
<tr>
<td>KANA_HWKATAKANA</td>
<td>VARCHAR2(30)</td>
<td>'kana_hwkatakana'</td>
<td>Converts any type of Kana character to a halfwidth Katakana character.</td>
</tr>
</tbody>
</table>

#### 266.4 Summary of UTL_I18N Subprograms

This table lists the UTL_I18N subprograms and briefly describes them.

### Table 266-2 UTL_I18N Package Subprograms

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESCAPE_REFERENCE Function</td>
<td>Converts a given text string to its character reference counterparts, for characters that fall outside the document character set.</td>
</tr>
<tr>
<td>GET_COMMON_TIME_ZONES Function</td>
<td>Returns the list of common time zone IDs that are independent of the locales.</td>
</tr>
<tr>
<td>GET_DEFAULT_CHARSET Function</td>
<td>Returns the default Oracle character set name or the default e-mail safe character set name from an Oracle language name.</td>
</tr>
<tr>
<td>GET_DEFAULT_ISO_CURRENCY Function</td>
<td>Returns the default ISO 4217 currency code for the specified territory.</td>
</tr>
<tr>
<td>GET_DEFAULT_LINGUISTIC_SORT Function</td>
<td>Returns the default linguistic sort name for the specified language.</td>
</tr>
<tr>
<td>GET_LOCAL_LANGUAGES Function</td>
<td>Returns the local language names for the specified territory.</td>
</tr>
<tr>
<td>GET_LOCAL_LINGUISTIC_SORTS Function</td>
<td>Returns the local linguistic sort names for the specified language.</td>
</tr>
<tr>
<td>GET_LOCAL_TERRITORIES Function</td>
<td>Returns the local territory names for the specified language.</td>
</tr>
<tr>
<td>GET_LOCAL_TIME_ZONES Function</td>
<td>Returns the local time zone IDs for the specified territory.</td>
</tr>
<tr>
<td>GET_MAX_CHARACTER_SIZE Function</td>
<td>Returns the maximum character size of a given character set.</td>
</tr>
<tr>
<td>GET_TRANSLATION Function</td>
<td>Returns the translation of the language and territory name in the specified translation language.</td>
</tr>
</tbody>
</table>
Table 266-2  (Cont.) UTL_I18N Package Subprograms

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP_CHARSET Function</td>
<td>· Maps an Oracle character set name to an IANA character set name.</td>
</tr>
<tr>
<td></td>
<td>· Maps an IANA character set name to an Oracle character set name.</td>
</tr>
<tr>
<td></td>
<td>· Maps an Oracle character set name to an e-mail safe character set name.</td>
</tr>
<tr>
<td>MAP_FROM_SHORT_LANGUAGE Function</td>
<td>Maps an Oracle short language name to an Oracle language name.</td>
</tr>
<tr>
<td>MAP_LANGUAGE_FROM_ISO Function</td>
<td>Returns an Oracle language name from an ISO locale name.</td>
</tr>
<tr>
<td>MAP_LOCALE_TO_ISO Function</td>
<td>Returns an ISO locale name from the Oracle language and territory name.</td>
</tr>
<tr>
<td>MAP_TERRITORY_FROM_ISO Function</td>
<td>Returns an Oracle territory name from an ISO locale name.</td>
</tr>
<tr>
<td>MAP_TO_SHORT_LANGUAGE Function</td>
<td>Maps an Oracle language name to an Oracle short language name.</td>
</tr>
<tr>
<td>RAW_TO_CHAR Functions</td>
<td>Converts RAW data that is not encoded in the database character set into a VARCHAR2 string</td>
</tr>
<tr>
<td>RAW_TO_NCHAR Functions</td>
<td>Converts RAW data that is not encoded in the national character set into an NVARCHAR2 string</td>
</tr>
<tr>
<td>STRING_TO_RAW Function</td>
<td>Converts a VARCHAR2 or NVARCHAR2 string to another character set. The result is returned as a RAW datatype.</td>
</tr>
<tr>
<td>TRANSLITERATE Function</td>
<td>Transliterates between Japanese hiragana and katakana.</td>
</tr>
<tr>
<td>UNESCAPE_REFERENCE Function</td>
<td>Converts an input string that contains character references to a text string.</td>
</tr>
<tr>
<td>VALIDATE_CHARACTER_ENCODING Function</td>
<td>Validates the character encoding of VARCHAR2, NVARCHAR2, CLOB, and NCLOB data.</td>
</tr>
</tbody>
</table>

266.4.1 ESCAPE_REFERENCE Function

This function converts a text string to its character reference counterparts for characters that fall outside the character set used by the current document.

Character references are mainly used in HTML and XML documents to represent characters independently of the encoding of the document. Character references may appear in two forms, numeric character references and character entity references. Numeric character references specify the Unicode code point value of a character, while character entity references use symbolic names to refer to the same character. For example, \&#xe5; is the numeric character reference for the small letter “a” with a ring above, whereas \&aring; is the character entity reference for the same character. Character entity references are also used to escape special characters, as an example, \&lt; represents the < (less than) sign. This is to avoid possible confusion with the beginning of a tag in Markup languages.

Syntax

```sql
UTL_I18N.ESCAPE_REFERENCE(
    str            IN VARCHAR2 CHARACTER SET ANY_CS,
)```
PARAMETERS

Table 266-3  ESCAPE_REFERENCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Specifies the input string</td>
</tr>
<tr>
<td>page_cs_name</td>
<td>Specifies the character set of the document. If page_cs_name is NULL, then</td>
</tr>
<tr>
<td></td>
<td>the database character set is used for CHAR data and the national character</td>
</tr>
<tr>
<td></td>
<td>set is used for NCHAR data.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid character set or a NULL string, then the function returns a NULL string.

Examples

UTL_I18N.ESCAPE_REFERENCE('hello &lt; '||chr(229),'us7ascii')

This returns 'hello &lt; å'.

266.4.2 GET_COMMON_TIME_ZONES Function

This function returns a listing of the most commonly used time zones. This list contains a subset of the time zones that are supported in the database.

Syntax

UTL_I18N.GET_COMMON_TIME_ZONES
RETURN STRING_ARRAY;

Examples

Returns the list of the most commonly used time zones.

DECLARE
retval UTL_I18N.STRING_ARRAY;
BEGIN
retval := UTL_I18N.GET_COMMON_TIME_ZONES;
END;
/

266.4.3 GET_DEFAULT_CHARSET Function

This function returns the default Oracle character set name or the default e-mail safe character set name from an Oracle language name.

See Also:

"MAP_CHARSET Function" for an explanation of an e-mail safe character set
Syntax

```sql
UTL_I18N.GET_DEFAULT_CHARSET(
    language IN VARCHAR2,
    context IN PLS_INTEGER DEFAULT GENERIC_CONTEXT,
    iswindows IN BOOLEAN DEFAULT FALSE)
RETURN VARCHAR2;
```

Parameters

Table 266-4  GET_DEFAULT_CHARSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>Specifies a valid Oracle language</td>
</tr>
</tbody>
</table>
| context   | GENERIC_CONTEXT | MAIL_CONTEXT
|           | GENERIC_CONTEXT: Returns the default character set for general cases |
|           | MAIL_CONTEXT: Returns the default e-mail safe character set name |
| iswindows | If context is set as MAIL_CONTEXT, then iswindows should be set to TRUE if the platform is Windows and FALSE if the platform is not Windows. The default is FALSE. |
|           | iswindows has no effect if context is set as GENERIC_CONTEXT. |

Usage Notes

If the user specifies an invalid language name or an invalid flag, then the function returns a NULL string.

Examples

**GENERIC_CONTEXT, iswindows=FALSE**

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.GENERIC_CONTEXT, FALSE)
```

This returns 'WE8ISO8859P1'.

**MAIL_CONTEXT, iswindows=TRUE**

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.MAIL_CONTEXT, TRUE)
```

This returns 'WE8MSWIN1252'.

**MAIL_CONTEXT, iswindows=FALSE**

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.MAIL_CONTEXT, FALSE)
```

This returns 'WE8ISO8859P1'.

266.4.4 GET_DEFAULT_ISO_CURRENCY Function

This function returns the default ISO 4217 currency code for the specified territory.

Syntax

```sql
UTL_I18N.GET_DEFAULT_ISO_CURRENCY (    territory    IN VARCHAR2 CHARACTER SET ANY_CS)    RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>territory</td>
<td>Specifies a valid Oracle territory. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

Examples

Displays the default ISO currency code for China.

```sql
DECLARE    retval VARCHAR2(50);    BEGIN    retval := UTL_I18N.GET_DEFAULT_ISO_CURRENCY('CHINA');    DBMS_OUTPUT.PUT_LINE(retval);    END;
/```

266.4.5 GET_DEFAULT_LINGUISTIC_SORT Function

This function returns the most commonly used Oracle linguistic sort for the specified language.

Syntax

```sql
UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT (    language    IN VARCHAR2 CHARACTER SET ANY_CS)    RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>Specifies a valid Oracle language. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.
Examples
Displays the name of the most appropriate linguistic sort name for the language used in the current SQL session.

```sql
DECLARE
    retval VARCHAR2(50);
BEGIN
    SELECT value INTO retval FROM nls_session_parameters
    WHERE parameter = 'NLS_LANGUAGE';
    retval := UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT(retval);
    DBMS_OUTPUT.PUT_LINE(retval);
END;
/
```

266.4.6 GET_LOCAL_LANGUAGES Function

This function returns the local language names for the specified territory.

Syntax

```sql
UTL_I18N.GET_LOCAL_LANGUAGES (  
    territory    IN VARCHAR2 CHARACTER SET ANY_CS)  
RETURN STRING_ARRAY;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>territory</td>
<td>Specifies a valid Oracle territory. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

Examples

Returns the list of local languages used in Belgium.

```sql
DECLARE
    retval UTL_I18N.STRING_ARRAY;
    cnt INTEGER;
BEGIN
    retval := UTL_I18N.GET_LOCAL_LANGUAGES('BELGIUM');
    DBMS_OUTPUT.PUT('Count = ');  
    DBMS_OUTPUT.PUT_LINE(retval.LAST);
    cnt := retval.FIRST;
    WHILE cnt IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(retval(cnt));
        cnt := retval.NEXT(cnt);
    END LOOP;
END;
/
...
Count = 2
266.4.7 GET_LOCAL_LINGUISTIC_SORTS Function

This function returns a list of the Oracle linguistic sort names that are appropriate for the specified language. A BINARY sort is included for all languages.

Syntax

```
UTL_I18N.GET_LOCAL_LINGUISTIC_SORTS (  
   language  IN VARCHAR2 CHARACTER SET ANY_CS)  
RETURN STRING_ARRAY;
```

Parameters

Table 266-8  GET_LOCAL_LINGUISTIC_SORTS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>Specifies a valid Oracle language. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

Examples

Displays the local linguistic sort names for JAPANESE.

```
DECLARE  
   retval UTL_I18N.STRING_ARRAY;  
   cnt INTEGER;  
BEGIN  
   retval := UTL_I18N.GET_LOCAL_LINGUISTIC_SORTS('Japanese');  
   DBMS_OUTPUT.PUT('Count = ');  
   DBMS_OUTPUT.PUT_LINE(retval.COUNT);  
   cnt := retval.FIRST;  
   WHILE cnt IS NOT NULL LOOP  
      DBMS_OUTPUT.PUT_LINE(retval(cnt));  
      cnt := retval.NEXT(cnt);  
   END LOOP;  
END;  
/  
...
```

Count = 2  
JAPANESE_M  
BINARY
266.4.8 GET_LOCAL_TERRITORIES Function

This function returns the local territory names for the specified language.

Syntax

UTL_I18N.GET_LOCAL_TERRITORIES (language IN VARCHAR2 CHARACTER SET ANY_CS) RETURN STRING_ARRAY;

Parameters

Table 266-9  GET_LOCAL_TERRITORIES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>Specifies a valid Oracle language. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

Examples

Returns the list of Oracle territories that use German as one of their local languages.

DECLARE
    retval UTL_I18N.STRING_ARRAY;
    cnt INTEGER;
BEGIN
    retval := UTL_I18N.GET_LOCAL_TERRITORIES('GERMAN');
    DBMS_OUTPUT.PUT('Count = ');
    DBMS_OUTPUT.PUT_LINE(retval.LAST);
    cnt := retval.FIRST;
    WHILE cnt IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(retval(cnt));
        cnt := retval.NEXT(cnt));
    END LOOP;
END;
/
... 
Count = 4
GERMANY
AUSTRIA
LUXEMBOURG
SWITZERLAND

266.4.9 GET_LOCAL_TIME_ZONES Function

This function returns the local time zone IDs for the specified territory.

Syntax

UTL_I18N.GET_LOCAL_TIME_ZONES (territory IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL) RETURN STRING_ARRAY;
Parameters

Table 266-10  GET_LOCAL_TIME_ZONES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>territory</td>
<td>Specifies a valid Oracle territory. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid territory name, then the function returns a `NULL` string.

Examples

Creates a function that returns the list of time zones locally used in the territory `AZERBAIJAN` followed by the general common time zones. This is useful for when the user's territory is known and the application still allows the user to choose other time zones as a user's preference.

```sql
CREATE OR REPLACE FUNCTION get_time_zones
(territory IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN utl_i18n.string_array
IS
  retval  utl_i18n.string_array;
  retval2 utl_i18n.string_array;
  stpos   INTEGER;
BEGIN
  retval  := utl_i18n.get_local_time_zones(territory);
  retval2 := utl_i18n.get_common_time_zones;
  stpos := retval.LAST + 1;
  retval(stpos) := '-----'; -- a separator
  FOR i IN retval2.FIRST..retval2.LAST LOOP
    stpos := stpos + 1;
    retval(stpos) := retval2(i);
  END LOOP;
  RETURN retval;
END;
/

Returns the list of local time zones for `AZERBAIJAN` followed by the common time zones with a separator string of five dashes (`-----`).

DECLARE
  retval UTL_I18N.STRING_ARRAY;
  cnt INTEGER;
BEGIN
  DBMS_OUTPUT.ENABLE(100000);
  retval UTL_I18N.GET_TIME_ZONES('AZERBAIJAN');
  cnt := retval.FIRST;
  WHILE cnt IS NOT NULL LOOP
    DBMS_OUTPUT.PUT_LINE(retval(cnt));
    cnt := retval.NEXT(cnt);
  END LOOP;
END;
/```

Asia/Baku

-----
266.4.10 GET_MAX_CHARACTER_SIZE Function

This function returns the maximum character size of a given character set.

Syntax

```sql
UTL_I18N.GET_MAX_CHARACTER_SIZE(
    charset_name       IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset_name</td>
<td>Specifies a valid character set name. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

For shift-sensitive character sets, the returned maximum character size will include the possible extra shift characters.

Examples

```sql
UTL_I18N.GET_MAX_CHARACTER_SIZE('AL32UTF8');
```

This returns 4.
266.4.11 GET_TRANSLATION Function

This function returns the translation of the language and territory name in the specified translation language.

Syntax

```
UTL_I18N.GET_TRANSLATION (  
  parameter        IN VARCHAR2 CHARACTER SET ANY_CS,  
  trans_language   IN VARCHAR2 'AMERICAN',  
  flag             IN PLS_INTEGER DEFAULT LANGUAGE_TRANS)  
RETURN VARCHAR2 CHARACTER SET parameter%CHARSET;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>Specifies a valid language name, territory name, or a combined string in the form of <code>language_territory</code>. It is case-insensitive.</td>
</tr>
<tr>
<td>trans_language</td>
<td>Specifies a translation language name. For example, <code>ITALIAN</code> is for the Italian language. The default is <code>AMERICAN</code>, which indicates American English.</td>
</tr>
<tr>
<td>flag</td>
<td>Specifies the translation type:</td>
</tr>
<tr>
<td></td>
<td>· <code>LANGUAGE_TRANS</code>: The function returns the language translation.</td>
</tr>
<tr>
<td></td>
<td>· <code>TERRITORY_TRANS</code>: The function returns the territory translation.</td>
</tr>
<tr>
<td></td>
<td>· <code>LANGUAGE_TERRITORY_TRANS</code>: The function returns the language and territory translation. The default translation type is <code>LANGUAGE_TRANS</code>.</td>
</tr>
</tbody>
</table>

Usage Notes

If `VARCHAR2` is used as a parameter type, the returned translation text can be corrupted due to the conversion to the database character set. Using `NVARCHAR2` as the parameter type will preserve the translation text because Unicode can encode all translated languages.

If the specified translation language is not available or an invalid name is provided, the default "American English" translations are returned. For example, Oracle does not provide `GUJARATI` translations, so the returned translation would be in American English.

Examples

The following returns the names of all the Oracle-supported languages in Italian.

```
DECLARE
  CURSOR c1 IS
    SELECT value FROM VSNLS_VALID_VALUES
    WHERE parameter = 'LANGUAGE'
    ORDER BY value;
  retval NVARCHAR2(100);
BEGIN
  FOR item IN c1 LOOP
    retval := UTL_I18N.GET_TRANSLATION (TO_CHAR(item.value), 'italian');
  END LOOP;
END;
```
266.4.12 MAP_CHARSET Function

This function maps a character set to another character set.

It maps the following:

- An Oracle character set name to an IANA character set name.
- An IANA character set name to an Oracle character set name.
- An Oracle character set to an e-mail safe character set name.

Syntax

```sql
UTL_I18N.MAP_CHARSET(
    charset   IN VARCHAR2,
    context   IN PLS_INTEGER DEFAULT GENERIC_CONTEXT,
    flag      IN PLS_INTEGER DEFAULT ORACLE_TO_IANA)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>Specifies the character set name to be mapped. The mapping is case-insensitive.</td>
</tr>
<tr>
<td>context</td>
<td>GENERIC_CONTEXT</td>
</tr>
<tr>
<td></td>
<td>GENERIC CONTEXT: The mapping is between an Oracle character set name and an IANA character set name. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>MAIL_CONTEXT: The mapping is between an Oracle character set name and an email safe character set name.</td>
</tr>
<tr>
<td>flag</td>
<td>ORACLE_TO_IANA</td>
</tr>
<tr>
<td></td>
<td>ORACLE_TO_IANA: Map from an Oracle character set name to an IANA character set name. This is the default.</td>
</tr>
<tr>
<td></td>
<td>IANA_TO_ORACLE: Map from an IANA character set name to an Oracle character set name.</td>
</tr>
<tr>
<td></td>
<td>MAIL_GENERIC</td>
</tr>
<tr>
<td></td>
<td>MAIL_GENERIC: Map from an Oracle character set name to an email safe character set name on a non-Windows platform.</td>
</tr>
<tr>
<td></td>
<td>MAIL_WINDOWS: Map from an Oracle character set name to an email safe character set name on a Windows platform.</td>
</tr>
</tbody>
</table>

Usage Notes

An e-mail safe character set is an Oracle character set that is commonly used by applications when they submit e-mail messages. The character set is usually used to convert contents in the database character set to e-mail safe contents. To specify the character set name in the mail header, you should use the corresponding IANA character set name obtained by calling the MAP_CHARSET function with the ORACLE_TO_IANA option, providing the e-mail safe character set name as input.

For example, no e-mail client recognizes message contents in the WE8DEC character set, whose corresponding IANA name is DEC-MCS. If WE8DEC is passed to the
The MAP_CHARSET function with the MAIL_CONTEXT option, then the function returns WE8ISO8859P1. Its corresponding IANA name, ISO-8859-1, is recognized by most e-mail clients.

The steps in this example are as follows:

1. Call the MAP_CHARSET function with the MAIL_CONTEXT | MAIL_GENERIC option with the database character set name, WE8DEC. The result is WE8ISO8859P1.
2. Convert the contents stored in the database to WE8ISO8859P1.
3. Call the MAP_CHARSET function with the ORACLE_TO_IANA | GENERIC_CONTEXT option with the e-mail safe character set, WE8ISO8859P1. The result is ISO-8859-1.
4. Specify ISO-8859-1 in the mail header when the e-mail message is submitted.

The function returns a character set name if a match is found. If no match is found or if the flag is invalid, then it returns NULL.

Note:
Many Oracle character sets can map to one e-mail safe character set. There is no function that maps an e-mail safe character set to an Oracle character set name.

Examples
Generic Context
UTL_I18N.MAP_CHARSET('iso-8859-1', UTL_I18N.GENERIC_CONTEXT, UTL_I18N.IANA_TO_ORACLE)
This returns 'WE8ISO8859P1'.

Context
UTL_I18N.MAP_CHARSET('WE8DEC', utl_i18n.mail_context, utl_i18n.mail_generic)
This returns 'WE8ISO8859P1'.

See Also:
Oracle Database Globalization Support Guide for a list of valid Oracle character sets

266.4.13 MAP_FROM_SHORT_LANGUAGE Function

This function maps an Oracle short language name to an Oracle language name.

Syntax

```sql
UTL_I18N.MAP_FROM_SHORT_LANGUAGE ( 
    language IN VARCHAR2 CHARACTER SET ANY_CS) 
RETURN VARCHAR2;
```
Parameters

Table 266-14 MAP_FROM_SHORT_LANGUAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>Specifies a valid short language name. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

Examples

Returns the default linguistic sort name for the customer with the ID of 9000. Note that the table customers is from the oe user in the Common Schema. Because the customer’s language preference is stored using a short language name, you need to convert to a full language name by calling the GET_DEFAULT_LINGUISTIC_SORT procedure.

```plsql
DECLARE
    short_n VARCHAR2(10);
    ling_n VARCHAR2(50);
BEGIN
    SELECT nls_language INTO short
    FROM customers WHERE customer_id = 9000;
    ling_n := UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT (UTL_I18N.MAP_FROM_SHORT_LANGUAGE(short_n));
    DBMS_OUTPUT.PUT_LINE(ling_n);
END;
/
```

266.4.14 MAP_LANGUAGE_FROM_ISO Function

This function returns an Oracle language name from an ISO locale name.

Syntax

```plsql
UTL_I18N.MAP_LANGUAGE_FROM_ISO(
    isolocale IN VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table 266-15 MAP_LANGUAGE_FROM_ISO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isolocale</td>
<td>Specifies the ISO locale. The mapping is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid locale string, then the function returns a NULL string.
If the user specifies a locale string that includes only the language (for example, `en` instead of `en_US`), then the function returns the default language name for the specified language (for example, `American`).

**Examples**

```
UTL_I18N.MAP_LANGUAGE_FROM_ISO('en_US')
```

This returns 'American'.

---

**See Also:**

`Oracle Database Globalization Support Guide` for a list of valid Oracle languages

---

### 266.4.15 MAP_LOCALE_TO_ISO Function

This function returns an ISO locale name from an Oracle language name and an Oracle territory name.

A valid string must include at least one of the following: a valid Oracle language name or a valid Oracle territory name.

**Syntax**

```
UTL_I18N.MAP_LOCALE_TO_ISO (  
   ora_language   IN VARCHAR2,  
   ora_territory  IN VARCHAR2)  
RETURN VARCHAR2;
```

**Parameters**

**Table 266-16  MAP_LOCALE_TO_ISO Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ora_language</td>
<td>Specifies an Oracle language name. It is case-insensitive.</td>
</tr>
<tr>
<td>ora_territory</td>
<td>Specifies an Oracle territory name. It is case-insensitive.</td>
</tr>
</tbody>
</table>

**Usage Notes**

If the user specifies an invalid string, then the function returns a `NULL` string.

**Examples**

```
UTL_I18N.MAP_LOCALE_TO_ISO('American','America')
```

This returns 'en_US'.

---

---
266.4.16 MAP_TERRITORY_FROM_ISO Function

This function returns an Oracle territory name from an ISO locale.

Syntax

```
UTL_I18N.MAP_TERRITORY_FROM_ISO (
isolocale IN VARCHAR2)
RETURN VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isolocale</td>
<td>Specifies the ISO locale. The mapping is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid locale string, then the function returns a NULL string.

If the user specifies a locale string that includes only the territory (for example, _fr instead of fr_fr), then the function returns the default territory name for the specified territory (for example, France).

Examples

```
UTL_I18N.MAP_TERRITORY_FROM_ISO('en_US')
```

This returns 'America'.

266.4.17 MAP_TO_SHORT_LANGUAGE Function

This function maps an Oracle language name to an Oracle short language name.

Syntax

```
UTL_I18N.MAP_TO_SHORT_LANGUAGE (language IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```
Parameters

Table 266-18  MAP_TO_SHORT_LANGUAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>Specifies a valid full language name. It is case-insensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

Examples

Returns the short language name for the language.

```sql
DECLARE  retval VARCHAR2(100);BEGIN  retval := UTL_I18N.MAP_TO_SHORT_LANGUAGE('american');  DBMS_OUTPUT.PUT_LINE(retval);END;/
```

266.4.18 RAW_TO_CHAR Functions

This function converts RAW data from a valid Oracle character set to a VARCHAR2 string in the database character set.

The function is overloaded. The different forms of functionality are described along with the syntax declarations.

Syntax

Buffer Conversion:

```sql
UTL_I18N.RAW_TO_CHAR(  
data          IN RAW,  
src_charset   IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

Piecewise conversion converts raw data into character data piece by piece:

```sql
UTL_I18N.RAW_TO_CHAR (  
data          IN RAW,  
src_charset   IN VARCHAR2 DEFAULT NULL,  
scanned_length OUT PLS_INTEGER,  
shift_status  IN OUT PLS_INTEGER)  
RETURN VARCHAR2;
```

Parameters

Table 266-19  RAW_TO_CHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Specifies the RAW data to be converted to a VARCHAR2 string</td>
</tr>
<tr>
<td>src_charset</td>
<td>Specifies the character set that the RAW data was derived from. If src_charset is NULL, then the database character set is used.</td>
</tr>
<tr>
<td>scanned_length</td>
<td>Specifies the number of bytes of source data scanned</td>
</tr>
</tbody>
</table>
Table 266-19  (Cont.) RAW_TO_CHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shift_status</td>
<td>Specifies the shift status at the end of the scan. The user must set it to SHIFT_IN the first time it is called in piecewise conversion.</td>
</tr>
</tbody>
</table>

Note: ISO 2022 character sets use escape sequences instead of shift characters to indicate the encoding method. shift_status cannot hold the encoding method information that is provided by the escape sequences for the next function call. As a result, this function cannot be used to reconstruct ISO 2022 character from raw data in a piecewise way unless each unit of input can be guaranteed to be a closed string. A closed string begins and ends in a 7-bit escape state.

Usage Notes

If the user specifies an invalid character set, NULL data, or data whose length is 0, then the function returns a NULL string.

Examples

Buffer Conversion

UTL_I18N.RAW_TO_CHAR(hextoraw('616263646566C2AA'), 'utf8')

This returns the following string in the database character set:

'abcde'||chr(170)

Piecewise Conversion

UTL_I18N.RAW_TO_CHAR(hextoraw('616263646566C2AA'),'utf8',shf,slen)

This expression returns the following string in the database character set:

'abcde'||chr(170)

It also sets shf to SHIFT_IN and slen to 8.

The following example converts data from the Internet piece by piece to the database character set.

rvalue RAW(1050);
nvalue VARCHAR2(1024);
conversion_state PLS_INTEGER = 0;
converted_len PLS_INTEGER;
rtmp RAW(10) = ' ';
conn utl_tcp.connection;
tlen PLS_INTEGER;

...  
conn := utl_tcp.open_connection ( remote_host => 'localhost',
               remote_port => 2000);
LOOP  
tlen := utl_tcp.read_raw(conn, rvalue, 1024);
rvalue := utl_raw.concat(rtemp, rvalue);
nvalue := utl_i18n.raw_to_char(rvalue, 'JA16SJIS', converted_len,
               conversion_stat);
       if (converted_len < utl_raw.length(rvalue) )
then
  rtemp := utl_raw.substr(rvalue, converted_len+1);
else
  rtemp := '';
end if;
/* do anything you want with nvalue */
/* e.g htp.prn(nvalue); */
END LOOP;
utl_tcp.close_connection(conn);
EXCEPTION
  WHEN utl_tcp.end_of_input THEN
    utl_tcp.close_connection(conn);
END;

### 266.4.19 RAW_TO_NCHAR Functions

This function converts RAW data from a valid Oracle character set to an NVARCHAR2 string in the national character set.

The function is overloaded. The different forms of functionality are described along with the syntax declarations.

#### Syntax

**Buffer Conversion:**

```sql
UTL_I18N.RAW_TO_NCHAR (  
data         IN RAW,  
src_charset  IN VARCHAR2 DEFAULT NULL)  
RETURN NVARCHAR2;
```

**Piecewise conversion converts raw data into character data piece by piece:**

```sql
UTL_I18N.RAW_TO_NCHAR (  
data            IN RAW,  
src_charset      IN VARCHAR2 DEFAULT NULL,  
scanned_length   OUT PLS_INTEGER,  
shift_status     IN OUT PLS_INTEGER)  
RETURN NVARCHAR2;
```

#### Parameters

**Table 266-20  RAW_TO_NCHAR Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Specifies the RAW data to be converted to an NVARCHAR2 string</td>
</tr>
<tr>
<td>src_charset</td>
<td>Specifies the character set that the RAW data was derived from. If src_charset is NULL, then the database character set is used.</td>
</tr>
<tr>
<td>scanned_length</td>
<td>Specifies the number of bytes of source data scanned</td>
</tr>
</tbody>
</table>
Table 266-20  (Cont.) RAW_TO_NCHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shift_status</td>
<td>Specifies the shift status at the end of the scan. The user must set it to SHIFT_IN the first time it is called in piecewise conversion. <strong>Note:</strong> ISO 2022 character sets use escape sequences instead of shift characters to indicate the encoding method. shift_status cannot hold the encoding method information that is provided by the escape sequences for the next function call. As a result, this function cannot be used to reconstruct ISO 2022 character from raw data in a piecewise way unless each unit of input can be guaranteed to be a closed string. A closed string begins and ends in a 7-bit escape state.</td>
</tr>
</tbody>
</table>

**Usage Notes**

If the user specifies an invalid character set, **NULL** data, or data whose length is 0, then the function returns a **NULL** string.

**Examples**

**Buffer Conversion**

UTL_I18N.RAW_TO_NCHAR(hextoraw('616263646566C2AA'),'utf8')

This returns the following string in the national character set:

'abcde'||chr(170)

**Piecewise Conversion**

UTL_I18N.RAW_TO_NCHAR(hextoraw('616263646566C2AA'),'utf8', shf, slen)

This expression returns the following string in the national character set:

'abcde'||chr(170)

It also sets shf to **SHIFT_IN** and slen to 8.

The following example converts data from the Internet piece by piece to the national character set.

```sql
rvalue RAW(1050);
nvalue NVARCHAR2(1024);
conversion_state PLS_INTEGER = 0;
converted_len PLS_INTEGER;
rtemp RAW(10) = '';
conn utl_tcp.connection;
tlen PLS_INTEGER;
...
conn := utl_tcp.open_connection ( remote_host => 'localhost',
                                      remote_port => 2000);
LOOP
  tlen := utl_tcp.read_raw(conn, rvalue, 1024);
rvalue := utl_raw.concat(rtemp, rvalue);
nvalue := utl_i18n.raw_to_nchar(rvalue, 'JA16SJIS', converted_len,
                                 conversion_stat);
  if (converted_len < utl_raw.length(rvalue) )
```
then
    rtemp := utl_raw.substr(rvalue, converted_len+1);
else
    rtemp := '':
end if;
/* do anything you want with nvalue */
/* e.g htp.prn(nvalue); */
END LOOP;
utl_tcp.close_connection(conn);
EXCEPTION
    WHEN utl_tcp.end_of_input THEN
        utl_tcp.close_connection(conn);
END;

266.4.20 STRING_TO_RAW Function

This function converts a VARCHAR2 or NVARCHAR2 string to another valid Oracle character set and returns the result as RAW data.

Syntax

UTL_I18N.STRING_TO_RAW(
    data          IN VARCHAR2 CHARACTER SET ANY_CS,
    dst_charset   IN VARCHAR2 DEFAULT NULL)
RETURN RAW;

Parameters

Table 266-21 STRING_TO_RAW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Specifies the VARCHAR2 or NVARCHAR2 string to convert.</td>
</tr>
<tr>
<td>dst_charset</td>
<td>Specifies the destination character set. If dst_charset is NULL, then the database character set is used for CHAR data and the national character set is used for NCHAR data.</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies an invalid character set, a NULL string, or a string whose length is 0, then the function returns a NULL string.

Examples

DECLARE
    r raw(50);
    s varchar2(20);
BEGIN
    s:='abcdef'||chr(170);
    r:=utl_i18n.string_to_raw(s,'utf8');
    dbms_output.put_line(rawtohex(r));
    end;
/

This returns a hex value of '616263646566C2AA'.

266.4.21 TRANSLITERATE Function

This function performs script transliteration. In this release, the TRANSLITERATE function only supports Japanese Kana conversion.

Syntax

UTL_I18N.TRANSLITERATE (  
    data IN VARCHAR2 CHARACTER SET ANY_CS,  
    name IN VARCHAR2)  
RETURN VARCHAR2 CHARACTER SET data%CHARSET;

Parameters

Table 266-22 TRANSLITERATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Specifies the data to be converted. Either CHAR or NCHAR datatype can be specified.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the transliteration name string. For a list of valid names, see Table 266-23.</td>
</tr>
</tbody>
</table>

Constants

These options specify Japanese Kana conversions.

Table 266-23 TRANSLITERATE Function Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KANA_FWKATAKANA</td>
<td>'kana_fwkatakana'</td>
<td>Converts any type of Kana character to a fullwidth Katakana character.</td>
</tr>
<tr>
<td>KANA_HWKATAKANA</td>
<td>'kana_hwkatakana'</td>
<td>Converts any type of Kana character to a halfwidth Katakana character.</td>
</tr>
<tr>
<td>KANA_HIRAGANA</td>
<td>'kana_hiragana'</td>
<td>Converts any type of Kana character to a fullwidth Hiragana character.</td>
</tr>
<tr>
<td>FWKATAKANA_HWKATAKANA</td>
<td>'fwkatakana_hwkatakana'</td>
<td>Converts only fullwidth Katakana characters to halfwidth Katakana characters.</td>
</tr>
<tr>
<td>FWKATAKANA_HIRAGANA</td>
<td>'fwkatakana_hiragana'</td>
<td>Converts only fullwidth Katakana characters to fullwidth Hiragana characters.</td>
</tr>
<tr>
<td>HIKATAKANA_FWKATAKANA</td>
<td>'hikatakana_fwkatakana'</td>
<td>Converts only halfwidth Katakana characters to fullwidth Katakana characters.</td>
</tr>
<tr>
<td>HIKATAKANA_HIRAGANA</td>
<td>'hikatakana_hiragana'</td>
<td>Converts only halfwidth Katakana characters to fullwidth Hiragana characters.</td>
</tr>
<tr>
<td>HIRAGANA_FWKATAKANA</td>
<td>'hiragana_fwkatakana'</td>
<td>Converts only fullwidth Hiragana characters to fullwidth Katakana characters.</td>
</tr>
<tr>
<td>HIRAGANA_HWKATAKANA</td>
<td>'hiragana_hwkatakana'</td>
<td>Converts only fullwidth Hiragana characters to halfwidth Katakana characters.</td>
</tr>
</tbody>
</table>
Usage Notes

The function returns the converted string.

Examples

Given a table `japanese_emp`, containing an `NVARCHAR2` column `ename`, the following statement can be used to normalize all the kana names in `ename` to hiragana:

```sql
UPDATE japanese_emp
SET ename = UTL_I18N.TRANSLITERATE (ename, 'kana_hiragana');
```

The following figure shows how this output might look.

![Figure 266-1 Loading Locale-Specific Data to the Database](image)

The following statement normalizes one kana name to hiragana:

```sql
DECLARE
    Name  japanese_emp.ename%TYPE;
    Eno   CONSTANT  NUMBER(4) := 1;
BEGIN
    SELECT ename INTO name FROM japanese_emp WHERE enumber = eno;
    name := UTL_I18N.TRANSLITERATE(name, UTL_I18N.KANA_HIRAGANA);
    UPDATE japanese_emp SET ename = name WHERE enumber = eno;
EXCEPTION
    WHEN UTL_I18N.UNSUPPORTED_TRANSLITERATION THEN
        DBMS_OUTPUT.PUT_LINE('transliteration not supported');
END;
/
```

266.4.22 UNEscape_REFERENCE Function

This function returns a string from an input string that contains character references. It decodes each character reference to the corresponding character value.

See Also:

"ESCAPE_REFERENCE Function" for more information about escape sequences
Syntax

UTL_I18N.UNESCAPE_REFERENCE (str IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2 CHARACTER SET str%CHARSET;

Parameters

Table 266-24  UNESCAPE_REFERENCE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>Specifies the input string</td>
</tr>
</tbody>
</table>

Usage Notes

If the user specifies a NULL string or a string whose length is 0, then the function returns a NULL string. If the function fails, then it returns the original string.

Examples

UTL_I18N.UNESCAPE_REFERENCE('hello &lt; &amp;#xe5;')

This returns 'hello '<||chr(229).

266.4.23 VALIDATE_CHARACTER_ENCODING Functions

This function validates the character encoding of VARCHAR2, NVARCHAR2, CLOB, and NCLOB data. The validation is based on the database character set for VARCHAR2 and CLOB data and national character set for NVARCHAR2 and NCLOB data.

For Unicode character sets, such as AL32UTF8, AL16UTF16, AL16UTF16LE, UTF8, and UTFE, any byte sequences mapped to the following Unicode code points are considered invalid:

- Unpaired surrogate code point
- Non-character code point

In addition, any irregular or illegal UTF-8 byte sequence is considered invalid for AL32UTF8 and UTF8 character sets.

The VALIDATE_CHARACTER_ENCODING function is overloaded. One function is for validating VARCHAR2 and NVARCHAR2 data, while the other function is for validating CLOB and NCLOB data.

- Validating VARCHAR2 and NVARCHAR2 data
  A VARCHAR2 or NVARCHAR2 byte or its byte sequence is considered invalid for a character set, if it does not map to any of the characters defined in the character set.

- Validating CLOB and NCLOB data
  A LOB character is considered invalid for a character set if a byte (in case of a single-byte database character set) or a byte pair (in case of UTF-16 encoding used with a multibyte database character set) corresponding to the encoding of
the LOB character does not map to any of the characters defined in the character set.

**Syntax**

This function validates VARCHAR2 and NVARCHAR2 data:

```sql
UTL_I18N.VALIDATE_CHARACTER_ENCODING (  
data IN VARCHAR2 CHARACTER SET ANY_CS)  
RETURN PLS_INTEGER;
```

This function validates CLOB and NCLOB data:

```sql
UTL_I18N.VALIDATE_CHARACTER_ENCODING (  
lob_loc IN CLOB CHARACTER SET ANY_CS)  
RETURN PLS_INTEGER;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>VARCHAR2 or NVARCHAR2 data to validate.</td>
</tr>
<tr>
<td>lob_loc</td>
<td>CLOB or NCLOB data to validate.</td>
</tr>
</tbody>
</table>

**Usage Notes**

This function returns the offset of the first invalid byte for the VARCHAR2 or NVARCHAR2 data. It returns the offset of the first invalid character for the CLOB or NCLOB data. It returns 0, if all the bytes in the character data are valid. It returns NULL, if the value of the parameter data or lob_loc is NULL.

**Examples**

This example validates the character encoding of NVARCHAR2 and CLOB data where the database character set is AL32UTF8 while the national character set is AL16UTF16.

```sql
CREATE TABLE temp(col1 NVARCHAR2(20), col2 CLOB);
INSERT INTO temp VALUES(UNISTR('foo\D800bar'), UNISTR('foo\D800bar'));
COMMIT;
SELECT UTL_I18N.VALIDATE_CHARACTER_ENCODING(col1) invalid_offset_column1,  
UTL_I18N.VALIDATE_CHARACTER_ENCODING(col2) invalid_offset_column2  
FROM temp;
```

The query returns:

```
INVALID_OFFSET_COLUMN1 INVALID_OFFSET_COLUMN2
---------------------- ----------------------
7                      4
```

Here, the surrogate code point U+D800 is invalid. The number 7 is returned as INVALID_OFFSET_COLUMN1, because for col1, 'foo' is encoded in 6 bytes in NVARCHAR2 and the invalid code point U+D800 starts at offset 7. The number 4 is returned as INVALID_OFFSET_COLUMN2, because for col2, 'foo' is encoded in 3 UTF-16 code points in CLOB and the invalid code point U+D800 starts at offset 4.
**UTL_IDENT**

The **UTL_IDENT** package specifies which Database or client PL/SQL is running.

This chapter contains the following topics:

- **Overview**
- **Security Model**
- **Constants**

**267.1 UTL_IDENT Overview**

The **UTL_IDENT** package is intended for use for conditional compilation of PL/SQL packages that are supported by Oracle, TimesTen Database, and clients such as Oracle Forms.

**267.2 UTL_IDENT Security Model**

The **UTL_IDENT** package runs as the package owner **SYS**. The public synonym **UTL_IDENT**, and **EXECUTE** permission on this package is granted to **PUBLIC**.

**267.3 UTL_IDENT Constants**

The **UTL_IDENT** package defines several constants to use when specifying parameter values.

These constants are shown in the following table.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_ORACLE_SERVER</td>
<td>BOOLEAN</td>
<td>TRUE/FALSE</td>
<td>Stipulates if Oracle Server or not</td>
</tr>
<tr>
<td>IS_ORACLE_CLIENT</td>
<td>BOOLEAN</td>
<td>TRUE/FALSE</td>
<td>Stipulates if Oracle Client or not</td>
</tr>
<tr>
<td>IS_TIMESTEN</td>
<td>BOOLEAN</td>
<td>TRUE/FALSE</td>
<td>Stipulates if TimesTen or not</td>
</tr>
<tr>
<td>IS_ORACLE_FORMS</td>
<td>BOOLEAN</td>
<td>TRUE/FALSE</td>
<td>Stipulates if Oracle Forms or not</td>
</tr>
</tbody>
</table>
The **UTL_INADDR** package provides a PL/SQL procedures to support internet addressing. It provides an API to retrieve host names and IP addresses of local and remote hosts.

This chapter contains the following topics:

- Security Model
- Exceptions
- Examples
- Summary of UTL_INADDR Subprograms

### 268.1 UTL_INADDR Security Model

This package is an invoker’s rights package, which means that the invoking user must be granted the `CONNECT` privilege in the access control list assigned to the remote network host to which he or she wishes to connect.

**Note:**

For more information about managing fine-grained access, see *Oracle Database Security Guide*.

### 268.2 UTL_INADDR Exceptions

This table describes exceptions raised by **UTL_INADDR** subprograms.

<table>
<thead>
<tr>
<th>Number</th>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-24247</td>
<td>NETWORK_ACCESS_DENIED</td>
<td>Access to network is denied.</td>
</tr>
<tr>
<td>ORA-29257</td>
<td>UNKNOWN_HOST</td>
<td>The host is unknown.</td>
</tr>
</tbody>
</table>

### 268.3 UTL_INADDR Examples

This **UTL_INADDR** example retrieves the local host name and IP address.

```sql
SET serveroutput on
BEGIN
  DBMS_OUTPUT.PUT_LINE(UTL_INADDR.GET_HOST_NAME);  -- get local host name
  DBMS_OUTPUT.PUT_LINE(UTL_INADDR.GET_HOST_ADDRESS);  -- get local IP addr
END;
/```
268.4 Summary of UTL_INADDR Subprograms

This table lists the UTL_INADDR subprograms and briefly describes them.

Table 268-2  UTL_INADDR Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_HOST_ADDRESS</td>
<td>Retrieves the IP address of the local or remote host given its name</td>
</tr>
<tr>
<td>GET_HOST_NAME</td>
<td>Retrieves the name of the local or remote host given its IP address</td>
</tr>
</tbody>
</table>

268.4.1 GET_HOST_ADDRESS Function

This function retrieves the IP address of the specified host.

Syntax

UTL_INADDR.GET_HOST_ADDRESS (    host  IN VARCHAR2 DEFAULT NULL)    RETURN host_address VARCHAR2;

Parameters

Table 268-3  GET_HOST_ADDRESS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The name of the host to retrieve the IP address.</td>
</tr>
</tbody>
</table>

Return Values

Table 268-4  GET_HOST_ADDRESS Function Return Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host_address</td>
<td>The IP address of the specified host, or that of the local host if host is NULL.</td>
</tr>
</tbody>
</table>

Exceptions

UNKNOWN_HOST: The specified IP address is unknown

Usage Notes

The permission to obtain the host name or IP address of the current host is controlled by the resolve privilege on LOCALHOST.
268.4.2 GET_HOST_NAME Function

This function retrieves the name of the local or remote host given its IP address.

Syntax

UTL_INADDR.GET_HOST_NAME (  
ip  IN VARCHAR2 DEFAULT NULL)  
RETURN host_name VARCHAR2;

Parameters

Table 268-5   GET_HOST_NAME Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>The IP address of the host used to determine its host name. If ip is not NULL, the official name of the host with its domain name is returned. If this is NULL, the name of the local host is returned and the name does not contain the domain to which the local host belongs.</td>
</tr>
</tbody>
</table>

Return Values

Table 268-6   GET_HOST_NAME Function Return Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host_name</td>
<td>The name of the local or remote host of the specified IP address.</td>
</tr>
</tbody>
</table>

Exceptions

UNKNOWN_HOST: The specified IP address is unknown

Usage Notes

The permission to obtain the host name or IP address of the current host is controlled by the resolve privilege granted through DBMS_NETWORK_ACL_ADMIN on LOCALHOST.
UTL_LMS retrieves and formats error messages in different languages.

This chapter contains the following topics:

- Security Model
- Summary of UTL_LMS Subprograms

See Also:
Oracle Database Globalization Support Guide

269.1 UTL_LMS Security Model

This package must be created as the user SYS.

269.2 Summary of UTL_LMS Subprograms

This table lists the UTL_LMS subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT_MESSAGE Function</td>
<td>Formats a retrieved error message</td>
</tr>
<tr>
<td>GET_MESSAGE Function</td>
<td>Retrieves an error message based on error number, product, facility,</td>
</tr>
<tr>
<td></td>
<td>language, and message specified</td>
</tr>
</tbody>
</table>

269.2.1 FORMAT_MESSAGE Function

This function formats a message retrieved by the GET MESSAGE function and returns the formatted message. If the function fails, then it returns a NULL result.

The following table shows special characters that can be used in the format string.

<table>
<thead>
<tr>
<th>Special Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'%s'</td>
<td>Substitute the next string argument</td>
</tr>
<tr>
<td>'%d'</td>
<td>Substitute the next integer argument</td>
</tr>
<tr>
<td>'%'</td>
<td>Represents the special character %</td>
</tr>
</tbody>
</table>
Syntax

```
UTL_LMS.FORMAT_MESSAGE (  
    format IN VARCHAR2 CHARACTER SET ANY_CS,  
    args   IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL)  
RETURN VARCHAR2 CHARACTER SET format%CHARSET;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Specifies the string to format</td>
</tr>
<tr>
<td>args</td>
<td>Specifies the list of arguments</td>
</tr>
</tbody>
</table>

Examples

```
DECLARE  
    s varchar2(200);  
    i  pls_integer;  
BEGIN  
    i:= utl_lms.get_message(26052, 'rdbms', 'ora', 'french', s);  
    dbms_output.put_line('before format, message is: '||s);  
    dbms_output.put_line('formatted message is: '||utl_lms.format_message(s, 9,  
    'my_column_name');  
END;
/
```

The following is an unformatted message:

Type %d non pris en charge pour l'expression SQL sur la colonne %s.

The following is the formatted message:

Type 9 non pris en charge pour l'expression SQL sur la colonne my_column_name.

269.2.2 GET_MESSAGE Function

This function retrieves an Oracle error message. The user can define user-specific error messages with the lmsgen utility.

It returns 0 when it is successful. It returns -1 when it fails.

See Also:

Oracle Database Globalization Support Guide for more information about the lmsgen utility

Syntax

```
UTL_LMS.GET_MESSAGE (  
    errnum    IN PLS_INTEGER,  
    product   IN VARCHAR2,  
    s        IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL)  
RETURN VARCHAR2 CHARACTER SET product%CHARSET;
```
facility  IN VARCHAR2,
language  IN VARCHAR2,
message   OUT NOCOPY VARCHAR2CHARACTER SET ANY_CS)
RETURN PLS_INTEGER;

Parameters

Table 269-3  GET_MESSAGE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errnum</td>
<td>Specifies the error number. Example: '972' (for ORA-00972)</td>
</tr>
<tr>
<td>product</td>
<td>Specifies the product to which the error message applies Example: 'rdbms'</td>
</tr>
<tr>
<td>facility</td>
<td>Specifies the error message prefix Example: 'ora'</td>
</tr>
<tr>
<td>language</td>
<td>Specifies the language of the message. The parameter is case-insensitive. The default is NULL, which causes GET_MESSAGE to use the value of the NLS_LANGUAGE session parameter.</td>
</tr>
<tr>
<td>message</td>
<td>Specifies the output buffer for the retrieved message</td>
</tr>
</tbody>
</table>

Usage Notes

If the language parameter is set to NULL, then the value of the NLS_LANGUAGE session parameter is used as the default.

Examples

DECLARE
  s varchar2(200);
i pls_integer;
BEGIN
  i:=utl_lms.get_message(601, 'rdbms', 'oci', 'french', s);
dbms_output.put_line('OCI--00601 is: '||s);
END /

The following output results:

OCI--00601 is: Echec du processus de nettoyage.
The UTL_MAIL package is a utility for managing email which includes commonly used email features, such as attachments, CC, and BCC.

This chapter contains the following topics:

- Security Model
- Operational Notes
- Rules and Limits
- Summary of UTL_MAIL Subprograms

270.1 UTL_MAIL Security Model

UTL_MAIL is not installed by default because of the SMTP_OUT_SERVER configuration requirement and the security exposure this involves. In installing UTL_MAIL, you should take steps to prevent the port defined by SMTP_OUT_SERVER being swamped by data transmissions.

This package is now an invoker's rights package and the invoking user will need the connect privilege granted in the access control list assigned to the remote network host to which he wants to connect.

Note:

For more information about managing fine-grained access, see Oracle Database Security Guide.

270.2 UTL_MAIL Operational Notes

You must both install UTL_MAIL and define the SMTP_OUT_SERVER.

- To install UTL_MAIL:
  sqlplus sys/<pwd>
  SQL> @$ORACLE_HOME/rdbms/admin/utlmail.sql
  SQL> @$ORACLE_HOME/rdbms/admin/prvtmail.plb

- You define the SMTP_OUT_SERVER parameter in the init.ora rdbms initialization file. However, if SMTP_OUT_SERVER is not defined, this invokes a default of DB_DOMAIN which is guaranteed to be defined to perform appropriately.
270.3 UTL_MAIL Rules and Limits

Use UTL_MAIL only within the context of the ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary-Coded Decimal Interchange Code) codes.

270.4 Summary of UTL_MAIL Subprograms

This table lists the UTL_MAIL subprograms and briefly describes them.

Table 270-1     UTL_MAIL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND Procedure</td>
<td>Packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients</td>
</tr>
<tr>
<td>SEND_ATTACH_RAW Procedure</td>
<td>Represents the SEND Procedure overloaded for RAW attachments</td>
</tr>
<tr>
<td>SEND_ATTACH_VARCHAR2 Procedure</td>
<td>Represents the SEND Procedure overloaded for VARCHAR2 attachments</td>
</tr>
</tbody>
</table>

270.4.1 SEND Procedure

This procedure packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients.

It hides the SMTP API and exposes a one-line email facility for ease of use.

Syntax

UTL_MAIL.SEND (sender IN VARCHAR2 CHARACTER SET ANY_CS,
recipient IN VARCHAR2 CHARACTER SET ANY_CS,
cc IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
bcc IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
subject IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
message IN VARCHAR2 CHARACTER SET ANY_CS,
mime_type IN VARCHAR2 DEFAULT 'text/plain; charset=us-ascii',
priority IN PLS_INTEGER DEFAULT 3,
replyto IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);

Parameters

Table 270-2     SEND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender</td>
<td>Email address of the sender</td>
</tr>
<tr>
<td>recipients</td>
<td>Email addresses of the recipient(s), separated by commas</td>
</tr>
</tbody>
</table>
### Table 270-2  (Cont.) SEND Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>Email addresses of the CC recipient(s), separated by commas, default is NULL</td>
</tr>
<tr>
<td>bcc</td>
<td>Email addresses of the BCC recipient(s), separated by commas, default is NULL</td>
</tr>
<tr>
<td>subject</td>
<td>String to be included as email subject string, default is NULL</td>
</tr>
<tr>
<td>message</td>
<td>Text message body</td>
</tr>
<tr>
<td>mime_type</td>
<td>Mime type of the message, default is 'text/plain; charset=us-ascii'</td>
</tr>
<tr>
<td>priority</td>
<td>Message priority, which maps to the X-priority field. 1 is the highest priority and 5 the lowest. The default is 3.</td>
</tr>
<tr>
<td>replyto</td>
<td>Defines to whom the reply email is to be sent</td>
</tr>
</tbody>
</table>

#### 270.4.2 SEND_ATTACH_RAW Procedure

This procedure is the SEND Procedure overloaded for RAW attachments.

### Syntax

```sql
UTL_MAIL.SEND_ATTACH_RAW (  
  sender           IN    VARCHAR2 CHARACTER SET ANY_CS,  
  recipients       IN    VARCHAR2 CHARACTER SET ANY_CS,  
  cc               IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,  
  bcc              IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,  
  subject          IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,  
  message          IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,  
  mime_type        IN    VARCHAR2 DEFAULT CHARACTER SET ANY_CS DEFAULT 'text/plain; charset=us-ascii',  
  priority         IN    PLS_INTEGER DEFAULT 3,  
  attachment       IN    RAW,  
  att_inline       IN    BOOLEAN DEFAULT TRUE,  
  att_mime_type    IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT 'text/plain; charset=us-ascii',  
  att_filename     IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,  
  replyto          IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender</td>
<td>Email address of the sender</td>
</tr>
<tr>
<td>recipients</td>
<td>Email addresses of the recipient(s), separated by commas</td>
</tr>
<tr>
<td>cc</td>
<td>Email addresses of the CC recipient(s), separated by commas, default is NULL</td>
</tr>
<tr>
<td>bcc</td>
<td>Email addresses of the BCC recipient(s), separated by commas, default is NULL</td>
</tr>
<tr>
<td>subject</td>
<td>String to be included as email subject string, default is NULL</td>
</tr>
</tbody>
</table>
Table 270-3  (Cont.) SEND_ATTACHMENT_RAW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Text message body</td>
</tr>
<tr>
<td>mime_type</td>
<td>Mime type of the message, default is 'text/plain; charset=us-ascii'</td>
</tr>
<tr>
<td>priority</td>
<td>Message priority, which maps to the X-priority field. 1 is the highest priority and 5 the lowest. The default is 3.</td>
</tr>
<tr>
<td>attachment</td>
<td>RAW attachment</td>
</tr>
<tr>
<td>att_inline</td>
<td>Specifies whether the attachment is viewable inline with the message body, default is TRUE</td>
</tr>
<tr>
<td>att_mime_type</td>
<td>Mime type of the attachment, default is 'application/octet'</td>
</tr>
<tr>
<td>att_filename</td>
<td>String specifying a filename containing the attachment, default is NULL</td>
</tr>
<tr>
<td>replyto</td>
<td>Defines to whom the reply email is to be sent</td>
</tr>
</tbody>
</table>

270.4.3 SEND_ATTACH_VARCHAR2 Procedure

This procedure is the SEND Procedure overloaded for VARCHAR2 attachments.

Syntax

```sql
UTL_MAIL.SEND_ATTACH_VARCHAR2 (sender            IN    VARCHAR2 CHARACTER SET ANY_CS,
recipients        IN    VARCHAR2 CHARACTER SET ANY_CS,
cc                IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
bcc               IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
subject           IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
message           IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
mime_type         IN    VARCHAR2 CHARACTER SET ANY_CS
                     DEFAULT 'text/plain; charset=us-ascii',
priority          IN    PLS_INTEGER DEFAULT 3,
attachment        IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
att_inline        IN    BOOLEAN DEFAULT TRUE,
att_mime_type     IN    VARCHAR2 CHARACTER SET ANY_CS
                     DEFAULT 'text/plain; charset=us-ascii',
att_filename      IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
replyto           IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

Parameters

Table 270-4  SEND_ATTACH_VARCHAR2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender</td>
<td>Email address of the sender</td>
</tr>
<tr>
<td>recipients</td>
<td>Email addresses of the recipient(s), separated by commas</td>
</tr>
<tr>
<td>cc</td>
<td>Email addresses of the CC recipient(s), separated by commas, default is NULL</td>
</tr>
<tr>
<td>bcc</td>
<td>Email addresses of the BCC recipient(s), separated by commas, default is NULL</td>
</tr>
</tbody>
</table>
Table 270-4 (Cont.) SEND_ATTACH_VARCHAR2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject</td>
<td>String to be included as email subject string, default is NULL</td>
</tr>
<tr>
<td>message</td>
<td>Text message body</td>
</tr>
<tr>
<td>mime_type</td>
<td>Mime type of the message, default is 'text/plain; charset=us-ascii'</td>
</tr>
<tr>
<td>priority</td>
<td>Message priority, which maps to the X-priority field. 1 is the highest priority and 5 the lowest. The default is 3.</td>
</tr>
<tr>
<td>attachment</td>
<td>Text attachment</td>
</tr>
<tr>
<td>att_inline</td>
<td>Specifies whether the attachment is inline, default TRUE</td>
</tr>
<tr>
<td>att_mime_type</td>
<td>Mime type of the attachment, default is 'text/plain; charset=us-ascii'</td>
</tr>
<tr>
<td>att_filename</td>
<td>String specifying a filename containing the attachment, default is NULL</td>
</tr>
<tr>
<td>replyto</td>
<td>Defines to whom the reply email is to be sent</td>
</tr>
</tbody>
</table>
The UTL_MATCH package facilitates matching two records. This is typically used to match names, such as two First Names or two Last Names.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of UTL_MATCH Subprograms

### 271.1 UTL_MATCH Overview

UTL_MATCH can use either the Edit Distance algorithm or Jaro-Winkler algorithm when determining matches.

Edit Distance, also known as Levenshtein Distance (named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965), is a measure of similarity between two strings, \( s_1 \) and \( s_2 \). The distance is the number of insertions, deletions or substitutions required to transform \( s_1 \) to \( s_2 \).

The Edit Distance between strings *shackleford* and *shackelford* = 2.

The "Jaro-Winkler algorithm" is another way of calculating Edit distance between two strings. This method, developed at the U.S. Census, is a String Comparator measure that gives values of partial agreement between two strings. The string comparator accounts for length of strings and partially accounts for typical human errors made in alphanumeric strings.

The following table shows similarity values returned by Jaro-Winkler and Edit Distance algorithms:

<table>
<thead>
<tr>
<th>String 1</th>
<th>String 2</th>
<th>Jaro Winkler</th>
<th>Edit Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dunningham</td>
<td>Cunningham</td>
<td>89</td>
<td>80</td>
</tr>
<tr>
<td>Abroms</td>
<td>Abrams</td>
<td>92</td>
<td>83</td>
</tr>
<tr>
<td>Lampley</td>
<td>Campley</td>
<td>90</td>
<td>86</td>
</tr>
<tr>
<td>Marhta</td>
<td>Martha</td>
<td>96</td>
<td>67</td>
</tr>
<tr>
<td>Jonathon</td>
<td>Jonathan</td>
<td>95</td>
<td>88</td>
</tr>
<tr>
<td>Jeraldine</td>
<td>Geraldine</td>
<td>92</td>
<td>89</td>
</tr>
</tbody>
</table>

### 271.2 UTL_MATCH Security Model

The UTL_MATCH package runs with definer's rights. UTL_MATCH must be created under SYS. Operations provided by this package are performed with SYS privileges.
271.3 Summary of UTL_MATCH Subprograms

This table lists the UTL_MATCH subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT_DISTANCE Function</td>
<td>Calculates the number of changes required to transform string-1 into string-2</td>
</tr>
<tr>
<td>EDIT_DISTANCE_SIMILARITY Function</td>
<td>Calculates the number of changes required to transform string-1 into string-2, returning a value between 0 (no match) and 100 (perfect match)</td>
</tr>
<tr>
<td>JARO_WINKLER Function</td>
<td>Calculates the measure of agreement between string-1 and string-2</td>
</tr>
<tr>
<td>JARO_WINKLER_SIMILARITY Function</td>
<td>Calculates the measure of agreement between string-1 and string-2, returning a value between 0 (no match) and 100 (perfect match)</td>
</tr>
</tbody>
</table>

271.3.1 EDIT_DISTANCE Function

This function calculates the number of insertions, deletions or substitutions required to transform string-1 into string-2.

Syntax

```sql
UTL_MATCH.EDIT_DISTANCE ( 
  s1  IN  VARCHAR2, 
  s2  IN  VARCHAR2) 
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>The string to be transformed</td>
</tr>
<tr>
<td>s2</td>
<td>The string into which s1 is to be transformed</td>
</tr>
</tbody>
</table>

Examples

```sql
SELECT UTL_MATCH.EDIT_DISTANCE('shackleford', 'shackelford') FROM DUAL; 
-------------
returns 2
```

271.3.2 EDIT_DISTANCE_SIMILARITY Function

This function calculates the number of insertions, deletions or substations required to transform string-1 into string-2, and returns the Normalized value of the Edit Distance between two strings.

The value is typically between 0 (no match) and 100 (perfect match).
Syntax
UTL_MATCH.EDIT_DISTANCE_SIMILARITY (  
    s1 IN VARCHAR2,  
    s2 IN VARCHAR2) 
RETURN PLS_INTEGER;

Parameters

Table 271-4  EDIT_DISTANCE_SIMILARITY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>The string to be transformed</td>
</tr>
<tr>
<td>s2</td>
<td>The string into which s1 is to be transformed</td>
</tr>
</tbody>
</table>

Examples
SELECT UTL_MATCH.EDIT_DISTANCE_SIMILARITY('shackleford', 'shackelford') FROM DUAL;  
--------------  
returns 82

271.3.3 JARO_WINKLER Function

This function calculates the measure of agreement between two strings.

Syntax
UTL_MATCH.JARO_WINKLER (  
    s1 IN VARCHAR2,  
    s2 IN VARCHAR2) 
RETURN BINARY_DOUBLE;

Parameters

Table 271-5  JARO_WINKLER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Input</td>
</tr>
<tr>
<td>s2</td>
<td>input</td>
</tr>
</tbody>
</table>

Examples
SELECT UTL_MATCH.JARO_WINKLER('shackleford', 'shackelford') FROM DUAL;  
--------------  
returns 9.818E-001
271.3.4 JARO_WINKLER_SIMILARITY Function

This function calculates the measure of agreement between two strings, and returns a score between 0 (no match) and 100 (perfect match).

Syntax

UTL_MATCH.JARO_WINKLER_SIMILARITY (  
  s1 IN VARCHAR2,  
  s2 IN VARCHAR2)  
RETURN PLS_INTEGER;

Parameters

Table 271-6  JARO_WINKLER_SIMILARITY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Input</td>
</tr>
<tr>
<td>s2</td>
<td>input</td>
</tr>
</tbody>
</table>

Examples

SELECT UTL_MATCH.JARO_WINKLER_SIMILARITY('shackleford', 'shackelford') FROM DUAL;  
--------------
returns 98
The UTL_NLA package exposes a subset of the BLAS and LAPACK (Version 3.0) operations on vectors and matrices represented as VARRAYs.

This chapter contains the following topics:

- Overview
- Rules and Limits
- Security Model
- Subprogram Groups
  - BLAS Level 1 (Vector-Vector Operations) Subprograms
  - BLAS Level 2 (Matrix-Vector Operations) Subprograms
  - BLAS Level 3 (Matrix-Matrix Operations) Subprograms
  - LAPACK Driver Routines (Linear Equations) Subprograms
  - LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
- Summary of UTL_NLA Subprograms

272.1 UTL_NLA Overview

The UTL_NLA package exposes a subset of the BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage) (Version 3.0) operations on vectors and matrices represented as VARRAYs.

Standards

For more information on the BLAS and LAPACK standards see

http://www.netlib.org/blas/
http://www.netlib.org/lapack/

Required Expertise

Users of this package are expected to have a sound grasp of linear algebra in general and of the BLAS and LAPACK libraries in particular.

Implementation

The mapping between BLAS and LAPACK procedures and their corresponding PL/SQL calls is one-to-one.

- All BLAS functions have the BLAS prefix (for example, the BLAS_ASUM Functions). The subroutines and functions in BLAS are mapped to PL/SQL procedures and functions, respectively.
• All LAPACK functions have the `LAPACK_` prefix (for example, the `LAPACK_GBSV Procedures`). The subroutines in LAPACK are mapped to PL/SQL procedures. Procedures that perform the same operation but differ only on the datatype of the arguments have the same overloaded names.

The mapping between BLAS and LAPACK procedure parameters and those of their corresponding PL/SQL subprograms is almost one-to-one.

• Also in the PL/SQL interface for LAPACK, all `/work/` arguments have been removed. The `UTL_NLA` package manages the allocation and de-allocation of all work areas required by the libraries.

• A new optional parameter, `pack`, has been added to the end of each LAPACK procedure that specifies if the matrix has been linearized in the row-major or column-major (default) format.

272.2 UTL_NLA Rules and Limits

Vectors and matrices are stored in `VARRAYs` with a maximum size of one million entries. Given this restriction, `UTL_NLA` vectors can be up to one million entries but matrices need to be of size `RxC <= 1,000,000`.

272.3 UTL_NLA Security Model

The `UTL_NLA` package is owned by user `SYS` and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers’ rights (run with the privileges of the current user).

272.4 Subprogram Groups

The `UTL_NLA` package contains subprogram groups for BLAS and LAPACK operations.

• BLAS Level 1 (Vector-Vector Operations) Subprograms
• BLAS Level 2 (Matrix-Vector Operations) Subprograms
• BLAS Level 3 (Matrix-Matrix Operations) Subprograms
• LAPACK Driver Routines (Linear Equations) Subprograms
• LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

272.4.1 UTL_NLA BLAS Level 1 (Vector-Vector Operations) Subprograms

This table lists and briefly describes the UTL_NLA BLAS Level 1 Vector-Vector Operations subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_ASUM Functions</td>
<td>Computes the sum of the absolute values of the vector components</td>
</tr>
</tbody>
</table>
Table 272-1  (Cont.) BLAS Level 1 (Vector-Vector Operations) Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_AXPY Procedures</td>
<td>Copies $\alpha X + Y$ into vector $Y$</td>
</tr>
<tr>
<td>BLAS_COPY Procedures</td>
<td>Copies the contents of vector $X$ to vector $Y$</td>
</tr>
<tr>
<td>BLAS_DOT Functions</td>
<td>Returns the dot (scalar) product of two vectors $X$ and $Y$</td>
</tr>
<tr>
<td>BLAS_IAMAX Functions</td>
<td>Computes the index of the first element of a vector that has the largest absolute value</td>
</tr>
<tr>
<td>BLAS_NRM2 Functions</td>
<td>Computes the vector 2-norm (Euclidean norm)</td>
</tr>
<tr>
<td>BLAS_ROT Procedures</td>
<td>Returns the plane rotation of points</td>
</tr>
<tr>
<td>BLAS_ROTG Procedures</td>
<td>Returns the Givens rotation of points</td>
</tr>
<tr>
<td>BLAS_SCAL Procedures</td>
<td>Scales a vector by a constant</td>
</tr>
<tr>
<td>BLAS_SWAP Procedures</td>
<td>Swaps the contents of two vectors each of size $n$</td>
</tr>
</tbody>
</table>

272.4.2 UTL_NLA BLAS Level 2 (Matrix-Vector Operations) Subprograms

This table lists and briefly describes the UTL_NLA BLAS Level 2 Matrix-Vector Operations subprograms.

Table 272-2  BLAS Level 2 (Matrix-Vector Operations) Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_GBMV Procedures</td>
<td>Performs the matrix-vector operation $y := \alpha A^\top x + beta y$ or $y := \alpha A x + beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are vectors and $A$ is an $m \times n$ band matrix, with $kl$ sub-diagonals and $ku$ super-diagonals</td>
</tr>
<tr>
<td>BLAS_GEMV Procedures</td>
<td>Performs the matrix-vector operations $y := \alpha A^\top x + beta y$ or $y := \alpha A x + beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are vectors and $A$ is an $m \times n$ matrix</td>
</tr>
<tr>
<td>BLAS_GER Procedures</td>
<td>Performs a rank 1 operation $A := \alpha x^\top y' + A$ where $\alpha$ is a scalar, $x$ is an $m$ element vector, $y$ is an $n$ element vector and $A$ is an $m \times n$ matrix</td>
</tr>
<tr>
<td>BLAS_SBMV Procedures</td>
<td>Performs a matrix-vector operation $y := \alpha A x + beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n \times n$ symmetric band matrix, with $k$ super-diagonals</td>
</tr>
<tr>
<td>BLAS_SPMV Procedures</td>
<td>Performs a matrix-vector operation $y := \alpha A x + beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n \times n$ symmetric matrix, supplied in packed form</td>
</tr>
<tr>
<td>BLAS_SPR Procedures</td>
<td>Performs a symmetric rank 1 operation $A := \alpha x^\top x' + A$ where $\alpha$ is a real scalar, $x$ is an $n$ element vector, and $A$ is an $n \times n$ symmetric matrix, supplied in packed form</td>
</tr>
<tr>
<td>BLAS_SPR2 Procedures</td>
<td>Performs a symmetric rank 2 operation $A := \alpha x^\top y' + alpha y^\top x' + A$ where $\alpha$ is a scalar, $x$ and $y$ are $n$ element vectors, and $A$ is an $n \times n$ symmetric matrix, supplied in packed form</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BLAS_SBMV Procedures</td>
<td>Performs a matrix-vector operation $y := \alpha A x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n \times n$ symmetric band matrix, with $k$ super-diagonals</td>
</tr>
<tr>
<td>BLAS_SYMV Procedures</td>
<td>Performs a matrix-vector operation $y := \alpha A x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n \times n$ symmetric matrix</td>
</tr>
<tr>
<td>BLAS_SYR Procedures</td>
<td>Performs a symmetric rank 1 operation $A := \alpha x^* x' + A$ where $\alpha$ is a real scalar, $x$ is an $n$ element vector, and $A$ is an $n \times n$ symmetric matrix</td>
</tr>
<tr>
<td>BLAS_SYR2 Procedures</td>
<td>Performs a symmetric rank 2 operation $A := \alpha x^* y' + \alpha y^* x' + A$ where $\alpha$ is a scalar, $x$ and $y$ are $n$ element vectors, and $A$ is an $n \times n$ symmetric matrix</td>
</tr>
<tr>
<td>BLAS_TBMV Procedures</td>
<td>Performs a matrix-vector operation $x := A^* x$ or $A^* y = b$ where $x$ is an $n$ element vector and $A$ is an $n \times n$ unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals</td>
</tr>
<tr>
<td>BLAS_TBSV Procedures</td>
<td>Solves one of the systems of equation $A^* x = b$ or $A^* y = b$ where $b$ and $x$ are $n$ element vectors and $A$ is an $n \times n$ unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals</td>
</tr>
<tr>
<td>BLAS_TPMV Procedures</td>
<td>Performs a matrix-vector operation $x := A^* x$ or $x := A^* x$ where $x$ is an $n$ element vector and $A$ is an $n \times n$ unit, or non-unit, upper or lower triangular matrix, supplied in packed form</td>
</tr>
<tr>
<td>BLAS_TPSV Procedures</td>
<td>Solves one of the systems of equation $A^* x = b$ or $A^* y = b$ where $b$ and $x$ are $n$ element vectors and $A$ is an $n \times n$ unit, or non-unit, upper or lower triangular matrix, supplied in packed form</td>
</tr>
<tr>
<td>BLAS_TRMV Procedures</td>
<td>Performs a matrix-vector operation $x := A^* x$ or $x := A^* x$ where $x$ is an $n$ element vector and $A$ is an $n \times n$ unit, or non-unit, upper or lower triangular matrix</td>
</tr>
<tr>
<td>BLAS_TRSV Procedures</td>
<td>Solves one of the systems of equation $A^* x = b$ or $A^* x = b$ where $b$ and $x$ are $n$ element vectors and $A$ is an $n \times n$ unit, or non-unit, upper or lower triangular matrix</td>
</tr>
</tbody>
</table>

### 272.4.3 UTL_NLA BLAS Level 3 (Matrix-Matrix Operations) Subprograms

This table lists and briefly describes the UTL_NLA BLAS Level 3 Matrix-Matrix Operations subprograms.
Table 272-3  BLAS Level 3 (Matrix-Matrix Operations) Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_GEMM Procedures</td>
<td>Performs one of the matrix-vector operations ( C := \alpha \text{op}(A) \cdot \text{op}(B) + \beta C ) where ( \text{op}(X) ) is one of ( \text{op}(X) = X ) or ( \text{op}(X) = X' ) where ( \alpha ) and ( \beta ) are scalars, and ( A, B ) and ( C ) are matrices, with ( \text{op}(A) ) an ( m ) by ( k ) matrix, ( \text{op}(B) ) a ( k ) by ( n ) matrix and ( C ) an ( m ) by ( n ) matrix</td>
</tr>
<tr>
<td>BLAS_SYMM Procedures</td>
<td>Performs one of the matrix-vector operations ( C := \alpha A \cdot B + \beta C ) or ( C := \alpha B \cdot A + \beta C ) where ( \alpha ) and ( \beta ) are scalars, ( A ) is a symmetric matrix, and ( B ) and ( C ) are ( m ) by ( n ) matrices</td>
</tr>
<tr>
<td>BLAS_SYR2K Procedures</td>
<td>Performs one of the symmetric rank2 ( k ) operations ( C := \alpha A \cdot B' + \alpha B \cdot A' + \beta C ) or ( C := \alpha A' \cdot B + \alpha B' \cdot A + \beta C ) where ( \alpha ) and ( \beta ) are scalars, ( C ) is an ( n ) by ( n ) symmetric matrix and ( A ) and ( B ) are ( n ) by ( k ) matrices in the first case and ( k ) by ( n ) matrices in the second case</td>
</tr>
<tr>
<td>BLAS_SYRK Procedures</td>
<td>Performs one of the symmetric rank ( k ) operations ( C := \alpha A \cdot A' + \beta C ) or ( C := \alpha A' \cdot A + \beta C ) where ( \alpha ) and ( \beta ) are scalars, ( C ) is an ( n ) by ( n ) symmetric matrix and ( A ) is an ( n ) by ( k ) matrix in the first case and a ( k ) by ( n ) matrix in the second case</td>
</tr>
<tr>
<td>BLAS_TRMM Procedures</td>
<td>Performs one of the matrix-vector operations ( B := \alpha \text{op}(A) \cdot B ) or ( B := \alpha B \cdot \text{op}(A) ) where ( \alpha ) is a scalar, ( B ) is an ( n ) by ( n ) matrix, ( A ) is a unit, or non-unit, upper or lower triangular matrix and ( \text{op}(A) ) is one of two alternatives</td>
</tr>
<tr>
<td>BLAS_TRSM Procedures</td>
<td>Performs one of the matrix-vector operations ( op(A) \cdot X = \alpha B ) or ( X \cdot op(A) = \alpha B ) where ( \alpha ) is a scalar, ( X ) and ( B ) are ( m ) by ( n ) matrices, ( A ) is a unit, or non-unit, upper or lower triangular matrix, ( \text{op}(A) ) is one of two alternatives. The matrix ( X ) is overwritten on ( B )</td>
</tr>
</tbody>
</table>

272.4.4 UTL_NLA LAPACK Driver Routines (Linear Equations) Subprograms

This table lists and briefly describes the LAPACK Driver Routines (Linear Equations) subprograms.

Table 272-4  LAPACK Driver Routines (Linear Equations) Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK_GBSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \cdot x = b ) where ( a ) is an ( n ) by ( n ) matrix and ( x ) and ( b ) are ( n ) by ( \text{nrhs} ) matrices. The ( LU ) decomposition with partial pivoting and row interchanges is used to factor ( A ).</td>
</tr>
<tr>
<td>LAPACK_GESV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \cdot x = b ) where ( a ) is an ( n ) by ( n ) matrix and ( x ) and ( b ) are ( n ) by ( \text{nrhs} ) matrices. The ( LU ) decomposition with partial pivoting and row interchanges is used to factor ( A ).</td>
</tr>
</tbody>
</table>
### Table 272-4  (Cont.) LAPACK Driver Routines (Linear Equations) Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK_GTSV Procedures</td>
<td>This procedure solves the equation ( a \ast x = b ) where ( a ) is an ( n ) by ( n ) tridiagonal matrix, by Gaussian elimination with partial pivoting.</td>
</tr>
<tr>
<td>LAPACK_PBSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \ast x = b ) where ( a ) is an ( n ) by ( n ) symmetric positive definite band matrix and ( x ) and ( b ) are ( n ) by ( nrhs ) matrices. The Cholesky decomposition is used to factor ( A ).</td>
</tr>
<tr>
<td>LAPACK_POSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \ast x = b ) where ( a ) is an ( n ) by ( n ) symmetric positive definite matrix and ( x ) and ( b ) are ( n ) by ( nrhs ) matrices. The Cholesky decomposition is used to factor ( A ).</td>
</tr>
<tr>
<td>LAPACK_PPSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \ast x = b ) where ( a ) is an ( n ) by ( n ) symmetric positive definite matrix stored in packed format and ( x ) and ( b ) are ( n ) by ( nrhs ) matrices. The Cholesky decomposition is used to factor ( A ).</td>
</tr>
<tr>
<td>LAPACK_PTSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \ast x = b ) where ( a ) is an ( n ) by ( n ) symmetric positive definite tridiagonal matrix, and ( x ) and ( b ) are ( n ) by ( nrhs ) matrices.</td>
</tr>
<tr>
<td>LAPACK_SPSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \ast x = b ) where ( a ) is an ( n ) by ( n ) symmetric matrix stored in packed format, and ( x ) and ( b ) are ( n ) by ( nrhs ) matrices. The diagonal pivoting method is used to factor ( A ).</td>
</tr>
<tr>
<td>LAPACK_SYSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( a \ast x = b ) where ( a ) is an ( n ) by ( n ) symmetric matrix, and ( x ) and ( b ) are ( n ) by ( nrhs ) matrices. The diagonal pivoting method is used to factor ( A ).</td>
</tr>
</tbody>
</table>

#### 272.4.5 UTL_NLA LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

This table lists and briefly describes the LAPACK Driver Routines (LLS and Eigenvalue) subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK_GEES Procedures</td>
<td>Computes for an ( n ) by ( n ) real nonsymmetric matrix ( A ), the eigenvalues, the real Schur form ( T ), and, optionally, the matrix of Schur vectors ( Z ). This gives the Schur factorization ( A = Z^{\ast}T(Z^{\ast}) ).</td>
</tr>
<tr>
<td>LAPACK_GEEV Procedures</td>
<td>Computes for an ( n ) by ( n ) real nonsymmetric matrix ( A ), the eigenvalues and, optionally, the left and/or right eigenvectors.</td>
</tr>
<tr>
<td>LAPACK_GELS Procedures</td>
<td>Solves overdetermined or underdetermined real linear systems involving an ( m ) by ( n ) matrix ( A ), or its transpose, using a ( QR ) or ( LQ ) factorization of ( A ). It is assumed that ( A ) has full rank.</td>
</tr>
</tbody>
</table>
Table 272-5 (Cont.) LAPACK Driver Routines (LLS and Eigenvalue Problems)

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK_GESDD</td>
<td>Computes the singular value decomposition (SVD) of a real $m \times n$ matrix $A$, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.</td>
</tr>
<tr>
<td>LAPACK_GESVD</td>
<td>Computes the singular value decomposition (SVD) of a real $m \times n$ matrix $A$, optionally computing the left and/or right singular vectors. The SVD is written $A = U \cdot \Sigma \cdot \text{transpose}(V)$.</td>
</tr>
<tr>
<td>LAPACK_SBEV</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix $A$.</td>
</tr>
<tr>
<td>LAPACK_SBEVD</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.</td>
</tr>
<tr>
<td>LAPACK_SPEV</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ in packed storage.</td>
</tr>
<tr>
<td>LAPACK_SPEVD</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.</td>
</tr>
<tr>
<td>LAPACK_STEV</td>
<td>Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix $A$.</td>
</tr>
<tr>
<td>LAPACK_STEVD</td>
<td>Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix $A$. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.</td>
</tr>
<tr>
<td>LAPACK_SYEV</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$.</td>
</tr>
<tr>
<td>LAPACK_SYEVD</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.</td>
</tr>
</tbody>
</table>

272.5 Summary of UTL_NLA Subprograms

This table lists the UTL_NLA subprograms and briefly describes them.

Table 272-6 UTL_NLA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_ASUM Functions</td>
<td>Computes the sum of the absolute values of the vector components</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_AXPY Procedures</td>
<td>Copies $alpha \cdot X + Y$ into vector $Y$</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_COPY Procedures</td>
<td>Copies the contents of vector $X$ to vector $Y$</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
</tbody>
</table>
Table 272-6  (Cont.) UTL_NLA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_DOT Functions</td>
<td>Returns the dot (scalar) product of two vectors X and Y</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_GBMV Procedures</td>
<td>Performs the matrix-vector operation ( y := \alpha A^t x + \beta y ) or ( y := \alpha A^t x + \beta y ) where ( \alpha ) and ( \beta ) are scalars, ( x ) and ( y ) are vectors and ( A ) is an ( m ) by ( n ) band matrix, with ( k_l ) sub-diagonals and ( k_u ) super-diagonals</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_GEMM Procedures</td>
<td>Performs one of the matrix-vector operations where (\alpha ) and ( \beta ) are scalars, and ( A, B ) and ( C ) are matrices, with ( \operatorname{op}(A) ) an ( m ) by ( k ) matrix, ( \operatorname{op}(B) ) a ( k ) by ( n ) matrix and ( C ) an ( m ) by ( n ) matrix</td>
<td>BLAS Level 3 (Matrix-Matrix Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_GEMV Procedures</td>
<td>Performs the matrix-vector operations ( y := \alpha A^t x + \beta y ) or ( y := \alpha A^t x + \beta y ) where ( \alpha ) and ( \beta ) are scalars, ( x ) and ( y ) are vectors and ( A ) is an ( m ) by ( n ) matrix</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_GER Procedures</td>
<td>Performs a rank 1 operation ( A := \alpha x y^t + A ) where ( \alpha ) is a scalar, ( x ) is an ( m ) element vector, ( y ) is an ( n ) element vector and ( A ) is an ( m ) by ( n ) matrix</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_IAMAX Functions</td>
<td>Computes the index of the first element of a vector that has the largest absolute value</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_NRM2 Functions</td>
<td>Computes the vector 2-norm (Euclidean norm)</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_ROT Procedures</td>
<td>Returns the plane rotation of points</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_ROTG Procedures</td>
<td>Returns the Givens rotation of points</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SBMV Procedures</td>
<td>Performs a matrix-vector operation ( y := \alpha A^t x + \beta y ) where ( \alpha ) and ( \beta ) are scalars, ( x ) and ( y ) are ( n ) element vectors and ( A ) is an ( n ) by ( n ) symmetric band matrix, with ( k ) super-diagonals</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SCAL Procedures</td>
<td>Scales a vector by a constant</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>BLAS_SPMV Procedures</td>
<td>Performs a matrix-vector operation ( y := \alpha A x + \beta y ) where ( \alpha ) and ( \beta ) are scalars, ( x ) and ( y ) are ( n ) element vectors and ( A ) is an ( n \times n ) symmetric matrix, supplied in packed form</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SPR Procedures</td>
<td>Performs a symmetric rank 1 operation ( A := \alpha x x' + A ) where ( \alpha ) is a real scalar, ( x ) is an ( n ) element vector, and ( A ) is an ( n \times n ) symmetric matrix, supplied in packed form</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SPR2 Procedures</td>
<td>Performs a symmetric rank 2 operation where ( \alpha ) is a scalar, ( x ) and ( y ) are ( n ) element vectors, and ( A ) is an ( n \times n ) symmetric matrix, supplied in packed form</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SWAP Procedures</td>
<td>Swaps the contents of two vectors each of size ( n )</td>
<td>BLAS Level 1 (Vector-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SYMM Procedures</td>
<td>Performs one of the matrix-vector operations where ( \alpha ) and ( \beta ) are scalars, ( A ) is a symmetric matrix, and ( B ) and ( C ) are ( m \times n ) matrices</td>
<td>BLAS Level 3 (Matrix-Matrix Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SYMV Procedures</td>
<td>Performs a matrix-vector operation where ( \alpha ) and ( \beta ) are scalars, ( x ) and ( y ) are ( n ) element vectors and ( A ) is an ( n \times n ) symmetric matrix</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SYR Procedures</td>
<td>Performs a symmetric rank 1 operation where ( \alpha ) is a real scalar, ( x ) is an ( n ) element vector, and ( A ) is an ( n \times n ) symmetric matrix</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SYR2 Procedures</td>
<td>Performs a symmetric rank 2 operation where ( \alpha ) is a scalar, ( x ) and ( y ) are ( n ) element vectors, and ( A ) is an ( n \times n ) symmetric matrix</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_SYR2K Procedures</td>
<td>Performs one of the symmetric rank2 ( k ) operations where ( \alpha ) and ( \beta ) are scalars, ( C ) is an ( n \times n ) symmetric matrix and ( A ) and ( B ) are ( n \times k ) matrices in the first case and ( k \times n ) matrices in the second case</td>
<td>BLAS Level 3 (Matrix-Matrix Operations) Subprograms</td>
</tr>
</tbody>
</table>
Table 272-6 (Cont.) UTL_NLA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_SYRK Procedures</td>
<td>Performs one of the symmetric rank ( k ) operations where ( \alpha ) and ( \beta ) are scalars, ( C ) is an ( n ) by ( n ) symmetric matrix and ( A ) is an ( n ) by ( k ) matrix in the first case and a ( k ) by ( n ) matrix in the second case</td>
<td>BLAS Level 3 (Matrix-Matrix Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_TBMV Procedures</td>
<td>Performs a matrix-vector operation where ( x ) is an ( n ) element vector and ( A ) is an ( n ) by ( n ) unit, or non-unit, upper or lower triangular band matrix, with ( (k + 1) ) diagonals</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_TBSV Procedures</td>
<td>Solves one of the systems of equation where ( b ) and ( x ) are ( n ) element vectors and ( A ) is an ( n ) by ( n ) unit, or non-unit, upper or lower triangular band matrix, with ( (k + 1) ) diagonals</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_TPMV Procedures</td>
<td>Performs a matrix-vector operation where ( x ) is an ( n ) element vector and ( A ) is an ( n ) by ( n ) unit, or non-unit, upper or lower triangular matrix, supplied in packed form</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_TPSV Procedures</td>
<td>Solves one of the systems of equation where ( b ) and ( x ) are ( n ) element vectors and ( A ) is an ( n ) by ( n ) unit, or non-unit, upper or lower triangular matrix, supplied in packed form</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_TRMM Procedures</td>
<td>Performs one of the matrix-vector operations where ( \alpha ) is a scalar, ( B ) is an ( m ) by ( n ) matrix, ( A ) is a unit, or non-unit, upper or lower triangular matrix and ( \text{op}(A) ) is one of two alternatives</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_TRMV Procedures</td>
<td>Performs a matrix-vector operation where ( x ) is an ( n ) element vector and ( A ) is an ( n ) by ( n ) unit, or non-unit, upper or lower triangular matrix</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>BLAS_TRSM Procedures</td>
<td>Performs one of the matrix-vector operations ( \text{op}(A) \cdot X = \alpha \cdot B ) or ( X \cdot \text{op}(A) = \alpha \cdot B ), where ( \alpha ) is a scalar, ( X ) and ( B ) are ( m ) by ( n ) matrices, ( A ) is a unit, or non-unit, upper or lower triangular matrix, ( \text{op}(A) ) is one of two alternatives. The matrix ( X ) is overwritten on ( B )</td>
<td>BLAS Level 3 (Matrix-Matrix Operations) Subprograms</td>
</tr>
</tbody>
</table>
### Table 272-6  (Cont.) UTL_NLA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS_TRSV Procedures</td>
<td>Solves one of the systems of equation where ( b ) and ( x ) are ( n ) element vectors and ( A ) is an ( n ) by ( n ) unit, or non-unit, upper or lower triangular matrix.</td>
<td>BLAS Level 2 (Matrix-Vector Operations) Subprograms</td>
</tr>
<tr>
<td>LAPACK_GBSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( A \times x = b ) where ( A ) is an ( n ) by ( n ) matrix and ( x ) and ( b ) are ( n ) by ( n \times r ) matrices. The LU decomposition with partial pivoting and row interchanges is used to factor ( A ).</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td>LAPACK_GEES Procedures</td>
<td>Computes for an ( n ) by ( n ) real nonsymmetric matrix ( A ), the eigenvalues, the real Schur form ( T ), and, optionally, the matrix of Schur vectors ( Z ). This gives the Schur factorization ( A = Z^* T (Z^* T) ).</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td>LAPACK_GEEV Procedures</td>
<td>Computes for an ( n ) by ( n ) real nonsymmetric matrix ( A ), the eigenvalues and, optionally, the left and/or right eigenvectors.</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td>LAPACK_GELS Procedures</td>
<td>Solves overdetermined or underdetermined real linear systems involving an ( m ) by ( n ) matrix ( A ), or its transpose, using a QR or LQ factorization of ( A ). It is assumed that ( A ) has full rank.</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td>LAPACK_GESDD Procedures</td>
<td>Computes the singular value decomposition (SVD) of a real ( m ) by ( n ) matrix ( A ), optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td>LAPACK_GESV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations ( A \times x = b ) where ( A ) is an ( n ) by ( n ) matrix and ( x ) and ( b ) are ( n ) by ( n \times r ) matrices. The LU decomposition with partial pivoting and row interchanges is used to factor ( A ).</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>LAPACK_GESVD</td>
<td>Computes the singular value decomposition (SVD) of a real m by n matrix A, optionally computing the left and/or right singular vectors. The SVD is written $A = U \times \text{SIGMA} \times \text{transpose}(V)$.</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td>LAPACK_GTSV</td>
<td>This procedure solves the equation $a \times x = b$ where $a$ is an n by n tridiagonal matrix, by Gaussian elimination with partial pivoting.</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td>LAPACK_PBSV</td>
<td>This procedure computes the solution to a real system of linear equations $a \times x = b$ where $a$ is an n by n symmetric positive definite band matrix and $x$ and $b$ are n by nrhs matrices. The Cholesky decomposition is used to factor $A$.</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td>LAPACK_POSV</td>
<td>This procedure computes the solution to a real system of linear equations $a \times x = b$ where $a$ is an n by n symmetric positive definite matrix and $x$ and $b$ are n by nrhs matrices. The Cholesky decomposition is used to factor $A$.</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td>LAPACK_PPSV</td>
<td>This procedure computes the solution to a real system of linear equations $a \times x = b$ where $a$ is an n by n symmetric positive definite matrix stored in packed format and $x$ and $b$ are n by nrhs matrices. The Cholesky decomposition is used to factor $A$.</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td>LAPACK_PTSV</td>
<td>This procedure computes the solution to a real system of linear equations $a \times x = b$ where $a$ is an n by n symmetric positive definite tridiagonal matrix, and $x$ and $b$ are n by nrhs matrices.</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td>LAPACK_SBEV</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix $A$.</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td>LAPACK_SBEVD</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
</tbody>
</table>
Table 272-6 (Cont.) UTL_NLA Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK_SPEV Procedures</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td></td>
<td>symmetric matrix $A$ in packed storage</td>
<td></td>
</tr>
<tr>
<td>LAPACK_SPEVD Procedures</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td></td>
<td>symmetric matrix $A$ in packed storage. If eigenvectors are desired, it</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uses a divide and conquer algorithm that makes mild assumptions about</td>
<td></td>
</tr>
<tr>
<td></td>
<td>floating point arithmetic.</td>
<td></td>
</tr>
<tr>
<td>LAPACK_SPSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td></td>
<td>$a \times x = b$ where $a$ is an $n$ by $n$ symmetric matrix stored in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>packed format, and $x$ and $b$ are $n$ by $nrhs$ matrices. The diagonal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pivoting method is used to factor $A$.</td>
<td></td>
</tr>
<tr>
<td>LAPACK_STEV Procedures</td>
<td>Computes all eigenvalues and, optionally, eigenvectors of a real</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td></td>
<td>symmetric tridiagonal matrix $A$.</td>
<td></td>
</tr>
<tr>
<td>LAPACK_STEVD Procedures</td>
<td>Computes all eigenvalues and, optionally, eigenvectors of a real</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td></td>
<td>symmetric tridiagonal matrix $A$. If eigenvectors are desired, it uses a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>divide and conquer algorithm that makes mild assumptions about floating</td>
<td></td>
</tr>
<tr>
<td></td>
<td>point arithmetic.</td>
<td></td>
</tr>
<tr>
<td>LAPACK_SYEVD Procedures</td>
<td>Computes all the eigenvalues and, optionally, eigenvectors of a real</td>
<td>LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</td>
</tr>
<tr>
<td></td>
<td>symmetric matrix $A$. If eigenvectors are desired, it uses a divide and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>conquer algorithm that makes mild assumptions about floating point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>arithmetic.</td>
<td></td>
</tr>
<tr>
<td>LAPACK_SYSV Procedures</td>
<td>This procedure computes the solution to a real system of linear equations</td>
<td>LAPACK Driver Routines (Linear Equations) Subprograms</td>
</tr>
<tr>
<td></td>
<td>$a \times x = b$ where $a$ is an $n$ by $n$ symmetric matrix, and $x$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and $b$ are $n$ by $nrhs$ matrices. The diagonal pivoting method is used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to factor $A$.</td>
<td></td>
</tr>
</tbody>
</table>
272.5.1 BLAS_ASUM Functions

This procedure computes the sum of the absolute values of the vector components.

See Also:
BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group

Syntax

UTL_NLA.BLAS_ASUM (
    n IN POSITIVEN,
    x IN UTL_NLA_ARRAY_DBL,
    incx IN POSITIVEN)
RETURN BINARY_DOUBLE;

UTL_NLA.BLAS_ASUM (
    n IN POSITIVEN,
    alpha IN SCALAR_DOUBLE,
    x IN UTL_NLA_ARRAY_FLT)
RETURN BINARY_FLOAT

Parameters

Table 272-7  BLAS_ASUM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAYFLT/DBL of dimension at least (1 + (n - 1)*abs(incx))</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
</tbody>
</table>

272.5.2 BLAS_AXPY Procedures

This procedure copies alpha*X + Y into vector Y.

See Also:
BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group
Syntax

UTL_NLA.BLAS_AXPY (  
  n      IN       POSITIVEN,  
  alpha  IN       SCALAR_DOUBLE,  
  x      IN       UTL_NLA_ARRAY_DBL,  
  incx   IN       POSITIVEN,  
  y      IN OUT   UTL_NLA_ARRAY_DBL,  
  incy   IN       POSITIVEN);  

UTL_NLA.BLAS_AXPY (  
  n      IN       POSITIVEN,  
  alpha  IN       SCALAR_DOUBLE,  
  x      IN       UTL_NLA_ARRAY_FLT,  
  incx   IN       POSITIVEN,  
  y      IN OUT   UTL_NLA_ARRAY_FLT,  
  incy   IN       POSITIVEN);  

Parameters

Table 272-8  BLAS_AXPY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>Specifies the scalar alpha.</td>
</tr>
</tbody>
</table>
| x         | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
  \( 1 + ( n - 1 )*\text{abs}( \text{incx} ) \) |
| incx      | Specifies the increment for the elements of x. incx must not be zero. |
| y         | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
  \( 1 + ( n - 1 )*\text{abs}( \text{incy} ) \) |
| incy      | Specifies the increment for the elements of y. incy must not be zero. |

272.5.3 BLAS_COPY Procedures

This procedure copies the contents of vector x to vector y.

See Also:  
BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group

Syntax

UTL_NLA.BLAS_COPY (  
  n IN POSITIVEN,  
  x IN UTL_NLA_ARRAY_DBL,  
  incx IN POSITIVEN,  
)
y     IN OUT UTL_NLA_ARRAY_DBL,
incy IN     POSITIVEN);

UTL_NLA.BLAS_COPY (
    n     IN     POSITIVEN,
    x     IN     UTL_NLA_ARRAY_FLT,
    incx IN     POSITIVEN,
    y     IN OUT UTL_NLA_ARRAY_FLT,
    incy IN     POSITIVEN);

Parameters

Table 272-9  BLAS_COPY Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DLB of dimension at least (1 + (n - 1) *abs{incx})</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAY_FLT/DLB of dimension at least (1 + (n - 1) *abs{incy})</td>
</tr>
<tr>
<td>incy</td>
<td>Specifies the increment for the elements of y. incy must not be zero.</td>
</tr>
</tbody>
</table>

272.5.4 BLAS_DOT Functions

This function returns the dot (scalar) product of two vectors x and y.

See Also:

BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group

Syntax

UTL_NLA.BLAS_DOT (
    n     IN     POSITIVEN,
    x     IN     UTL_NLA_ARRAY_DBLO,
    incx IN     POSITIVEN,
    y     IN     UTL_NLA_ARRAY_DBLO,
    incy IN     POSITIVEN)
RETURN BINARY_DOUBLE;

UTL_NLA.BLAS_DOT (
    n     IN     POSITIVEN,
    x     IN     UTL_NLA_ARRAY_FLT,
    incx IN     POSITIVEN,
Parameters

Table 272-10  BLAS_DOT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least ( (1 + (n - 1) \times \text{abs}(\text{incy}) )</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least ( (1 + (n - 1) \times \text{abs}(\text{incy}) )</td>
</tr>
<tr>
<td>incy</td>
<td>Specifies the increment for the elements of y. incy must not be zero.</td>
</tr>
</tbody>
</table>

272.5.5 BLAS_GBMV Procedures

This procedure performs one of the matrix-vector operations

\[
y := \alpha A^T x + \beta y \quad \text{or} \quad y := \alpha A^T x + \beta y
\]

where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are vectors and \( A \) is an \( m \) by \( n \) band matrix, with \( k_1 \) sub-diagonals and \( k_u \) super-diagonals.

See Also:
BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_GBMV (  
  trans IN      flag,  
  m      IN      POSITIVEN,   n      IN      POSITIVEN,  
  kl     IN      NATURALN,  
  ku     IN      NATURALN,  
  alpha IN      SCALAR_DOUBLE,  
  a      IN      UTL_NLA_ARRAY_DBL,  
  lda IN      POSITIVEN,  
  x IN      UTL_NLA_ARRAY_DBL,  
  incx IN      POSITIVEN,  
  beta IN      SCALAR_DOUBLE,  
  y IN OUT    UTL_NLA_ARRAY_DBL,  
  incy IN      POSITIVEN,  
  pack IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_GBMV (  
  trans IN      flag,  
  m      IN      POSITIVEN,  
```

Parameters

Table 272-11  BLAS_GBMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| trans     | Specifies the operation to be performed:  
  - trans = 'N' or 'n': y := alpha*A*x + beta*y  
  - trans = 'T' or 't': y := alpha*A'*x + beta*y  
  - trans = 'C' or 'c': y := alpha*A'*x + beta*y |
| m         | Specifies the number of rows of the matrix A. m must be at least zero. |
| n         | Specifies the number of columns of the matrix A. n must be at least zero. |
| kl        | Specifies the number of sub-diagonals of the matrix A. kl must satisfy 0. le. kl. |
| ku        | Specifies the number of super-diagonals of the matrix A. ku must satisfy 0 .le. ku. |
| alpha     | SCALAR_FLOAT/DDOUBLE. Specifies the scalar alpha. |
| a         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku+1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku+2), and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced. |
| lda       | Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (kl+ku+1). |
| x         | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
  - (1 + ( n - 1 )*abs( incx ) )  
  - (1 + ( m - 1 )*abs( incx ) )  
  otherwise. Before entry, the incremented array X must contain the vector x. |
### Table 272-11  (Cont.) BLAS_GBMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. Must not be zero.</td>
</tr>
<tr>
<td>beta</td>
<td>SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.</td>
</tr>
</tbody>
</table>
| y         | UTL_NLA_ARRAY_FLT/DBL of dimension at least 
|           | ( 1 + ( m - 1 )*abs( incy ) ) |
|           | when trans = 'N' or 'n' and at least 
|           | (1+(n-1)*abs(incy)) |
|           | otherwise. Before entry with beta non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y. |
| incy      | Specifies the increment for the elements of y. Must not be zero. |
| pack      | (Optional) Flags the packing of the matrices: |
|           | · 'C': column-major (default) |
|           | · 'R': row-major |

#### 272.5.6 BLAS_GEMM Procedures

This procedure performs one of the matrix-matrix operations.

\[ C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C \]

where \( \text{op}(X) \) is one of

\( \text{op}(X) = X \)

or

\( \text{op}(X) = X' \)

where \( \alpha \) and \( \beta \) are scalars, and \( A, B \) and \( C \) are matrices, with \( \text{op}(A) \) an \( m \) by \( k \) matrix, \( \text{op}(B) \) a \( k \) by \( n \) matrix and \( C \) an \( m \) by \( n \) matrix.

**See Also:**

[BLAS Level 3 (Matrix-Matrix Operations) Subprograms](#) for other subprograms in this group

**Syntax**

```sql
UTL_NLA.BLAS_GEMM ( 
    transa IN      flag, 
    transb IN      flag, 
    m      IN      POSITIVEN, 
    n      IN      POSITIVEN, 
    k      IN      POSITIVEN,
)```

ORACLE
Parameters

Table 272-12  BLAS_GEMM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| transa    | Specifies the form of \( \text{op}(A) \) to be used in the matrix multiplication as follows:  
  - \( \text{transa} = 'N' \) or \( 'n' \): \( \text{op}(A) = 'A' \)  
  - \( \text{transa} = 'T' \) or \( 't' \): \( \text{op}(A) = 'A' \)  
  - \( \text{transa} = 'C' \) or \( 'c' \): \( \text{op}(A) = 'A' \) |
| transb    | Specifies the form of \( \text{op}(B) \) to be used in the matrix multiplication as follows:  
  - \( \text{transb} = 'N' \) or \( 'n' \): \( \text{op}(B) = B \)  
  - \( \text{transb} = 'T' \) or \( 't' \): \( \text{op}(B) = B' \)  
  - \( \text{transb} = 'C' \) or \( 'c' \): \( \text{op}(B) = B' \) |
| m         | Specifies the number of rows of the matrix \( \text{op}(A) \) and of the matrix \( C \). \( m \) must be at least zero. |
| n         | Specifies the number of columns of the matrix \( \text{op}(B) \) and of the matrix \( C \). \( n \) must be at least zero. |
| k         | Specifies the rows of the matrix \( \text{op}(A) \) and the number of columns of the matrix \( \text{op}(B) \). \( k \) must be at least zero. |
| alpha     | SCALAR_FLOAT/Doubles. Specifies the scalar alpha. |
| a         | UTL_NLA_ARRAY_FLT/DOUBLE of DIMENSION (lda, ka) where \( ka \) is \( k \) when \( \text{transa} = 'N' \) or \( 'n' \), and is \( m \) otherwise. Before entry with \( \text{transa} = 'N' \) or \( 'n' \), the leading \( m \) by \( k \) part of the array \( A \) must contain the matrix \( A \), otherwise the leading \( k \) by \( m \) part of the array \( A \) must contain the matrix \( A \). |
Table 272-12  (Cont.) BLAS_GEMM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lda</td>
<td>Specifies the first dimension of a as declared in the calling (sub) program. When transa = 'N' or 'n', lda must be at least max (1,k).</td>
</tr>
<tr>
<td>b</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, kb) where kb is n when transb = 'N' or 'n', and is k otherwise. Before entry with transb = 'N' or 'n', the leading k by n part of the array b must contain the matrix B, otherwise the leading n by k part of the array b must contain the matrix B.</td>
</tr>
<tr>
<td>ldb</td>
<td>Specifies the first dimension of b as declared in the calling (sub) program. When transb = 'N' or 'n', ldb must be at least max (1, n).</td>
</tr>
<tr>
<td>beta</td>
<td>SCALAR_FLOAT/DOPBLE. Specifies the scalar beta. When beta is supplied as zero then c need not be set on input.</td>
</tr>
<tr>
<td>c</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc, n). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)<em>op(B) + beta</em>C).</td>
</tr>
<tr>
<td>ldc</td>
<td>Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least max(1, m).</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices: • 'C': column-major (default) • 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.7 BLAS_GEMV Procedures

This procedure performs one of the matrix-vector operations: y := alpha*A*x + beta*y or y := alpha*A'*x + beta*y where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```plaintext
UTL_NLA.BLAS_GEMV ( 
  trans IN     flag,
  m      IN     POSITIVEN,
  n      IN     POSITIVEN,
  alpha IN    SCALAR_DOUBLE,
  a      IN     UTL_NLA_ARRAY_DBL,
  lda    IN     POSITIVEN,
  x      IN     UTL_NLA_ARRAY_DBL,
  incx   IN     POSITIVEN,
```

Chapter 272
Summary of UTL_NLA Subprograms
beta IN SCALAR_DOUBLE,
y IN OUT UTL_NLA_ARRAY_DBL,
incy IN POSITIVEN,
pack IN flag DEFAULT 'C');

UTL_NLA.BLAS_GEMV (  
  trans IN flag,  
  m IN POSITIVEN,  
  n IN POSITIVEN,  
  alpha IN SCALAR_FLOAT,  
  a IN UTL_NLA_ARRAY_FLT,  
  lda IN POSITIVEN,  
  x IN UTL_NLA_ARRAY_FLT,  
  incx IN POSITIVEN,  
  beta IN SCALAR_FLOAT,  
  y IN OUT UTL_NLA_ARRAY_FLT,  
  incy IN POSITIVEN,  
  pack IN flag DEFAULT 'C');

Parameters

Table 272-13  BLAS_GEMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| trans     | Specifies the operation to be performed:  
    - trans = 'N' or 'n', \( y := \alpha A \times x + \beta y \)  
    - trans = 'T' or 't', \( y := \alpha A^\top \times x + \beta y \)  
    - trans = 'C' or 'c', \( y := \alpha A^\top \times x + \beta y \)  
| m         | Specifies the number of rows of the matrix A. \( m \) must be at least zero.  
| n         | Specifies the number of columns of the matrix A. \( n \) must be at least zero.  
| alpha     | SCALAR_FLOAT/Doubles. Specifies the scalar \( \alpha \).  
| a         | UTL_NLA_ARRAY_FLT/DBL of \text{DIMENSION}\ (\text{lda}, n). Before entry,  
    the leading \( m \) by \( n \) part of the array \( a \) must contain the matrix of  
    coefficients.  
| lda       | Specifies the first dimension of \( a \) as declared in the calling (sub)  
    program. \( lda \) must be at least \( \max(1, m) \).  
| x         | UTL_NLA_ARRAY_FLT/DBL of \text{DIMENSION} at least  
    \( (1 + (n - 1) \times \text{abs} (\text{incx})) \)  
    when \( \text{trans} = 'N' \) or 'n' and at least  
    \( (1 + (m - 1) \times \text{abs} (\text{incx})) \)  
    otherwise. Before entry, the incremented array \( x \) must contain the  
    vector \( x \).  
| incx      | Specifies the increment for the elements of \( x \). Must not be zero.  
| beta      | SCALAR_FLOAT/Doubles. Specifies the scalar \( \beta \). When \( \beta \) is  
    supplied as zero then \( y \) need not be set on input.
Table 272-13 (Cont.) BLAS_GEMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAY_FLT of dimension at least ((1 + (m - 1) \times \text{abs}(\text{incy}))) when (\text{trans} = 'N') or (n') and at least ((1 + (n - 1) \times \text{abs}(\text{incy}))) otherwise. Before entry with (\beta) nonzero, the incremented array (Y) must contain the vector (y). On exit, (Y) is overwritten by the updated vector (y).</td>
</tr>
</tbody>
</table>

| incy      | Specifies the increment for the elements of \(y\). Must not be zero. |
| pack      | (Optional) Flags the packing of the matrices: |
|           | · 'C': column-major (default) |
|           | · 'R': row-major |

272.5.8 BLAS_GER Procedures

This procedure performs the rank 1 operation: 
\[ A := \alpha x^T y' + A \]
where \(\alpha\) is a scalar, \(x\) is an \(m\) element vector, \(y\) is an \(n\) element vector and \(A\) is an \(m\) by \(n\) matrix.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_GER (  
m IN POSITIVEN,  
n IN POSITIVEN,  
alpha IN SCALAR_DBL,  
x IN OUT UTL_NLA_ARRAY_DBL,  
incx IN POSITIVEN,  
y IN UTL_NLA_ARRAY_DBL,  
incy IN POSITIVEN,  
a IN OUT UTL_NLA_ARRAY_DBL,  
lda IN POSITIVEN,  
pack IN flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_GER (  
m IN POSITIVEN,  
n IN POSITIVEN,  
alpha IN SCALAR_FLT,  
x IN OUT UTL_NLA_ARRAY_FLT,  
incx IN POSITIVEN,  
y IN UTL_NLA_ARRAY_FLT,  
incy IN POSITIVEN,  
a IN OUT UTL_NLA_ARRAY_FLT,
```
Parameters

Table 272-14  BLAS_GER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Specifies the number of rows of the matrix A. m must be at least zero.</td>
</tr>
<tr>
<td>n</td>
<td>Specifies the number of columns of the matrix A. n must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>Specifies the scalar alpha.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAYFLT of dimension at least (1 + (m - 1) \times \text{abs}(\text{incx})). Before entry, the incremented array x must contain the m element vector x.</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAYFLT of dimension at least (1 + (n - 1) \times \text{abs}(\text{incy})). Before entry, the incremented array y must contain the m element vector y.</td>
</tr>
<tr>
<td>incy</td>
<td>Specifies the increment for the elements of y. incy must not be zero.</td>
</tr>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAYFLT of dimension (lda, n). Before entry, the leading m by n part of the array a must contain the matrix of coefficients. On exit, a is overwritten by the updated matrix.</td>
</tr>
<tr>
<td>lda</td>
<td>Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (\max(1, m)).</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>• 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>• 'R': row-major</td>
</tr>
</tbody>
</table>
272.5.9 BLAS_IAMAX Functions

This function computes the index of first element of a vector that has the largest absolute value.

See Also:

BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_IAMAX (  
   n IN POSITIVEN,  
   x IN UTL_NLA_ARRAY_DBL,  
   incx IN POSITIVEN,  
RETURN POSITIVEN;
);
```

```
UTL_NLA.BLAS_IAMAX (  
   n IN POSITIVEN,  
   x IN UTL_NLA_ARRAY_FLT,  
   incx IN POSITIVEN,  
RETURN POSITIVEN;
);
```

Parameters

Table 272-15  BLAS_IAMAX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least</td>
</tr>
<tr>
<td></td>
<td>( 1 + ( n - 1 )*abs( incx ) )</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
</tbody>
</table>

272.5.10 BLAS_NRM2 Functions

This function computes the vector 2-norm (Euclidean norm).

See Also:

BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group
Syntax

UTL_NLA.BLAS_NRM2 (  
   n IN   POSITIVEN,  
   x IN   UTL_NLA_ARRAY_DBL,  
   incx IN   POSITIVEN)  
RETURN BINARY_DOUBLE;

UTL_NLA.BLAS_NRM2 (  
   n IN   POSITIVEN,  
   x IN   UTL_NLA_ARRAY_FLT,  
   incx IN   POSITIVEN)  
RETURN BINARY_FLOAT;

Parameters

Table 272-16   BLAS_NRM2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
</tbody>
</table>
| x         | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
             ( 1 + ( n - 1 )*abs( incx ) ) |
| incx      | Specifies the increment for the elements of x. incx must not be zero. |

272.5.11 BLAS_ROT Procedures

This procedure returns the plane rotation of points.

See Also:

BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group

Syntax

UTL_NLA.BLAS_ROT (  
   n IN   POSITIVEN,  
   x IN OUT UTL_NLA_ARRAY_DBL,  
   incx IN   POSITIVEN,  
   y IN OUT UTL_NLA_ARRAY_DBL,  
   incy IN   POSITIVEN,  
   c IN   SCALAR_DOUBLE,  
   s IN   SCALAR_DOUBLE);  

UTL_NLA.BLAS_ROT (  
   n IN   POSITIVEN,  
   x IN OUT UTL_NLA_ARRAY_FLT,  
   incx IN   POSITIVEN,  
   y IN OUT UTL_NLA_ARRAY_FLT,  
   incy IN   POSITIVEN,  
   c IN   SCALAR_DOUBLE,  
   s IN   SCALAR_DOUBLE);
Parameters

Table 272-17  BLAS_ROT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least ((1+(n-1)^*\text{abs}(\text{incx})))</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least ((1+(n-1)^*\text{abs} (\text{incy})))</td>
</tr>
<tr>
<td>incy</td>
<td>Specifies the increment for the elements of y. incy must not be zero.</td>
</tr>
<tr>
<td>c</td>
<td>SCALAR_FLOAT/DOUBLE. Specifies the scalar C.</td>
</tr>
<tr>
<td>s</td>
<td>SCALAR_FLOAT/DOUBLE. Specifies the scalar S.</td>
</tr>
</tbody>
</table>

272.5.12 BLAS_ROTG Procedures

This procedure returns the Givens rotation of points.

See Also:

BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group
Parameters

Table 272-18  BLAS_ROTG Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>SCALAR_FLOAT/Dbl. Specifies the scalar A.</td>
</tr>
<tr>
<td>b</td>
<td>SCALAR_FLOAT/Dbl. Specifies the scalar B.</td>
</tr>
<tr>
<td>c</td>
<td>SCALAR_FLOAT/Dbl. Specifies the scalar C.</td>
</tr>
<tr>
<td>s</td>
<td>SCALAR_FLOAT/Dbl. Specifies the scalar S.</td>
</tr>
</tbody>
</table>

272.5.13 BLAS_SCAL Procedures

This procedure scales a vector by a constant.

See Also:

BLAS Level 1 (Vector-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SCAL (
    n      IN  POSITIVEN,
    alpha  IN  SCALAR_DOUBLE,
    x      IN  OUT UTL_NLA_ARRAY_DBL,
    incx   IN  POSITIVEN);
```

```
UTL_NLA.BLAS_SCAL (
    n      IN  POSITIVEN,
    alpha  IN  SCALAR_FLOAT,
    x      IN  OUT UTL_NLA_ARRAY_FLT,
    incx   IN  POSITIVEN);
```

Parameters

Table 272-19  BLAS_SCAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>Specifies the scalar alpha.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)\times\text{abs}(\text{incx}))</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
</tbody>
</table>
272.5.14 BLAS_SPMV Procedures

This procedure performs the matrix-vector operation \( y := \alpha A x + \beta y \), where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are \( n \) element vectors and \( A \) is an \( n \) by \( n \) symmetric matrix, supplied in packed form.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

\[
\text{UTL_NLA.BLAS_SPMV (}
\begin{array}{ll}
\text{uplo} & \text{IN} \quad \text{flag}, \\
\text{n} & \text{IN} \quad \text{POSITIVEN}, \\
\text{alpha} & \text{IN} \quad \text{SCALAR\_DOUBLE}, \\
\text{ap} & \text{IN} \quad \text{UTL\_NLA\_ARRAY\_DBL}, \\
\text{x} & \text{IN} \quad \text{UTL\_NLA\_ARRAY\_DBL}, \\
\text{incx} & \text{IN} \quad \text{POSITIVEN}, \\
\text{beta} & \text{IN} \quad \text{SCALAR\_DOUBLE}, \\
\text{y} & \text{IN\ OUT} \quad \text{UTL\_NLA\_ARRAY\_DBL}, \\
\text{incy} & \text{IN} \quad \text{POSITIVEN}, \\
\text{pack} & \text{IN} \quad \text{flag DEFAULT 'C');}
\end{array}
\]

\[
\text{UTL_NLA.BLAS_SPMV (}
\begin{array}{ll}
\text{uplo} & \text{IN} \quad \text{flag}, \\
\text{n} & \text{IN} \quad \text{POSITIVEN}, \\
\text{alpha} & \text{IN} \quad \text{SCALAR\_FLOAT}, \\
\text{ap} & \text{IN} \quad \text{UTL\_NLA\_ARRAY\_FLT}, \\
\text{x} & \text{IN} \quad \text{UTL\_NLA\_ARRAY\_FLT}, \\
\text{incx} & \text{IN} \quad \text{POSITIVEN}, \\
\text{beta} & \text{IN} \quad \text{SCALAR\_FLOAT}, \\
\text{y} & \text{IN\ OUT} \quad \text{UTL\_NLA\_ARRAY\_FLT}, \\
\text{incy} & \text{IN} \quad \text{POSITIVEN}, \\
\text{pack} & \text{IN} \quad \text{flag DEFAULT 'C');}
\end{array}
\]

Parameters

Table 272-20  BLAS_SPMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies the upper or lower triangular part of the matrix \( A \) is supplied in the packed array \( AP \):  
  - uplo = 'U' or 'u'. The upper triangular part of \( A \) is supplied in \( AP \).  
  - uplo = 'L' or 'l'. The lower triangular part of \( A \) is supplied in \( AP \).  
| n         | Specifies the order of the matrix \( A \). \( n \) must be at least zero.  
| alpha     | SCALAR\_FLOAT\_DOUBLE. Specifies the scalar alpha. |
Table 272-20  (Cont.) BLAS_SPMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ap</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least ((n^2+n+1)/2)</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'U' or 'u', the array ap must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(1,2) and a(2,2) respectively, and so on.</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'L' or 'l', the array ap must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains, ap(2) and ap(3) contain a(2,1) and a(3,1) respectively, and so on.</td>
</tr>
<tr>
<td><strong>x</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least ((1+(n-1)*abs(incx)))</td>
</tr>
<tr>
<td></td>
<td>Before entry, the incremented array X must contain the n element vector x.</td>
</tr>
<tr>
<td><strong>incx</strong></td>
<td>Specifies the increment for the elements of x. Must not be zero.</td>
</tr>
<tr>
<td><strong>beta</strong></td>
<td>SCALAR_FLOAT/Doubles. Specifies the scalar beta. When beta is supplied as zero then Y need not be set on input.</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least ((1+(n-1)*abs(incy)))</td>
</tr>
<tr>
<td></td>
<td>Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.</td>
</tr>
<tr>
<td><strong>incy</strong></td>
<td>Specifies the increment for the elements of y. Must not be zero.</td>
</tr>
<tr>
<td><strong>pack</strong></td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>• 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>• 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.15 BLAS_SPR Procedures

This procedure performs the rank 1 operation \( A := \alpha x^T x + A \), where \( \alpha \) is a real scalar, \( x \) is an \( n \) element vector, and \( A \) is an \( n \times n \) symmetric matrix, supplied in packed form.

**See Also:**

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

**Syntax**

```
UTL_NLA.BLAS_SPR ( 
    uplo IN flag,
```

Parameters

**Table 272-21** BLAS_SPR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>Specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array ap:</td>
</tr>
<tr>
<td></td>
<td>· uplo = 'U' or 'u': The upper triangular part of A is supplied in ap.</td>
</tr>
<tr>
<td></td>
<td>· uplo = 'L' or 'l': The lower triangular part of A is supplied in ap.</td>
</tr>
<tr>
<td>n</td>
<td>Specifies the order of the matrix A. n must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>Specifies the scalar alpha.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least</td>
</tr>
<tr>
<td></td>
<td>(1+(n-1)*abs(incx))</td>
</tr>
<tr>
<td></td>
<td>Before entry, the incremented array X must contain the m element vector x.</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
<tr>
<td>ap</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least</td>
</tr>
<tr>
<td></td>
<td>((n*(n +1))/2)</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'U' or 'u', the array ap must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(1,2) and a(2,2) respectively, and so on. On exit, the array ap is overwritten by the upper triangular part of the updated matrix.</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'L' or 'l', the array ap must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(2,1) and a(3,1) respectively, and so on. On exit, the array ap is overwritten by the lower triangular part of the updated matrix.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>· 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>· 'R': row-major</td>
</tr>
</tbody>
</table>
272.5.16 BLAS_SPR2 Procedures

This procedure performs the rank 2 operation \( A := \alpha x' y' + \alpha y' x' + A \), where \( \alpha \) is a scalar, \( x \) and \( y \) are \( n \) element vectors, and \( A \) is an \( n \) by \( n \) symmetric matrix, supplied in packed form.

See Also:
BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SPR2 ( 
  uplo   IN      flag,
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_DBL,
  x      IN      UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  y      IN      UTL_NLA_ARRAY_DBL,
  incy   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SPR2 ( 
  uplo   IN      flag,
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_FLT,
  x      IN      UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  y      IN      UTL_NLA_ARRAY_FLT,
  incy   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_FLT,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

Parameters

Table 272-22  BLAS_SPR2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies whether the upper or lower triangular part of the matrix \( A \) is supplied in the packed array \( ap \):  
  * \( uplo = 'U' \) or \( 'u' \): The upper triangular part of \( A \) is supplied in \( ap \).  
  * \( uplo = 'L' \) or \( 'l' \): The lower triangular part of \( A \) is supplied in \( ap \).  
| n         | Specifies the order of the matrix \( A \). \( n \) must be at least zero.  
| alpha     | Specifies the scalar \( \alpha \).  |
### Table 272-22 (Cont.) BLAS_SPR2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| x         | UTL_NLA_ARRAY_FLT/DBL of dimension at least \((1+(n-1)\times\text{abs}(\text{incx}))\)  
Before entry, the incremented array \(X\) must contain the \(m\) element vector \(x\).  
\text{incx} specifies the increment for the elements of \(x\). \(\text{incx}\) must not be zero.  
\(y\) | UTL_NLA_ARRAY_FLT/DBL of dimension at least \((1+(n-1)\times\text{abs}(\text{incy}))\)  
Before entry, the incremented array \(X\) must contain the \(m\) element vector \(y\).  
\(\text{incy}\) specifies the increment for the elements of \(y\). \(\text{incy}\) must not be zero.  
\(\text{ap}\) | UTL_NLA_ARRAY_FLT/DBL of dimension at least \(((n\times(n+1))/2)\)  
Before entry with \(\text{uplo} = 'U'\) or \( 'u'\), the array \(\text{ap}\) must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that \(\text{ap}(1)\) contains \(a(1,1)\), \(\text{ap}(2)\) and \(\text{ap}(3)\) contain \(a(1,2)\) and \(a(2,2)\) respectively, and so on. On exit, the array \(\text{ap}\) is overwritten by the upper triangular part of the updated matrix.  
Before entry with \(\text{uplo} = 'L'\) or \( 'l'\), the array \(\text{ap}\) must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that \(\text{ap}(1)\) contains \(a(1,1)\), \(\text{ap}(2)\) and \(\text{ap}(3)\) contain \(a(2,1)\) and \(a(3,1)\) respectively, and so on. On exit, the array \(\text{ap}\) is overwritten by the lower triangular part of the updated matrix.  
\(\text{lda}\) | Specifies the first dimension of \(a\) as declared in the calling (sub) program. \(\text{lda}\) must be at least \((k + 1)\).  
pack | (Optional) Flags the packing of the matrices:  
- \('C'\): column-major (default)  
- \('R'\): row-major
272.5.17 BLAS_SBMV Procedures

This procedure performs the matrix-vector operation \( y := \alpha A x + \beta y \), where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are \( n \) element vectors and \( A \) is an \( n \times n \) symmetric band matrix, with \( k \) super-diagonals.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_SBMV (  
   uplo   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   alpha IN      SCALAR_DOUBLE,
   a      IN      UTL_NLA_ARRAY_DBL,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_DBL,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_DOUBLE,
   y      IN OUT  UTL_NLA_ARRAY_DBL,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SBMV (  
   uplo   IN      flag,
   n      IN      POSITIVEN,
   k      IN      NATURALN,
   alpha IN      SCALAR_FLOAT,
   a      IN      UTL_NLA_ARRAY_FLT,
   lda    IN      POSITIVEN,
   x      IN      UTL_NLA_ARRAY_FLT,
   incx   IN      POSITIVEN,
   beta   IN      SCALAR_FLOAT,
   y      IN OUT  UTL_NLA_ARRAY_FLT,
   incy   IN      POSITIVEN,
   pack   IN      flag DEFAULT 'C');
```

Parameters

Table 272-23  BLAS_SBMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies whether the upper or lower triangular part of the band matrix \( A \) is being supplied:  
  - uplo = 'U' or 'u'. The upper triangular part of \( A \) is supplied.  
  - uplo = 'L' or 'l'. The lower triangular part of \( A \) is supplied. |
| n         | Specifies the order of the matrix \( A \). \( n \) must be at least zero. |
Table 272-23  (Cont.) BLAS_SBMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>Specifies the number of super-diagonals of the matrix A. k must satisfy 0 .le. k.</td>
</tr>
<tr>
<td>alpha</td>
<td>SCALAR_FLOAT/DATA. Specifies the scalar alpha.</td>
</tr>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Unchanged on exit</td>
</tr>
<tr>
<td>lda</td>
<td>Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (k + 1).</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)) Before entry, the incremented array X must contain the n element vector x.</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. Must not be zero.</td>
</tr>
<tr>
<td>beta</td>
<td>SCALAR_FLOAT/DATA. Specifies the scalar beta.</td>
</tr>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incy)) Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.</td>
</tr>
<tr>
<td>incy</td>
<td>Specifies the increment for the elements of y. Must not be zero.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:  • 'C': column-major (default)  • 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.18 BLAS_SWAP Procedures

This procedure swaps the contents of two vectors each of size n.

Syntax

```nalu
UTL_NLA.BLAS_SWAP ( n IN POSITIVEN, x IN OUT UTL_NLA_ARRAY_DBL, incx IN POSITIVEN, )
```
y    IN OUT UTL_NLA_ARRAY_DBL,
incy IN     POSITIVEN);

UTL_NLA.BLAS_SWAP (
  n IN     POSITIVEN,
  x IN OUT UTL_NLA_ARRAY_FLT,
  incx IN     POSITIVEN,
  y IN OUT UTL_NLA_ARRAY_FLT,
  incy IN     POSITIVEN);

Parameters

Table 272-24  BLAS_SWAP Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Specifies the number of elements of the vectors x and y. n must be at least zero.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT of dimension at least (1+(n-1)*abs(incx))</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. incx must not be zero.</td>
</tr>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAY_FLT of DIMENSION at least (1+(n-1)*abs(incy))</td>
</tr>
<tr>
<td>incy</td>
<td>Specifies the increment for the elements of y. incy must not be zero.</td>
</tr>
</tbody>
</table>

272.5.19 BLAS_SYMM Procedures

This procedure performs one of the matrix-matrix operations
C := alpha*A*B + beta*C or C := alpha*B*A + beta*C, where alpha and beta are scalars, A is a symmetric matrix, and B and C are m by n matrices.

See Also:
BLAS Level 3 (Matrix-Matrix Operations) Subprograms for other subprograms in this group

Syntax

UTL_NLA.BLAS_SYMM (side IN     flag,
  uplo IN     flag,
  m IN     POSITIVEN,
  n IN     POSITIVEN,
  alpha IN     SCALAR_DOUBLE,
  a IN     UTL_NLA_ARRAY_DBL,
  lda IN     POSITIVEN,
  b IN     UTL_NLA_ARRAY_DBL,
  ldb IN     POSITIVEN,
beta  IN  SCALAR_DOUBLE,
c  IN OUT  UTL_NLA_ARRAY_DBL,
ldc  IN  POSITIVEN,
pack IN  flag DEFAULT 'C');

UTL_NLA.BLAS_SYMM ( side IN  flag,
uplo IN  flag,
m  IN  POSITIVEN,
n  IN  POSITIVEN,
alpha IN  SCALAR_FLOAT,
a  IN  UTL_NLA_ARRAYFLT,
lda IN  POSITIVEN,
b  IN  UTL_NLA_ARRAYFLT,
ldb IN  POSITIVEN,
beta IN  SCALAR_FLOAT,
c  IN OUT  UTL_NLA_ARRAYFLT,
ldc IN  POSITIVEN,
pack IN  flag DEFAULT 'C');

Parameters

Table 272-25  BLAS_SYMM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>side</td>
<td>Specifies whether the symmetric matrix A appears on the left or right in the operation:</td>
</tr>
<tr>
<td></td>
<td>side = 'L' or 'l': C := alpha<em>A</em>B + beta*C</td>
</tr>
<tr>
<td></td>
<td>side = 'R' or 'r': C := alpha<em>B</em>A + beta*C</td>
</tr>
<tr>
<td>uplo</td>
<td>Specifies whether the upper or lower triangular part of the array A is to be referenced:</td>
</tr>
<tr>
<td></td>
<td>uplo = 'U' or 'u': Only the upper triangular part of the symmetric matrix is to be referenced.</td>
</tr>
<tr>
<td></td>
<td>uplo = 'L' or 'l': Only the lower triangular part of the symmetric matrix is to be referenced.</td>
</tr>
<tr>
<td>m</td>
<td>Specifies the number of rows of the matrix C. m must be at least zero.</td>
</tr>
<tr>
<td>n</td>
<td>Specifies the number of columns of the matrix C. n must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.</td>
</tr>
</tbody>
</table>
**Table 272-25  (Cont.) BLAS_SYMM Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION <code>(lda, ka)</code> where <code>ka</code> is <code>m</code> when <code>side = 'L'</code> or <code>'l'</code>, and is <code>n</code> otherwise. Before entry with <code>side = 'L'</code> or <code>'l'</code>, the leading <code>m</code> by <code>m</code> part of the array <code>A</code> must contain the symmetric matrix, such that when <code>uplo = 'U'</code> or <code>'u'</code>, the leading <code>m</code> by <code>m</code> upper triangular part of the array <code>A</code> must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of <code>A</code> is not referenced, and when <code>uplo = 'L'</code> or <code>'l'</code>, the leading <code>m</code> by <code>m</code> lower triangular part of the array <code>A</code> must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of <code>A</code> is not referenced. Before entry with <code>side = 'R'</code> or <code>'r'</code>, the <code>n</code> by <code>n</code> part of the array <code>A</code> must contain the symmetric matrix, such that when <code>uplo = 'U'</code> or <code>'u'</code>, the leading <code>n</code> by <code>n</code> upper triangular part of the array <code>A</code> must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of <code>A</code> is not referenced, and when <code>uplo = 'L'</code> or <code>'l'</code>, the leading <code>n</code> by <code>n</code> lower triangular part of the array <code>A</code> must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of <code>A</code> is not referenced.</td>
</tr>
<tr>
<td><code>lda</code></td>
<td>Specifies the first dimension of <code>a</code> as declared in the calling (sub) program. When <code>side = 'L'</code> or <code>'l'</code>, <code>lda</code> must be at least <code>max(1, m)</code> otherwise <code>lda</code> must be at least <code>max(1, n)</code>.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION <code>(ldb, n)</code>. Before entry, the leading <code>m</code> by <code>n</code> part of the array <code>B</code> must contain the matrix <code>B</code>.</td>
</tr>
<tr>
<td><code>ldb</code></td>
<td>Specifies the first dimension of <code>b</code> as declared in the calling (sub) program. <code>ldb</code> must be at least <code>max(1, m)</code>.</td>
</tr>
<tr>
<td><code>beta</code></td>
<td>SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When <code>beta</code> is supplied as zero then <code>c</code> need not be set on input.</td>
</tr>
<tr>
<td><code>c</code></td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION <code>(ldc, n)</code>. Before entry, the leading <code>m</code> by <code>n</code> part of the array <code>C</code> must contain the matrix <code>C</code>, except when <code>beta</code> is zero, in which case <code>C</code> need not be set on entry. On exit, the array <code>C</code> is overwritten by the <code>m</code> by <code>n</code> updated matrix.</td>
</tr>
<tr>
<td><code>ldc</code></td>
<td>Specifies the first dimension of <code>C</code> as declared in the calling (sub) program. <code>ldc</code> must be at least <code>max(1, m)</code>.</td>
</tr>
<tr>
<td><code>pack</code></td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>• <code>'C'</code>: column-major (default)</td>
</tr>
<tr>
<td></td>
<td>• <code>'R'</code>: row-major</td>
</tr>
</tbody>
</table>
272.5.20 BLAS_SYMV Procedures

This procedure performs the matrix-vector operation \( y := \alpha A x + \beta y \), where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are \( n \) element vectors and \( A \) is an \( n \) by \( n \) symmetric matrix.

**See Also:**

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

**Syntax**

```plaintext
UTL_NLA.BLAS_SYMV (
    uplo   IN      flag,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_DOUBLE/DOUBLE,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    x      IN      UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    beta   IN      SCALAR_DOUBLE,
    y      IN OUT  UTL_NLA_ARRAY_DBL,
    incy   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_SYMV (
    uplo   IN      flag,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_FLOAT,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    x      IN      UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    beta   IN      SCALAR_FLOAT,
    y      IN OUT  UTL_NLA_ARRAY_FLT,
    incy   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

**Parameters**

Table 272-26  BLAS_SYMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies whether the upper or lower triangular part of the array \( A \) is to be referenced:  
  * uplo = 'U' or 'u'. Only the upper triangular part of \( A \) is to be referenced.  
  * uplo = 'L' or 'l'. Only the lower triangular part of \( A \) is to be referenced.  
| n         | Specifies the order of the matrix \( A \). \( n \) must be at least zero.  
| alpha     | SCALAR_FLOAT/DOUBLE. Specifies the scalar \( \alpha \).  |
Table 272-26  (Cont.) BLAS_SYMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.</td>
</tr>
<tr>
<td>lda</td>
<td>Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1,n).</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx)) Before entry, the incremented array X must contain the n element vector x.</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. Must not be zero.</td>
</tr>
<tr>
<td>beta</td>
<td>SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.</td>
</tr>
<tr>
<td>y</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incy)) Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.</td>
</tr>
<tr>
<td>incy</td>
<td>Specifies the increment for the elements of y. Must not be zero.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices: • 'C': column-major (default) • 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.21 BLAS_SYR Procedures

This procedure performs the rank 1 operation \( A := \alpha x^T x + A \), where \( \alpha \) is a real scalar, \( x \) is an \( n \) element vector, and \( A \) is an \( n \) by \( n \) symmetric matrix.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```plaintext
UTL_NLA.BLAS_SYR (  
  uplo IN flag,  
  n IN POSITIVEN,  
  alpha IN SCALAR_DBL,  
```

Chapter 272
Summary of UTL_NLA Subprograms
UTL_NLA.BLAS_SYR (
  uplo IN flag,
  n IN POSITIVEN,
  alpha IN SCALAR_FLT,
  x IN OUT UTL_NLA_ARRAY_FLT,
  incx IN POSITIVEN,
  a IN OUT UTL_NLA_ARRAY_FLT,
  lda IN POSITIVEN,
  pack IN flag DEFAULT 'C');

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>Specifies whether the upper or lower triangular part of the array $A$ is to be referenced:</td>
</tr>
<tr>
<td></td>
<td>- uplo = 'U' or 'u': Only the upper triangular part of $A$ is to be referenced.</td>
</tr>
<tr>
<td></td>
<td>- uplo = 'L' or 'l': Only the lower triangular part of $A$ is to be referenced.</td>
</tr>
<tr>
<td>n</td>
<td>Specifies the order of the matrix $A$. $n$ must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>Specifies the scalar $\alpha$.</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1+(n-1)\times\text{abs}(\text{incx}))$</td>
</tr>
<tr>
<td></td>
<td>Before entry, the incremented array $X$ must contain the $m$ element vector $x$.</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of $x$. $\text{incx}$ must not be zero.</td>
</tr>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION $(\text{lda}, n)$</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'U' or 'u', the leading $n$ by $n$ upper triangular part of the array $A$ must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of $A$ is not referenced. On exit, the upper triangular part of the array $A$ is overwritten by the upper triangular part of the updated matrix.</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'L' or 'l', the leading $n$ by $n$ lower triangular part of the array $A$ must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of $A$ is not referenced. On exit, the lower triangular part of the array $A$ is overwritten by the lower triangular part of the updated matrix.</td>
</tr>
<tr>
<td>lda</td>
<td>Specifies the first dimension of $a$ as declared in the calling (sub) program. $\text{lda}$ must be at least</td>
</tr>
<tr>
<td></td>
<td>$\max(1, n)$</td>
</tr>
</tbody>
</table>
Table 272-27  (Cont.) BLAS_SYR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>· 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>· 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.22 BLAS_SYR2 Procedures

This procedure performs the rank 2 operation $A := \alpha x'y' + \alpha y'x' + A$, where $\alpha$ is a scalar, $x$ and $y$ are $n$ element vectors, and $A$ is an $n$ by $n$ symmetric matrix.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

UTL_NLA.BLAS_SYR2 (  
  uplo IN flag,  
  n IN POSITIVEN,  
  alpha IN SCALAR_DBL,  
  x IN UTL_NLA_ARRAY_DBL,  
  incx IN POSITIVEN,  
  y IN UTL_NLA_ARRAY_DBL,  
  incy IN POSITIVEN,  
  a IN OUT UTL_NLA_ARRAY_DBL,  
  lda IN POSITIVEN,  
  pack IN flag DEFAULT 'C');

UTL_NLA.BLAS_SYR2 (  
  uplo IN flag,  
  n IN POSITIVEN,  
  alpha IN SCALAR_FLT,  
  x IN UTL_NLA_ARRAY_FLT,  
  incx IN POSITIVEN,  
  y IN UTL_NLA_ARRAY_FLT,  
  incy IN POSITIVEN,  
  a IN OUT UTL_NLA_ARRAY_FLT,  
  lda IN POSITIVEN,  
  pack IN flag DEFAULT 'C');
### Parameters

#### Table 272-28  BLAS_SYR2 Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **uplo**  | Specifies whether the upper or lower triangular part of the array A is to be referenced:  
|           | · uplo = 'U' or 'u': Only the upper triangular part of A is to be referenced.  
|           | · uplo = 'L' or 'l': Only the lower triangular part of A is to be referenced.  |
| **n**     | Specifies the order of the matrix A. n must be at least zero. |
| **alpha** | Specifies the scalar alpha. |
| **x**     | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
|           | ( 1 + ( n - 1 )*abs( incx ) )  
|           | Before entry, the incremented array X must contain the m element vector x. |
| **incx**  | Specifies the increment for the elements of x. incx must not be zero. |
| **y**     | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
|           | ( 1 + ( n - 1 )*abs( incy ) )  
|           | Before entry, the incremented array Y must contain the m element vector y. |
| **incy**  | Specifies the increment for the elements of y. incy must not be zero. |
| **a**     | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n)  
|           | With uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix.  
|           | With uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix. |
| **lda**   | Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least  
|           | max( 1, n ) |
| **pack**  | (Optional) Flags the packing of the matrices:  
|           | · 'C': column-major (default)  
|           | · 'R': row-major |

#### 272.5.23 BLAS_SYR2K Procedures

It performs one of the symmetric rank2 k operations  
$C := \alpha A^*B + \alpha B^*A' + \beta C$ or  
$C := \alpha A'^*B + \alpha B'^*A + \beta C$, where alpha and beta are scalars, C is
an \( n \) by \( n \) symmetric matrix and \( A \) and \( B \) are \( n \) by \( k \) matrices in the first case and \( k \) by \( n \) matrices in the second case.

### See Also:

BLAS Level 3 (Matrix-Matrix Operations) Subprograms for other subprograms in this group

#### Syntax

```fortran
UTL_NLA.BLAS_SYR2K (
    uplo IN flag,
    trans IN flag,
    n IN POSITIVEN,
    k IN POSITIVEN,
    alpha IN SCALAR_DOUBLE,
    a IN UTL_NLA_ARRAY_DBL,
    lda IN POSITIVEN,
    b IN UTL_NLA_ARRAY_DBL,
    ldb IN POSITIVEN,
    beta IN SCALAR_DOUBLE,
    c IN OUT UTL_NLA_ARRAY_DBL,
    ldc IN POSITIVEN,
    pack IN flag DEFAULT 'C');

UTL_NLA.BLAS_SYR2K (
    uplo IN flag,
    trans IN flag,
    n IN POSITIVEN,
    k IN POSITIVEN,
    alpha IN SCALAR_FLOAT,
    a IN UTL_NLA_ARRAYFLT,
    lda IN POSITIVEN,
    b IN OUT UTL_NLA_ARRAYFLT,
    ldb IN POSITIVEN,
    beta IN SCALAR_FLOAT,
    c IN OUT UTL_NLA_ARRAYFLT,
    ldc IN POSITIVEN,
    pack IN flag DEFAULT 'C');
```

#### Parameters

**Table 272-29**  
BLAS_SYR2K Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `uplo`    | Specifies whether the upper or lower triangular part of the array \( C \) is to be referenced:
|           | - \( uplo = 'U' \) or \('u' \): Only the upper triangular part of \( C \) is to be referenced.
|           | - \( uplo = 'L' \) or \('l' \): Only the lower triangular part of \( C \) is to be referenced. |
Table 272-29  (Cont.) BLAS_SYR2K Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **trans** | Specifies the operations to be performed:  
  - trans = 'N' or 'n': \( C := alpha*A*B' + alpha*B*A' + beta*C \)  
  - trans = 'T' or 't': \( C := alpha*A'*B + alpha*B'*A + beta*C \)  
  - trans = 'C' or 'c': \( C := alpha*A'*B + alpha*B'*A + beta*C \)  
| **n** | Specifies the order of matrix C. n must be at least zero.  
| **k** | On entry with trans = 'N' or 'n', k specifies the number of columns of the matrices A and B. On entry with trans = 'T' or 't' or trans = 'C' or 'c', k specifies the number of rows of the matrices A and B. k must be at least zero. |
| **alpha** | SCALAR_FLOAT/DouBLER. Specifies the scalar alpha. |
| **a** | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, ka) where kb is k when trans = 'N' or 'n', and is n otherwise. Before entry with trans = 'N' or 'n', the leading \( n \) by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A. |
| **lda** | Specifies the first dimension of a as declared in the calling (sub) program. When trans = 'N' or 'n', lda must be at least max(1, n), otherwise lda must be at least max(1, k). |
| **b** | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, kb) where kb is k when trans = 'N' or 'n', and is n otherwise. Before entry with trans = 'N' or 'n', the leading \( n \) by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B. |
| **ldb** | Specifies the first dimension of b as declared in the calling (sub) program. When trans = 'N' or 'n', ldb must be at least max(1, n), otherwise ldb must be at least max(1, k). |
| **beta** | SCALAR_FLOAT/DouBLER. Specifies the scalar beta. |
| **c** | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc, n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix. |
| **ldc** | Specifies the first dimension of c as declared in the calling (sub) program. ldc must be at least max(1, n). |
Table 272-29  (Cont.) BLAS_SYR2K Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>• 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>• 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.24 BLAS_SYRK Procedures

This procedure performs one of the symmetric rank k operations \( C := \alpha A^A A' + \beta C \) or \( C := \alpha A'A + \beta C \), where \( \alpha \) and \( \beta \) are scalars, \( C \) is an \( n \times n \) symmetric matrix and \( A \) is an \( n \times k \) matrix in the first case and a \( k \times n \) matrix in the second case.

See Also:
BLAS Level 3 (Matrix-Matrix Operations) Subprograms for other subprograms in this group

Syntax

```plaintext
UTL_NLA.BLAS_SYRK ( 
  uplo IN   flag, 
  trans IN  flag, 
  n IN      POSITIVEN, 
  k IN      POSITIVEN, 
  alpha IN  SCALAR_DOUBLE, 
  a IN      UTL_NLA_ARRAY_DBL, 
  lda IN    POSITIVEN, 
  beta IN   SCALAR_DOUBLE, 
  c IN OUT  UTL_NLA_ARRAY_DBL, 
  ldc IN    POSITIVEN, 
  pack IN   flag DEFAULT 'C');

UTL_NLA.BLAS_SYRK ( 
  uplo IN   flag, 
  trans IN  flag, 
  n IN      POSITIVEN, 
  k IN      POSITIVEN, 
  alpha IN  SCALAR_FLOAT, 
  a IN      UTL_NLA_ARRAY_FLT, 
  lda IN    POSITIVEN, 
  beta IN   SCALAR_FLOAT, 
  c IN OUT  UTL_NLA_ARRAY_DBL, 
  ldc IN    POSITIVEN, 
  pack IN   flag DEFAULT 'C');
```
### Parameters

#### Table 272-30  BLAS_SYRK Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies whether the upper or lower triangular part of the array C is to be referenced:  
  - uplo = 'U' or 'u': Only the upper triangular part of C is to be referenced.  
  - uplo = 'L' or 'l': Only the lower triangular part of C is to be referenced.  |
| trans     | Specifies the operations to be performed:  
  - trans = 'N' or 'n': $C := \alpha A^T A + \beta C$  
  - trans = 'T' or 't': $C := \alpha A A^T + \beta C$  
  - trans = 'C' or 'c': $C := \alpha A^T A + \beta C$  |
| n         | Specifies the order of matrix C. n must be at least zero.  |
| k         | On entry with trans = 'N' or 'n', k specifies the number of columns of the matrix A. On entry with trans = 'T' or 't' or trans = 'C' or 'c', k specifies the number of rows of the matrix A. k must be at least zero.  |
| alpha     | SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.  |
| a         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,ka) where ka is k when trans = 'N' or 'n', and is n otherwise.  
  Before entry with trans = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.  |
| lda       | Specifies the first dimension of a as declared in the calling (sub) program. When trans = 'N' or 'n', lda must be at least max(1,n), otherwise lda must be at least max(1,k).  |
| beta      | SCALAR_FLOAT/DOUBLE. Specifies the scalar beta.  |
| c         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc,n).  
  Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix.  
  Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.  |
| ldc       | Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least max(1,n).  |
| pack      | (Optional) Flags the packing of the matrices:  
  - 'C': column-major (default)  
  - 'R': row-major  |
272.5.25 BLAS_TBMV Procedures

This procedure performs the matrix-vector operations \( x := A \cdot x \) or \( x := A^* \cdot x \), where \( x \) is an \( n \) element vector and \( A \) is an \( n \times n \) unit, or non-unit, upper or lower triangular band matrix, with \((k+1)\) diagonals.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

\[
\begin{align*}
\text{UTL_NLA.BLAS_TBMV (}
& \quad \text{uplo IN flag,} \\
& \quad \text{trans IN flag,} \\
& \quad \text{diag IN flag,} \\
& \quad \text{n IN POSITIVEN,} \\
& \quad \text{k IN NATURALN,} \\
& \quad \text{a IN UTL_NLA_ARRAY_DBL,} \\
& \quad \text{lda IN POSITIVEN,} \\
& \quad \text{x IN OUT UTL_NLA_ARRAY_DBL,} \\
& \quad \text{incx IN POSITIVEN,} \\
& \quad \text{pack IN flag DEFAULT 'C');}
\end{align*}
\]

\[
\begin{align*}
\text{UTL_NLA.BLAS_TBMV (}
& \quad \text{uplo IN flag,} \\
& \quad \text{trans IN flag,} \\
& \quad \text{diag IN flag,} \\
& \quad \text{n IN POSITIVEN,} \\
& \quad \text{k IN NATURALN,} \\
& \quad \text{a IN UTL_NLA_ARRAY_FLT,} \\
& \quad \text{lda IN POSITIVEN,} \\
& \quad \text{x IN OUT UTL_NLA_ARRAY_FLT,} \\
& \quad \text{incx IN POSITIVEN,} \\
& \quad \text{pack IN flag DEFAULT 'C');}
\end{align*}
\]

Parameters

Table 272-31  BLAS_TBMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies whether the matrix is an upper or lower triangular matrix:  
  - \( \text{uplo} = 'U' \) or \( 'u' \). \( A \) is an upper triangular matrix.  
  - \( \text{uplo} = 'L' \) or \( 'l' \). \( A \) is a lower triangular matrix. |
| trans     | Specifies the operation to be performed:  
  - \( \text{trans} = 'N' \) or \( 'n' \). \( x := A \cdot x \)  
  - \( \text{trans} = 'T' \) or \( 't' \). \( x := A^* \cdot x \)  
  - \( \text{trans} = 'C' \) or \( 'c' \). \( x := A^* \cdot x \) |
Table 272-31  (Cont.) BLAS_TBMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| diag      | Specifies whether or not A is unit triangular:  
  • diag = 'U' or 'u'. A is assumed to be unit triangular.  
  • diag = 'N' or 'n'. A is not assumed to be unit triangular. |
| n         | Specifies the order of the matrix A. n must be at least zero. |
| k         | Specifies whether or not A is unit triangular:  
  • with uplo = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.  
  • with uplo = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A.  
  K must satisfy 0 <= k. |
| a         | UTL_NLA_ARRAYFLT/DBL of DIMENSION (lda,n).  
  Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced.  
  Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced.  
  Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. |
| lda       | Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (k+1). |
| x         | UTL_NLA_ARRAYFLT/DBL of dimension at least (1+ (n-1)*abs(incx)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x. |
| incx      | Specifies the increment for the elements of x. Must not be zero. |
| pack      | (Optional) Flags the packing of the matrices:  
  • 'C': column-major (default)  
  • 'R': row-major |
272.5.26 BLAS_TBSV Procedures

This procedure solves one of the systems of equations $A^x = b$ or $A'^x = b$, where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular band matrix, with $(k+1)$ diagonals.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```plaintext
UTL_NLA.BLAS_TBSV (uplo IN flag,
     trans IN flag,
     diag IN flag,
     n IN POSITIVEN,
     k IN NATURALN,
     a IN UTL_NLA_ARRAY_DBL,
     lda IN POSITIVEN,
     x IN OUT UTL_NLA_ARRAY_DBL,
     incx IN POSITIVEN,
     pack IN flag DEFAULT 'C');

UTL_NLA.BLAS_STBSV (uplo IN flag,
     trans IN flag,
     diag IN flag,
     n IN POSITIVEN,
     k IN NATURALN,
     a IN UTL_NLA_ARRAY_FLT,
     lda IN POSITIVEN,
     x IN OUT UTL_NLA_ARRAY_FLT,
     incx IN POSITIVEN,
     pack IN flag DEFAULT 'C');
```

Parameters

Table 272-32  BLAS_TBSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>Specifies whether the matrix is an upper or lower triangular matrix:</td>
</tr>
<tr>
<td></td>
<td>· uplo = 'U' or 'u': $A$ is an upper triangular matrix.</td>
</tr>
<tr>
<td></td>
<td>· uplo = 'L' or 'l': $A$ is a lower triangular matrix.</td>
</tr>
<tr>
<td>trans</td>
<td>Specifies the equations to be solved:</td>
</tr>
<tr>
<td></td>
<td>· trans = 'N' or 'n': $A^x = b$</td>
</tr>
<tr>
<td></td>
<td>· trans = 'T' or 't': $A'^x = b$</td>
</tr>
<tr>
<td></td>
<td>· trans = 'C' or 'c': $A'^x = b$</td>
</tr>
</tbody>
</table>
Table 272-32  (Cont.) BLAS_TBSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| diag      | Specifies whether or not A is unit triangular:  
  - diag = 'U' or 'u': A is assumed to be unit triangular.  
  - diag = 'N' or 'n': A is not assumed to be unit triangular. |
| n         | Specifies the order of the matrix A. n must be at least zero. |
| k         | Specifies whether or not A is unit triangular:  
  - with uplo = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.  
  - with uplo = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A.  
  K must satisfy 0 <= k. |
| a         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n).  
  Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced.  
  Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced.  
  Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. |
| lda       | On entry, lda specifies the first dimension of A as declared in the calling (sub) program. lda must be at least (k+1). |
| x         | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
  ( 1 + ( n - 1 ) *abs( incx ) )  
  Before entry, the incremented array X must contain the n element right-hand side vector b.  
  On exit, X is overwritten with the solution vector x. |
| incx      | Specifies the increment for the elements of x. incx must not be zero. |
| pack      | (Optional) Flags the packing of the matrices:  
  - 'C': column-major (default)  
  - 'R': row-major |

Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.
272.5.27 BLAS_TPMV Procedures

This procedure performs the matrix-vector operations $x := A \cdot x$ or $x := A^* \cdot x$, where $x$ is an $n$ element vector and $A$ is an $n \times n$ unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

See Also:

BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TPMV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    ap     IN      UTL_NLA_ARRAY_DBL,
    x      IN OUT  UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TBMV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    ap     IN      UTL_NLA_ARRAY_FLT,
    x      IN OUT  UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

Parameters

Table 272-33  BLAS_TPMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>Specifies whether the matrix is an upper or lower triangular matrix:</td>
</tr>
<tr>
<td></td>
<td>- uplo = 'U' or 'u'. $A$ is an upper triangular matrix.</td>
</tr>
<tr>
<td></td>
<td>- uplo = 'L' or 'l'. $A$ is a lower triangular matrix.</td>
</tr>
<tr>
<td>trans</td>
<td>Specifies the operation to be performed:</td>
</tr>
<tr>
<td></td>
<td>- trans = 'N' or 'n'. $x := A \cdot x$</td>
</tr>
<tr>
<td></td>
<td>- trans = 'T' or 't'. $x := A^* \cdot x$</td>
</tr>
<tr>
<td></td>
<td>- trans = 'C' or 'c'. $x := A^* \cdot x$</td>
</tr>
<tr>
<td>diag</td>
<td>Specifies whether or not $A$ is unit triangular:</td>
</tr>
<tr>
<td></td>
<td>- diag = 'U' or 'u'. $A$ is assumed to be unit triangular.</td>
</tr>
<tr>
<td></td>
<td>- diag = 'N' or 'n'. $A$ is not assumed to be unit triangular.</td>
</tr>
<tr>
<td>n</td>
<td>Specifies the order of the matrix $A$. $n$ must be at least zero.</td>
</tr>
</tbody>
</table>
Table 272-33  (Cont.) BLAS_TPMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ap</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda,n). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.</td>
</tr>
<tr>
<td><strong>x</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+ (n-1)*abs(incx)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.</td>
</tr>
<tr>
<td><strong>incx</strong></td>
<td>Specifies the increment for the elements of x. Must not be zero.</td>
</tr>
<tr>
<td><strong>pack</strong></td>
<td>(Optional) Flags the packing of the matrices: 'C': column-major (default) 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.28 BLAS_TPSV Procedures

This procedure solves one of the systems of equations A*x = b or A'*x = b, where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

See Also:
BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TPSV ( uplo IN flag, trans IN flag, diag IN flag, n IN POSITIVEN, ap IN UTL_NLA_ARRAY_DBL, x IN OUT UTL_NLA_ARRAY_DBL, incx IN POSITIVEN, pack IN flag DEFAULT 'C');
```
UTL_NLA.BLAS_TPSV (  
  uplo IN flag,  
  trans IN flag,  
  diag IN flag,  
  n IN POSITIVEN,  
  ap IN UTL_NLA_ARRAY_FLT,  
  x IN OUT UTL_NLA_ARRAY_FLT,  
  incx IN POSITIVEN,  
  pack IN flag DEFAULT 'C');

Parameters

Table 272-34 BLAS_TPSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies whether the matrix is an upper or lower triangular matrix:  
  - uplo = 'U' or 'u': A is an upper triangular matrix.  
  - uplo = 'L' or 'l': A is a lower triangular matrix. |
| trans     | Specifies the operation to be performed:  
  - trans = 'N' or 'n': A*x = b  
  - trans = 'T' or 't': A'*x = b  
  - trans = 'C' or 'c': A'*x = b |
| diag      | Specifies whether or not A is unit triangular:  
  - diag = 'U' or 'u': A is assumed to be unit triangular.  
  - diag = 'N' or 'n': A is not assumed to be unit triangular. |
| n         | Specifies the order of the matrix A. n must be at least zero. |
| ap        | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
  \((n*(n+1))/2\)  
  Before entry with uplo = 'U' or 'u', the array ap must contain the upper triangular matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(1,2) and a(2,2) respectively, and so on.  
  Before entry with uplo = 'L' or 'l', the array ap must contain the lower triangular matrix packed sequentially, column by column, so that ap(1) contains a(1,1), ap(2) and ap(3) contain a(2,1) and a(3,1) respectively, and so on.  
  Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity. |
| x         | UTL_NLA_ARRAY_FLT/DBL of dimension at least  
  \((1 + (n - 1) *abs(incx))\)  
  Before entry, the incremented array x must contain the n element right-hand side vector b. On exit, x is overwritten with the solution vector x. |
| incx      | Specifies the increment for the elements of x. incx must not be zero. |
| pack      | (Optional) Flags the packing of the matrices:  
  - 'C': column-major (default)  
  - 'R': row-major |
Usage Notes
No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

272.5.29 BLAS_TRMM Procedures
This procedure performs a matrix-matrix operation.

It performs one of the following matrix-matrix operations:

\[ \begin{align*}
B & := \alpha \cdot \text{op}(A) \cdot B \\
\text{or} \\
B & := \alpha \cdot B \cdot \text{op}(A)
\end{align*} \]

where \( \alpha \) is a scalar, \( B \) is an \( m \times n \) matrix, \( A \) is a unit, or non-unit, upper or lower triangular matrix and \( \text{op}(A) \) is one of

\[ \text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A' \]

See Also:
BLAS Level 3 (Matrix-Matrix Operations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TRMM ( 
   side   IN      flag, 
   uplo   IN      flag, 
   transa IN      flag, 
   diag   IN      flag, 
   m      IN      POSITIVEN, 
   n      IN      POSITIVEN, 
   alpha  IN      SCALAR_DOUBLE, 
   a      IN      UTL_NLA_ARRAY_DBL, 
   lda    IN      POSITIVEN, 
   b      IN OUT UTL_NLA_ARRAY_DBL, 
   ldb    IN      POSITIVEN, 
   pack   IN      flag DEFAULT 'C');

UTL_NLA.BLAS_TRMM ( 
   side   IN      flag, 
   uplo   IN      flag, 
   transa IN      flag, 
   diag   IN      flag, 
   m      IN      POSITIVEN, 
   n      IN      POSITIVEN, 
   alpha  IN      SCALAR_FLOAT, 
   a      IN      UTL_NLA_ARRAY_FLT, 
```
Parameters

Table 272-35  BLAS_TRMM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| side      | Specifies whether the symmetric matrix A appears on the left or right in the operation:  
|           | - side = 'L' or 'l': B := alpha*op(A)*B  
|           | - side = 'R' or 'r': B := alpha*B*op(A)  
| uplo      | Specifies whether the upper or lower triangular part of the array A is to be referenced:  
|           | - uplo = 'U' or 'u': A is an upper triangular matrix.  
|           | - uplo = 'L' or 'l': A is a lower triangular matrix.  
| transa    | Specifies the form of op(A) to be used in the matrix multiplication as follows:  
|           | - transa = 'N' or 'n': op(A) = A  
|           | - transa = 'T' or 't': op(A) = A'  
|           | - transa = 'C' or 'c': op(A) = A'  
| diag      | Specifies whether or not A is unit triangular:  
|           | - diag = 'U' or 'u': A is assumed to be unit triangular.  
|           | - diag = 'N' or 'n': A is not assumed to be unit triangular.  
| m         | Specifies the number of rows of the B. m must be at least zero.  
| n         | Specifies the number of columns of B. n must be at least zero.  
| alpha     | SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.  
| a         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, k) where k is m when side = 'L' or 'l', and is n when side = 'R' or 'r'.  
|           | Before entry with uplo = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix, and the strictly lower triangular part of A is not referenced.  
|           | Before entry with uplo = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced.  
|           | Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.  
| lda       | Specifies the first dimension of a as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least max(1, m), otherwise lda must be at least max(1, n).  
| b         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb, n).  
|           | Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.  
| ldb       | Specifies the first dimension of b as declared in the calling (sub) program. ldb must be at least max(1, m).
Table 272-35  (Cont.) BLAS_TRMM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| pack      | (Optional) Flags the packing of the matrices:  
|           | · 'C': column-major (default)  
|           | · 'R': row-major |

272.5.30 BLAS_TRMV Procedures

This procedure performs the matrix-vector operations \( \mathbf{x} := \mathbf{A}\mathbf{x} \) or \( \mathbf{x} := \mathbf{A}'\mathbf{x} \), where \( \mathbf{x} \) is an \( n \) element vector and \( \mathbf{A} \) is an \( n \) by \( n \) unit, or non-unit, upper or lower triangular matrix.

See Also:

[BLAS Level 2 (Matrix-Vector Operations) Subprograms](#) for other subprograms in this group

Syntax

```
UTL_NLA.BLAS_TRMV (  
  uplo   IN      flag,  
  trans  IN      flag,  
  diag   IN      flag,  
  n      IN      POSITIVEN,  
  a      IN      UTL_NLA_ARRAY_DBL,  
  lda    IN      POSITIVEN,  
  x      IN OUT  UTL_NLA_ARRAY_DBL,  
  incx   IN      POSITIVEN,  
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TRMV (  
  uplo   IN      flag,  
  trans  IN      flag,  
  diag   IN      flag,  
  n      IN      POSITIVEN,  
  a      IN      UTL_NLA_ARRAY_FLT,  
  lda    IN      POSITIVEN,  
  x      IN OUT  UTL_NLA_ARRAY_FLT,  
  incx   IN      POSITIVEN,  
  pack   IN      flag DEFAULT 'C');
```

Parameters

Table 272-36  BLAS_TRMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | Specifies whether the matrix is an upper or lower triangular matrix:  
|           | · uplo = 'U' or 'u'. \( \mathbf{A} \) is an upper triangular matrix.  
|           | · uplo = 'L' or 'l'. \( \mathbf{A} \) is a lower triangular matrix. |
Table 272-36  (Cont.) BLAS_TRMV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trans</td>
<td>Specifies the operation to be performed:</td>
</tr>
<tr>
<td></td>
<td>• trans = 'N' or 'n': := A*x</td>
</tr>
<tr>
<td></td>
<td>• trans= 'T' or 't' : := A'*x</td>
</tr>
<tr>
<td></td>
<td>• trans = 'C' or 'c': := A'*x</td>
</tr>
<tr>
<td>diag</td>
<td>Specifies whether or not A is unit triangular:</td>
</tr>
<tr>
<td></td>
<td>• diag = 'U' or 'u'. A is assumed to be unit triangular.</td>
</tr>
<tr>
<td></td>
<td>• diag = 'N' or 'n'. A is not assumed to be unit triangular.</td>
</tr>
<tr>
<td>n</td>
<td>Specifies the order of the matrix A. n must be at least zero.</td>
</tr>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n).</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'U' or 'u', the leading n by n upper</td>
</tr>
<tr>
<td></td>
<td>triangular part of the array A must contain the upper triangular</td>
</tr>
<tr>
<td></td>
<td>matrix and the strictly lower triangular part of A is not</td>
</tr>
<tr>
<td></td>
<td>referenced.</td>
</tr>
<tr>
<td></td>
<td>Before entry with uplo = 'L' or 'l', the leading n by n lower</td>
</tr>
<tr>
<td></td>
<td>triangular part of the array A must contain the lower triangular</td>
</tr>
<tr>
<td></td>
<td>matrix and the strictly upper triangular part of A is not</td>
</tr>
<tr>
<td></td>
<td>referenced.</td>
</tr>
<tr>
<td></td>
<td>Note that when diag = 'U' or 'u', the diagonal elements of A are</td>
</tr>
<tr>
<td></td>
<td>not referenced either, but are assumed to be unity</td>
</tr>
<tr>
<td>lda</td>
<td>Specifies the first dimension of a as declared in the calling (sub)</td>
</tr>
<tr>
<td></td>
<td>program. lda must be at least max(1,n).</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+</td>
</tr>
<tr>
<td></td>
<td>(n-1)*as(incx)). Before entry, the incremented array X must</td>
</tr>
<tr>
<td></td>
<td>contain the n element vector x.</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. Must not be zero.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>• 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>• 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.31 BLAS_TRSM Procedures

This procedure performs a matrix-matrix operation.

It performs one of the matrix-matrix operations:

\[ \text{op(A)} \times X = \alpha B \]

or

\[ X \times \text{op(A)} = \alpha B \]

where \( \alpha \) is a scalar, \( X \) and \( B \) are \( m \) by \( n \) matrices, \( A \) is a unit, or non-unit, upper or lower triangular matrix and \( \text{op(A)} \) is one of

\[ \text{op(A)} = A \]

or

\[ \text{op(A)} = A^T \]
The matrix \( X \) is overwritten on \( B \).

### See Also:

BLAS Level 3 (Matrix-Matrix Operations) Subprograms for other subprograms in this group

### Syntax

```fortran
UTL_NLA.BLAS_TRSM (
    side   IN      flag,
    uplo   IN      flag,
    transa IN      flag,
    diag   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_DOUBLE,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    b      IN OUT  UTL_NLA_ARRAY_DBL,
    ldb    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```fortran
UTL_NLA.BLAS_TRSM (
    side   IN      flag,
    uplo   IN      flag,
    transa IN      flag,
    diag   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_FLOAT,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    b      IN OUT  UTL_NLA_ARRAY_FLT,
    ldb    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

### Parameters

#### Table 272-37  BLAS_TRSM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| side      | Specifies whether the symmetric matrix \( A \) appears on the left or right in the operation:  
  - \( \text{side} = 'L' \text{ or } 'l' \text{ : } \text{op}(A) \times X = \alpha B \)  
  - \( \text{side} = 'R' \text{ or } 'r' \text{ : } X \times \text{op}(A) = \alpha B \)  
| uplo      | Specifies whether the upper or lower triangular part of the array \( A \) is to be referenced:  
  - \( \text{uplo} = 'U' \text{ or } 'u' \text{ : } A \text{ is an upper triangular matrix.} \)  
  - \( \text{uplo} = 'L' \text{ or } 'l' \text{ : } A \text{ is a lower triangular matrix.} \) |
Table 272-37  (Cont.) BLAS_TRSM Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| transa    | Specifies the form of \( \text{op}(A) \) to be used in the matrix multiplication as follows:  
  - transa = 'N' or 'n': \( \text{op}(A) = A \)  
  - transa = 'T' or 't': \( \text{op}(A) = A' \)  
  - transa = 'C' or 'c': \( \text{op}(A) = A' \) |
| diag      | Specifies whether or not \( A \) is unit triangular:  
  - diag = 'U' or 'u'. \( A \) is assumed to be unit triangular.  
  - diag = 'N' or 'n'. \( A \) is not assumed to be unit triangular. |
| m         | Specifies the number of rows of the \( B \). \( m \) must be at least zero. |
| n         | Specifies the number of columns of \( B \). \( n \) must be at least zero. |
| alpha     | Specifies the scalar alpha. When alpha is zero then \( A \) is not referenced and \( B \) need not be set before entry. |
| a         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, k) where \( k \) is \( m \) when side = 'L' or 'l', and is \( n \) when side = 'R' or 'r'.  
  Before entry with uplo = 'U' or 'u', the leading \( k \) by \( k \) upper triangular part of the array \( A \) must contain the upper triangular matrix, and the strictly lower triangular part of \( A \) is not referenced.  
  Before entry with uplo = 'L' or 'l', the leading \( k \) by \( k \) lower triangular part of the array \( A \) must contain the lower triangular matrix and the strictly upper triangular part of \( A \) is not referenced.  
  Note that when diag = 'U' or 'u', the diagonal elements of \( A \) are not referenced either, but are assumed to be unity. |
| lda       | Specifies the first dimension of \( a \) as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least \( \max(1, m) \), otherwise lda must be at least \( \max(1, n) \). |
| b         | UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb, n).  
  Before entry, the leading \( m \) by \( n \) part of the array \( B \) must contain the matrix \( B \), and on exit is overwritten by the solution matrix \( X \). |
| ldb       | Specifies the first dimension of \( b \) as declared in the calling (sub) program. ldb must be at least \( \max(1, m) \). |
| pack      | (Optional) Flags the packing of the matrices:  
  - 'C': column-major (default)  
  - 'R': row-major |
272.5.32 BLAS_TRSV Procedures

This procedure solves one of the systems of equations $A \times x = b$ or $A^T \times x = b$, where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix.

See Also:
BLAS Level 2 (Matrix-Vector Operations) Subprograms for other subprograms in this group

Syntax

```c
UTL_NLA.BLAS_TRSV (  
    uplo   IN      flag,  
    trans  IN      flag,  
    diag   IN      flag,  
    n      IN      POSITIVEN,  
    a      IN      UTL_NLA_ARRAY_DBL,  
    lda    IN      POSITIVEN,  
    x      IN OUT  UTL_NLA_ARRAY_DBL,  
    incx   IN      POSITIVEN,  
    pack   IN      flag DEFAULT 'C');
```

```c
UTL_NLA.BLAS_TRSV (  
    uplo   IN      flag,  
    trans  IN      flag,  
    diag   IN      flag,  
    n      IN      POSITIVEN,  
    a      IN      UTL_NLA_ARRAY_FLT,  
    lda    IN      POSITIVEN,  
    x      IN OUT  UTL_NLA_ARRAY_FLT,  
    incx   IN      POSITIVEN,  
    pack   IN      flag DEFAULT 'C');
```

Parameters

Table 272-38  BLAS_TRSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>Specifies whether the matrix is an upper or lower triangular matrix:</td>
</tr>
<tr>
<td></td>
<td>• $uplo = 'U'$ or 'u'. $A$ is an upper triangular matrix.</td>
</tr>
<tr>
<td></td>
<td>• $uplo = 'L'$ or 'l'. $A$ is a lower triangular matrix.</td>
</tr>
<tr>
<td>trans</td>
<td>Specifies the operation to be performed:</td>
</tr>
<tr>
<td></td>
<td>• $trans = 'N'$ or 'n'. $A^T \times x = b$</td>
</tr>
<tr>
<td></td>
<td>• $trans = 'T'$ or 't'. $A^T \times x = b$</td>
</tr>
<tr>
<td></td>
<td>• $trans = 'C'$ or 'c'. $A^T \times x = b$</td>
</tr>
<tr>
<td>diag</td>
<td>Specifies whether or not $A$ is unit triangular:</td>
</tr>
<tr>
<td></td>
<td>• $diag = 'U'$ or 'u'. $A$ is assumed to be unit triangular.</td>
</tr>
<tr>
<td></td>
<td>• $diag = 'N'$ or 'n'. $A$ is not assumed to be unit triangular.</td>
</tr>
<tr>
<td>n</td>
<td>Specifies the order of the matrix $A$. $n$ must be at least zero.</td>
</tr>
</tbody>
</table>
Table 272-38  (Cont.) BLAS_TRSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.</td>
</tr>
<tr>
<td>lda</td>
<td>Specifies the first dimension of A as declared in the calling (sub) program. lda must be at least max(1, n).</td>
</tr>
<tr>
<td>x</td>
<td>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n - 1) * abs (incx)) Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.</td>
</tr>
<tr>
<td>incx</td>
<td>Specifies the increment for the elements of x. Must not be zero.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:  · 'C': column-major (default)  · 'R': row-major</td>
</tr>
</tbody>
</table>

Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

272.5.33 LAPACK_GBSV Procedures

This procedure computes the solution to a real system of linear equations a * x = b, where a is a band matrix of order n with kl sub diagonals and ku superdiagonals, and x and b are n by nrhs matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

\[ a = L \cdot U \]

where L is a product of permutation and unit lower triangular matrices with kl sub diagonals, and U is upper triangular with kl+ku superdiagonals. The factored form of a is then used to solve the system of equations

\[ a \cdot x = b \]

See Also:

LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group
### Syntax

`UTL_NLA.LAPACK_GBSV`

```plaintext
n      IN      POSITVEN,
kl     IN      NATURALN,
ku     IN      NATURALN,
nrhs   IN      POSITVEN,
ab     IN OUT  UTL_NLA_ARRAY_DBL,
ldab   IN      POSITVEN,
ipiv   IN OUT  UTL_NLA_ARRAY_INT,
b      IN OUT  UTL_NLA_ARRAY_DBL,
ldb    IN      POSITVEN,
info   OUT  INTEGER,
pack   IN      flag DEFAULT 'C');
```

```plaintext
UTL_NLA.LAPACK_GBSV
n      IN      POSITVEN,
kl     IN      NATURALN,
ku     IN      NATURALN,
nrhs   IN      POSITVEN,
ab     IN OUT  UTL_NLA_ARRAY_FLT,
ldab   IN      POSITVEN,
ipiv   IN OUT  UTL_NLA_ARRAY_INT,
b      IN OUT  UTL_NLA_ARRAY_FLT,
ldb    IN      POSITVEN,
info   OUT  INTEGER,
pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 272-39  LAPACK_GBSV Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The number of linear equations, equivalent to the order of the matrix $A$. $n \geq 0$</td>
</tr>
<tr>
<td>kl</td>
<td>The number of sub diagonals within the band of $A$. $kl \geq 0$.</td>
</tr>
<tr>
<td>ku</td>
<td>The number of superdiagonals within the band of $A$. $ku \geq 0$.</td>
</tr>
<tr>
<td>nrhs</td>
<td>The number of right-hand sides, which is the number of columns of the matrix $B$. $nrhs \geq 0$.</td>
</tr>
</tbody>
</table>
| ab        | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the matrix $A$ in band storage, in rows $kl+1$ to $2*kl+ku+1$; rows $1$ to $kl$ of the array need not be set. The $j$-th column of $A$ is stored in the $j$-th column of the array $ab$:

$$ab(kl+ku+1+i-j,j) = a(i,j) \text{ for max}(1,j-ku) \leq i \leq \min(n,j+kl)$$

On exit, details of the factorization: $U$ is stored as an upper triangular band matrix with $kl+ku$ superdiagonals in rows $1$ to $KL+KU+1$, and the multipliers used during the factorization are stored in rows $kl+ku+2$ to $2*kl+ku+1$. |
| ldab      | The leading dimension of the array $ab$. ldab $\geq 2*kl+ku+1$ |
Table 272-39  (Cont.) LAPACK_GBSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipiv</td>
<td>INTEGER array, DIMENSION (n). The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row ipiv(i).</td>
</tr>
<tr>
<td>b</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0 , the n by nrhs solution matrix X.</td>
</tr>
<tr>
<td>ldb</td>
<td>The leading dimension of the array b. ldb &gt;= max(1,n)</td>
</tr>
</tbody>
</table>
| info      | - = 0 : successful exit  
  - < 0 : if info = -i , the i-th argument had an illegal value  
  - > 0 : if info = i , U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and the solution has not been computed |
| pack      | (Optional) Flags the packing of the matrices:  
  - 'C': column-major (default)  
  - 'R': row-major |

272.5.34 LAPACK_GEES Procedures

This procedure computes for an n by n real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z.

This gives the Schur factorization A = Z*T*(Z**T).

A matrix is in real Schur form if it is upper quasi-triangular with 1 by 1 and 2 by 2 blocks. 2 by 2 blocks will be standardized in the form

\[
\begin{bmatrix}
  a & b \\
  c & a
\end{bmatrix}
\]

where b*c < 0. The eigenvalues of such a block are a ± sqrt(bc).

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GEES (    
  jobvs IN flag,    
  n IN POSITIVEN,    
  a IN OUT UTL_NLA_ARRAY_DBL,    
  lda IN POSITIVEN,    
  wr IN OUT UTL_NLA_ARRAY_DBL,    
) 
```


Parameters

Table 272-40   LAPACK_GEES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| jobz      | 'N': Schur vectors are not computed.  
           | 'V': Schur vectors are computed.     |
| n         | The order of the matrix \( A \). \( N \geq 0 \). |
| a         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n).  
           | On entry, the \( n \) by \( n \) matrix \( A \).  
           | On exit, \( A \) has been overwritten by its real Schur form \( T \). |
| lda       | The leading dimension of the array \( a \). \( lda \geq \max(1,n) \). |
| wr        | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).  
           | \( wr \) and \( wi \) contain the real and imaginary parts respectively of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form \( T \). Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first. |
| wi        | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).  
           | \( wr \) and \( wi \) contain the real and imaginary parts respectively of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form \( T \). Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first. |
| vs        | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).  
           | If \( jobvs = 'V' \), vs contains the orthogonal matrix \( Z \) of Schur vectors.  
           | If \( jobvs = 'N' \), vs is not referenced. |
| ldvs      | The leading dimension of the array \( vs \). \( ldvs \geq 1 \). If \( jobvs = 'V' \), \( ldvs \geq N \). |
Table 272-40  (Cont.) LAPACK_GEES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| info      | • = 0 : successful exit  
            • < 0: if info = -i, the i-th argument had an illegal value  
            • > 0: if info = i, and i is <= N: the QR algorithm failed to  
              compute all the eigenvalues. Elements 1:ilo-1 and i+1:n of wr  
              and wi contain those eigenvalues which have converged. If jobvs  
              = 'V', vs contains the matrix which reduces A to its partially  
              converged Schur form. |
| pack      | (Optional) Flags the packing of the matrices:  
            • 'C': column-major (default)  
            • 'R': row-major |

272.5.35 LAPACK_GELS Procedures

This procedure solves overdetermined or underdetermined real linear systems
involving an $m$ by $n$ matrix $A$, or its transpose, using a QR or LQ factorization of $A$. It is
assumed that $A$ has full rank.

The following options are provided:

1. If $\text{TRANS} = 'N'$ and $m \geq n$: find the least squares solution of an overdetermined
   system, that is, solve the least squares problem.
   
   minimize $|| B - A*X ||$

2. If $\text{TRANS} = 'N'$ and $m < n$: find the minimum norm solution of an underdetermined
   system $A \times X = B$.

3. If $\text{TRANS} = 'T'$ and $m \geq n$: find the minimum norm solution of an undetermined
   system $A^{**T} \times X = B$.

4. If $\text{TRANS} = 'T'$ and $m < n$: find the least squares solution of an overdetermined
   system, that is, solve the least squares problem minimize $|| B - A^{**T} \times X ||$.

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for
other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GELS (  
    trans   IN    flag,  
    m       IN    POSITIVEN,  
    n       IN    POSITIVEN,  
    nrhs    IN    POSITIVEN,  
    a       IN OUT UTL_NLA_ARRAY_DBL,  
    lda     IN    POSITIVEN,  
    b       IN OUT UTL_NLA_ARRAY_DBL,  
    ldb     IN    POSITIVEN,  
)  
```
info OUT INTEGER,
pack IN flag DEFAULT 'C');

UTL_NLA.LAPACK_GELS {
  trans IN flag,
  m IN POSITIVEN,
  n IN POSITIVEN,
  nrhs IN POSITIVEN,
  a IN OUT UTL_NLA_ARRAY_FLT,
  lda IN POSITIVEN,
  b IN OUT UTL_NLA_ARRAY_FLT,
  ldb IN POSITIVEN,
  info OUT INTEGER,
  pack IN flag DEFAULT 'C');

Parameters

Table 272-41  LAPACK_GELS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| trans     | • CHARACTER = 'N': The linear system involves A.  
            • CHARACTER = 'T': The linear system involves A**T. |
| m         | The number of rows of the matrix a. M >= 0. |
| n         | The number of columns of the matrix a. N >= 0. |
| nrhs      | The number of right-hand sides, which is the number of columns of the matrix b and x.nrhs >= 0. |
| a         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the matrix b of right hand side vectors, stored columnwise; b is m by nrhs if TRANS = 'N', or n by nrhs if TRANS = 'T'. On exit, if m >= n, a is overwritten by details of its QR factorization as returned by SGEQRF. If m < n, A is overwritten by details of its LQ factorization as returned by SGELQF. |
| lda       | The leading dimension of the array A. lda >= max(1,m). |
| b         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the matrix b of right hand side vectors, stored columnwise. b is m by nrhs if TRANS = 'N', or n by nrhs if TRANS = 'T'. On exit, b is overwritten by the solution vectors, stored columnwise:  
            • If TRANS = 'N' and m >= n, rows 1 to n of b contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements n+1 to m in that column.  
            • If TRANS = 'N' and m < n, rows 1 to n of b contain the minimum norm solution vectors.  
            • If TRANS = 'T' and m >= n, rows 1 to m of b contain the minimum norm solution vectors.  
            • If TRANS = 'T' and m < n, rows 1 to m of b contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements m+1 to n in that column. |
| ldb       | The leading dimension of the array b. ldb >= max(1,m,n) |
Table 272-41  (Cont.) LAPACK_GELS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| info      | ·  = 0 : successful exit  
|           | · < 0 : if info = -i, the i-th argument had an illegal value |
| pack      | (Optional) Flags the packing of the matrices:  
|           | · 'C': column-major (default)  
|           | · 'R': row-major |

272.5.36 LAPACK_GESDD Procedures

This procedures computes the singular value decomposition (SVD) of a real m by n matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.

The SVD is written

\[ A = U \times SIGMA \times transpose(V) \]

where SIGMA is an m by n matrix which is zero except for its min(m,n) diagonal elements, U is an m by m orthogonal matrix, and V is an n by n orthogonal matrix. The diagonal elements of SIGMA are the singular values of A, they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V**T, not V.

See Also:
LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GESDD (  
  jobz   IN       flag,  
  m      IN       POSITIVEN,  
  n      IN       POSITIVEN,  
  a      IN OUT   UTL_NLA_ARRAY_DBL,  
  lda    IN       POSITIVEN,  
  s      IN OUT   UTL_NLA_ARRAY_DBL,  
  u      IN OUT   UTL_NLA_ARRAY_DBL,  
  ldu    IN       POSITIVEN,  
  vt     IN OUT   UTL_NLA_ARRAY_DBL,  
  ldvt   IN       POSITIVEN,  
  info   OUT      INTEGER,  
  pack   IN       flag DEFAULT 'C');

UTL_NLA.LAPACK_GESDD (  
  jobz   IN       flag,  
  m      IN       POSITIVEN,  
```
Parameters

Table 272-42  LAPACK_GESDD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobz</td>
<td>Specifies options for computing all or part of the matrix $U$:</td>
</tr>
</tbody>
</table>
|           | $'A'$: All $m$ columns of $U$ and all $n$ rows of $V^*T$ are returned in arrays $U$ and $V^*$.
|           | $'S'$: The first $\min(m,n)$ columns of $U$ and the first $\min(m,n)$ rows of $V^*T$ are returned in the arrays $U$ and $V^*$.
|           | $'O'$: The first $\min(m,n)$ columns of $U$ (the left singular vectors) are overwritten on the array $A$. $\text{jobu}$ and $\text{jobvt}$ cannot both be $'O'$.
|           | $'N'$: No columns of $U$ (no left singular vectors) are computed. |
| m         | The order of the matrix $A$. $m \geq 0$. |
| n         | The order of the matrix $A$. $n \geq 0$. |
| a         | $\text{UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n)}$. On entry, the $n$ by $n$ matrix $A$. On exit: |
|           | - If $\text{jobz} = 'O'$, $a$ is overwritten with the first $\min(m,n)$ columns of $U$ (the left singular vectors, stored columnwise). |
|           | - If $m \geq n$, $a$ is overwritten with the first $m$ rows of $V^*T$ (the right singular vectors, stored rowwise). |
|           | - If $\text{jobz} = 'O'$, the contents of $a$ are destroyed. |
| lda       | The leading dimension of the array $A$. $lda \geq \max(1,m)$. |
| s         | $\text{UTL_NLA_ARRAY_FLT/DBL, DIMENSION (min(m,n))}$. The singular values of $A$, sorted so that $S(i) \geq S(i+1)$. |
| u         | $\text{UTL_NLA_ARRAY_FLT/DBL, ucol = m if jobz = 'A' or jobz = 'O' and m < n; ucol = min(m,n) if jobz = 'S'.}$ |
|           | - If $\text{jobz} = 'A'$ or $\text{jobz} = 'O'$ and $m < n$, $u$ contains the $m$ by $m$ orthogonal matrix $U$. |
|           | - If $\text{jobz} = 'S'$, $u$ contains the first $\min(m,n)$ columns of $U$ (the left singular vectors, stored columnwise). |
|           | - If $\text{jobz} = 'O'$ and $m \geq n$, or $\text{jobz} = 'N'$, $u$ is not referenced. |
| ldu       | The leading dimension of the array $U$. $ldu \geq 1$. If $\text{jobz} = 'S'$ or $'A'$, or $\text{jobz} = 'O'$ and $m < n$, $ldu \geq m$. |
Table 272-42  (Cont.) LAPACK_GESDD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vt</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvt, n).</td>
</tr>
<tr>
<td></td>
<td>· If jobz = 'A' or jobz = 'O' and m &gt;= n, vt contains the n by n orthogonal matrix V**T.</td>
</tr>
<tr>
<td></td>
<td>· If jobz = 'S', vt contains the first min(m, n) rows of V**T (the right singular vectors, stored rowwise).</td>
</tr>
<tr>
<td></td>
<td>· If jobz = 'O' and m &lt; n, or jobz = 'N', vt is not referenced.</td>
</tr>
<tr>
<td>ldvt</td>
<td>The leading dimension of the array vt. ldvt &gt;= 1.</td>
</tr>
<tr>
<td></td>
<td>· If jobz = 'A', or jobz = 'O' and m &gt;= n, ldvt &gt;= n.</td>
</tr>
<tr>
<td></td>
<td>· If jobz = 'S', ldvt &gt;= min(m, n).</td>
</tr>
<tr>
<td>info</td>
<td>= 0 : successful exit</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 : If info = -i, the i-th argument had an illegal value</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 : SBDSDC did not converge, updating process failed.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>· 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>· 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.37 LAPACK_GESV Procedure

This procedure computes the solution to a real system of linear equations a * x = b, where a is an n by n matrix and x and b are n by nrhs matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

a = P * L * U

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of a is then used to solve the system of equations

a * x = b

See Also:
LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group

Syntax

UTL_NLA.LAPACK_GESV {
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  a      IN OUT   UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  ipiv   IN OUT   UTL_NLA_ARRAY_INT,
  b      IN OUT   UTL_NLA_ARRAY_DBL,
  ldb    IN      POSITIVEN,
  info   OUT     INTEGER,
  pack   IN      flag DEFAULT 'C');
UTL_NLA.LAPACK_GESV (  
  n    IN     POSITIVEN,  
  nrhs IN     POSITIVEN,  
  a    IN OUT  UTL_NLA_ARRAY_FLT,  
  lda  IN     POSITIVEN,  
  ipiv IN OUT  UTL_NLA_ARRAY_INT,  
  b    IN OUT  UTL_NLA_ARRAY_FLT,  
  ldb  IN     POSITIVEN,  
  info OUT INTEGER,  
  pack IN     flag DEFAULT 'C');

Parameters

Table 272-43  LAPACK_GESV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The number of linear equations, equivalent to the order of the matrix a. n &gt;= 0</td>
</tr>
<tr>
<td>nrhs</td>
<td>The number of right-hand sides, which is the number of columns of the matrix b. nrhs &gt;= 0.</td>
</tr>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the n by n coefficient matrix a. On exit, the factors L and U from the factorization a = P<em>L</em>U; the unit diagonal elements of L are not stored.</td>
</tr>
<tr>
<td>lda</td>
<td>The leading dimension of the array a. lda &gt;= max(1,n)</td>
</tr>
<tr>
<td>ipiv</td>
<td>INTEGER array, DIMENSION (n). The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row ipiv(i).</td>
</tr>
<tr>
<td>b</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0, the n by nrhs solution matrix X.</td>
</tr>
<tr>
<td>ldb</td>
<td>The leading dimension of the array b. ldb &gt;= max(1,n)</td>
</tr>
</tbody>
</table>
| info      | = 0 : successful exit  
|          | < 0 : if info = -i, the i-th argument had an illegal value  
|          | > 0 : if info = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed. |
| pack      | (Optional) Flags the packing of the matrices:  
|          | 'C': column-major (default)  
|          | 'R': row-major |

272.5.38 LAPACK_GESVD Procedures

This procedure computes the singular value decomposition (SVD) of a real m by n matrix A, optionally computing the left and/or right singular vectors.

The SVD is written
A = U * SIGMA * transpose(V)

where SIGMA is an \( m \times n \) matrix which is zero except for its \( \min(m,n) \) diagonal elements, \( U \) is an \( m \times m \) orthogonal matrix, and \( V \) is an \( n \times n \) orthogonal matrix. The diagonal elements of SIGMA are the singular values of \( A \), they are real and non-negative, and are returned in descending order. The first \( \min(m,n) \) columns of \( U \) and \( V \) are the left and right singular vectors of \( A \).

Note that the routine returns \( V^{*T} \), not \( V \).

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GESVD (  
jobu IN flag,  
jobvt IN flag,  
m IN POSITIVEN,  
n IN POSITIVEN,  
a IN OUT UTL_NLA_ARRAY_DBL,  
ida IN POSITIVEN,  
s IN OUT UTL_NLA_ARRAY_DBL,  
u IN OUT UTL_NLA_ARRAY_DBL,  
ldu IN POSITIVEN,  
vt IN OUT UTL_NLA_ARRAY_DBL,  
ldvt IN POSITIVEN,  
info OUT INTEGER,  
pack IN flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_GESVD (  
jobu IN flag,  
jobvt IN flag,  
m IN POSITIVEN,  
n IN POSITIVEN,  
a IN OUT UTL_NLA_ARRAY_FLT,  
ida IN POSITIVEN,  
s IN OUT UTL_NLA_ARRAY_FLT,  
u IN OUT UTL_NLA_ARRAY_FLT,  
ldu IN POSITIVEN,  
vt IN OUT UTL_NLA_ARRAY_FLT,  
ldvt IN POSITIVEN,  
info OUT INTEGER,  
pack IN flag DEFAULT 'C');
```
Parameters

### Table 272-44  LAPACK_GESVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jobu</strong></td>
<td>Specifies options for computing all or part of the matrix $U$:</td>
</tr>
<tr>
<td>· 'A': All $m$ columns of $U$ are returned in array $U$.</td>
<td></td>
</tr>
<tr>
<td>· 'S': The first $\min(m,n)$ columns of $U$ (the left singular vectors) are returned in the array $U$.</td>
<td></td>
</tr>
<tr>
<td>· 'O': The first $\min(m,n)$ columns of $U$ (the left singular vectors) are overwritten on the array $a$. jobu and jobvt cannot both be 'O'</td>
<td></td>
</tr>
<tr>
<td>· 'N': No columns of $U$ (no left singular vectors) are computed.</td>
<td></td>
</tr>
<tr>
<td><strong>jobvt</strong></td>
<td>Specifies options for computing all or part of the matrix $V^T$:</td>
</tr>
<tr>
<td>· 'A': All $n$ rows of $V^T$ are returned in the array $v_t$.</td>
<td></td>
</tr>
<tr>
<td>· 'S': The first $\min(m,n)$ rows of $V^T$ (the right singular vectors) are returned in the array $v_t$.</td>
<td></td>
</tr>
<tr>
<td>· 'O': The first $\min(m,n)$ rows of $V^T$ (the right singular vectors) are overwritten on the array $a$. jobvt and jobu cannot both be 'O'.</td>
<td></td>
</tr>
<tr>
<td>· 'N': No rows of $V^T$ (no right singular vectors) are computed.</td>
<td></td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>The order of the matrix $a$. $M &gt;= 0$.</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>The order of the matrix $a$. $N &gt;= 0$.</td>
</tr>
<tr>
<td><strong>a</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the $n$ by $n$ matrix $A$. On exit:</td>
</tr>
<tr>
<td>· If jobu = 'O', $A$ is overwritten with the first $\min(m,n)$ columns of $U$ (the left singular vectors, stored columnwise);</td>
<td></td>
</tr>
<tr>
<td>· If jobvt = 'O', $A$ is overwritten with the first $\min(m,n)$ rows of $V^T$ (the right singular vectors, stored rowwise);</td>
<td></td>
</tr>
<tr>
<td>· If jobu.ne.'O' and jobvt.ne.'O', the contents of $A$ are destroyed.</td>
<td></td>
</tr>
<tr>
<td><strong>lda</strong></td>
<td>The leading dimension of the array $a$. $lda &gt;= \max(1,n)$.</td>
</tr>
<tr>
<td><strong>s</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (min(m,n)). The singular values of $A$, sorted so that $S(i) &gt;= S(i+1)$.</td>
</tr>
<tr>
<td><strong>u</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldu,ucol).(ldu,m) if jobu = 'A' or (ldu,min(m,n)) if jobu = 'S'.</td>
</tr>
<tr>
<td>· If jobu = 'A', $U$ contains the $m$ by $m$ orthogonal matrix $U$.</td>
<td></td>
</tr>
<tr>
<td>· If jobu = 'S', $U$ contains the first $\min(m,n)$ columns of $U$ (the left singular vectors, stored columnwise).</td>
<td></td>
</tr>
<tr>
<td>· If jobu = 'N' or 'O', $U$ is not referenced.</td>
<td></td>
</tr>
<tr>
<td><strong>ldu</strong></td>
<td>The leading dimension of the array $U$. $ldu &gt;= 1$. If jobu = 'S' or 'A', ldu &gt; m.</td>
</tr>
<tr>
<td><strong>vt</strong></td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvt, n).</td>
</tr>
<tr>
<td>· If jobvt = 'A', $vt$ contains the $n$ by $n$ orthogonal matrix $V^T$.</td>
<td></td>
</tr>
<tr>
<td>· If jobvt = 'S', $vt$ contains the first $\min(m,n)$ rows of $V^T$ (the right singular vectors, stored rowwise).</td>
<td></td>
</tr>
<tr>
<td>· If jobvt = 'N' or 'O', $vt$ is not referenced.</td>
<td></td>
</tr>
</tbody>
</table>
Table 272-44  (Cont.) LAPACK_GESVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldvt</td>
<td>The leading dimension of the array vt. ldvt &gt;= 1.</td>
</tr>
<tr>
<td></td>
<td>· If jobvt = 'A', ldvt &gt;= n.</td>
</tr>
<tr>
<td></td>
<td>· If jobvt = 'S', ldvt &gt;= min(m,n).</td>
</tr>
<tr>
<td>info</td>
<td>= 0 : successful exit</td>
</tr>
<tr>
<td></td>
<td>· &lt; 0 : If info = -i, the i-th argument had an illegal value</td>
</tr>
<tr>
<td></td>
<td>· &gt; 0 : If SBDSDR did not converge, info specifies how many</td>
</tr>
<tr>
<td></td>
<td>superdiagonals of an intermediate bidiagonal form B did not</td>
</tr>
<tr>
<td></td>
<td>converge to zero.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>· 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>· 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.39 LAPACK_GEEV Procedures

This procedures computes for an \( n \) by \( n \) real nonsymmetric matrix \( A \), the eigenvalues and, optionally, the left and/or right eigenvectors.

- The right eigenvector \( v(j) \) of \( A \) satisfies \( A \ast v(j) = \lambda(j) \ast v(j) \) where \( \lambda(j) \) is its eigenvalue.
- The left eigenvector \( u(j) \) of \( A \) satisfies \( u(j)\astH \ast A = \lambda(j) \ast u(j)\astH \) where \( u(j)\astH \) denotes the conjugate transpose of \( u(j) \).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```plaintext
UTL_NLA.LAPACK_GEEV (  
    jobvl IN flag, 
    jobvr IN flag, 
    n IN POSITIVEN, 
    a IN OUT UTL_NLA_ARRAY_DBL, 
    lda IN POSITIVEN, 
    wr IN OUT UTL_NLA_ARRAY_DBL, 
    wi IN OUT UTL_NLA_ARRAY_DBL, 
    vl IN OUT UTL_NLA_ARRAY_DBL, 
    ldvl IN POSITIVEN, 
    vr IN OUT UTL_NLA_ARRAY_DBL, 
    ldvr IN POSITIVEN, 
    info OUT INTEGER, 
    pack IN flag DEFAULT 'C');
```
UTL_NLA.LAPACK_GEEV ( 
   jobvl IN flag, 
   jobvr IN flag, 
   n IN POSITIVEN, 
   a IN OUT UTL_NLA_ARRAY_FLT, 
   lda IN POSITIVEN, 
   wr IN OUT UTL_NLA_ARRAY_FLT, 
   wi IN OUT UTL_NLA_ARRAY_FLT, 
   vl IN OUT UTL_NLA_ARRAY_FLT, 
   ldvl IN POSITIVEN, 
   vr IN OUT UTL_NLA_ARRAY_FLT, 
   ldvr IN POSITIVEN, 
   info OUT INTEGER, 
   pack IN flag DEFAULT 'C');

Parameters

Table 272-45  LAPACK_GEEV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobvl</td>
<td>'N': Left eigenvectors of A are not computed.</td>
</tr>
<tr>
<td></td>
<td>'V': Left eigenvectors of A are computed.</td>
</tr>
<tr>
<td>jobvr</td>
<td>'N': Right eigenvectors of A are not computed.</td>
</tr>
<tr>
<td></td>
<td>'V': Right eigenvectors of A are computed.</td>
</tr>
<tr>
<td>n</td>
<td>The order of the matrix a. N &gt;= 0.</td>
</tr>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n).</td>
</tr>
<tr>
<td></td>
<td>On entry, the n by n matrix A.</td>
</tr>
<tr>
<td></td>
<td>On exit, A has been overwritten.</td>
</tr>
<tr>
<td>lda</td>
<td>The leading dimension of the array a. lda &gt;= max(1,n).</td>
</tr>
<tr>
<td>wr</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</td>
</tr>
<tr>
<td></td>
<td>wr and wi contain the real and imaginary parts respectively of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.</td>
</tr>
<tr>
<td>wi</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).</td>
</tr>
<tr>
<td></td>
<td>wr and wi contain the real and imaginary parts respectively of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.</td>
</tr>
<tr>
<td>vl</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</td>
</tr>
<tr>
<td></td>
<td>If jobvl = 'V', the left eigenvectors u(j) are stored one after another in the columns of vl, in the same order as their eigenvalues.</td>
</tr>
<tr>
<td></td>
<td>If jobvs = 'N', vl is not referenced.</td>
</tr>
<tr>
<td></td>
<td>If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of vl.</td>
</tr>
<tr>
<td></td>
<td>If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i<em>VL(:,j+1) and u(j+1) = VL(:,j) - i</em>VL(:,j+1).</td>
</tr>
<tr>
<td>ldvl</td>
<td>The leading dimension of the array vl. ldvl &gt;= 1. If jobvl = 'v', ldvl &gt;= n.</td>
</tr>
</tbody>
</table>
Table 272-45  (Cont.) LAPACK_GEEV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vr</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvr, n).</td>
</tr>
<tr>
<td></td>
<td>· If jobvr = 'V', the right eigenvectors ( v(j) ) are stored one after another in the columns of ( vr ), in the same order as their eigenvalues..</td>
</tr>
<tr>
<td></td>
<td>· If jobvr = 'N', vr is not referenced.</td>
</tr>
<tr>
<td></td>
<td>· If the ( j )-th eigenvalue is real, then ( v(j) = VR(:,j) ) the ( j )-th column of ( vr ).</td>
</tr>
<tr>
<td></td>
<td>· If the ( j )-th and ( (j+1) )-st eigenvalues form a complex conjugate pair, then ( v(j) = VR(:,j) + i<em>VR(:,j+1) ) and ( v(j+1) = VR(:,j) - i</em>VR(:,j+1) ).</td>
</tr>
<tr>
<td>ldvr</td>
<td>The leading dimension of the array ( vr ). ( ldvr &gt;= 1 ). If jobvr = 'V', ( ldvr &gt;= N ).</td>
</tr>
<tr>
<td>info</td>
<td>· = 0 : successful exit</td>
</tr>
<tr>
<td></td>
<td>· &lt; 0 : if info = -i, the ( i )-th argument had an illegal value</td>
</tr>
<tr>
<td></td>
<td>· &gt; 0 : if info = i, and ( i ) is &lt;= ( N ), the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed. Elements ( i+1:N ) of ( wr ) and ( wi ) contain eigenvalues which have converged..</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>· 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>· 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.40 LAPACK_GTSV Procedure

This procedure solves the equation \( a \times x = b \), where \( a \) is an \( n \) by \( n \) tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation \( a^{**}x = b \) may be solved by interchanging the order of the arguments \( du \) and \( dl \).

See Also:

LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_GTSV {
    n IN POSITIVEN,
    nrhs IN POSITIVEN,
    dl IN OUT UTL_NLA_ARRAY_DBL,
    d IN OUT UTL_NLA_ARRAY_DBL,
    du IN OUT UTL_NLA_ARRAY_DBL,
    b IN OUT UTL_NLA_ARRAY_DBL,
    ldb IN POSITIVEN,
    info OUT INTEGER,
    pack IN flag DEFAULT 'C');
```
UTL_NLA.LAPACK_GTSV (  
    n      IN      POSITIVEN,  
    nrhs   IN      POSITIVEN,  
    dl     IN OUT  UTL_NLA_ARRAYFLT,  
    d      IN OUT  UTL_NLA_ARRAYFLT,  
    du     IN OUT  UTL_NLA_ARRAYFLT,  
    b      IN OUT  UTL_NLA_ARRAYFLT,  
    ldb    IN      POSITIVEN,  
    info   OUT  INTEGER,  
    pack   IN      flag DEFAULT 'C');

Parameters

Table 272-46    LAPACK_GTSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The order of the matrix ( a ). ( n \geq 0 )</td>
</tr>
<tr>
<td>nrhs</td>
<td>The number of right-hand sides, which is the number of columns of the matrix ( b ). ( nrhs \geq 0 ).</td>
</tr>
</tbody>
</table>
| dl        | UTL_NLA_ARRAYFLT/DBL, DIMENSION \((n-1)\).  
            On entry, \( dl \) must contain the \((n-1)\) sub-diagonal elements of \( a \).  
            On exit, \( dl \) is overwritten by the \((n-2)\) elements of the second super-diagonal of the upper triangular matrix \( U \) from the \( LU \) factorization of \( a \), \( dl(1), \ldots, dl(n-2) \). |
| d         | UTL_NLA_ARRAYFLT/DBL, DIMENSION \((n)\).  
            On entry, \( d \) must contain the diagonal elements of \( a \).  
            On exit, \( d \) is overwritten by the \( n \) diagonal elements of \( U \). |
| du        | UTL_NLA_ARRAYFLT/DBL, DIMENSION \((n-1)\).  
            On entry, \( du \) must contain the \((n-1)\) super-diagonal elements of \( a \).  
            On exit, \( du \) is overwritten by the \((n-1)\) elements of the first super-diagonal of \( U \). |
| b         | UTL_NLA_ARRAYFLT/DBL, DIMENSION \((LDB, nrhs)\).  
            On entry, the \( n \) by \( nrhs \) matrix of right hand side matrix \( b \).  
            On exit, if \( info = 0 \), the \( n \) by \( nrhs \) solution matrix \( X \). |
| ldb       | The leading dimension of the array \( b \).  
            ldb \( \geq \) max \((1, n)\) |
| info      |  
            \( = 0 \): successful exit  
            \( < 0 \): if \( info = -i \), the \( i \)-th argument had an illegal value  
            \( > 0 \): if \( info = i \), \( U(i,i) \) is exactly zero, and the solution has not been computed. The factorization has not been completed unless \( i = n \). |
| pack      | (Optional) Flags the packing of the matrices:  
            \( 'C' \): column-major (default)  
            \( 'R' \): row-major |
272.5.41 LAPACK_PBSV Procedures

This procedure computes the solution to a real system of linear equations \( a \cdot x = b \), where \( a \) is an \( n \times n \) symmetric positive definite band matrix and \( x \) and \( b \) are \( n \times n_{\text{rhs}} \) matrices.

The Cholesky decomposition is used to factor \( A \) as

\[
A = U^* U \quad \text{if } \text{UPLO} = 'U'
\]

or

\[
A = L \cdot L^* \quad \text{if } \text{UPLO} = 'L'
\]

where \( U \) is an upper triangular matrix and \( L \) is a lower triangular matrix. The factored form of \( A \) is then used to solve the system of equations \( A \cdot X = B \).

See Also:

LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group

Syntax

```c
UTL_NLA.LAPACK_PBSV( 
  uplo       IN     flag,
  n          IN     POSITIVEN,
  kd         IN     NATURALN,
  nrhs       IN     POSITIVEN,
  ab         IN OUT  UTL_NLA_ARRAY_DBL,
  ldab       IN     POSITIVEN,
  b          IN OUT  UTL_NLA_ARRAY_DBL,
  ldb        IN     POSITIVEN,
  info       OUT    INTEGER,
  pack       IN     flag DEFAULT 'C');
```

```c
UTL_NLA.LAPACK_PBSV( 
  uplo       IN     flag,
  n          IN     POSITIVEN,
  kd         IN     NATURALN,
  nrhs       IN     POSITIVEN,
  ab         IN OUT  UTL_NLA_ARRAY_FLT,
  ldab       IN     POSITIVEN,
  b          IN OUT  UTL_NLA_ARRAY_FLT,
  ldb        IN     POSITIVEN,
  info       OUT    INTEGER,
  pack       IN     flag DEFAULT 'C');
```
Parameters

Table 272-47  LAPACK_PBSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | · uplo = 'U'. Upper triangular of A is stored.  
           | · uplo = 'L'. Lower triangular of A is stored. |
| n         | The number of linear equations, that is, the order of the matrix a. n >= 0 |
| kd        | The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0. |
| nrhs      | The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0. |
| ab        | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the upper or lower triangle of the symmetric band matrix a, stored in the first kd+1 rows of the array. The j-th column of a is stored in the j-th column of the array ab is as follows:  
           | · if uplo = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j;  
           | · if uplo = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD)  
           | .See below for further details. On exit, if info = 0, the triangular factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T of the bandmatrix A, in the same storage format as a. |
| ldab      | The leading dimension of the array ab. |
| ldb       | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0, the n by nrhs solution matrix X. |
| info      | = 0 : successful exit  
           | < 0 : if info = -i, the i-th argument had an illegal value  
           | > 0 : if info = i, the leading minor of order i of a is not positive definite, so the factorization could not be completed, and the solution has not been computed. |
| pack      | (Optional) Flags the packing of the matrices:  
           | · 'C': column-major (default)  
           | · 'R': row-major |

272.5.42 LAPACK_POSV Procedures

This procedure computes the solution to a real system of linear equations a * x = b, where a is an n by n symmetric positive definite matrix and x and b are n by nrhs matrices.

The Cholesky decomposition is used to factor A as

A = U**T* U if uplo = 'U'
or

\[ A = L \times L^T \text{ if } \text{UPLO} = 'L' \]

where \( U \) is an upper triangular matrix and \( L \) is a lower triangular matrix. The factored form of \( A \) is then used to solve the system of equations \( A \times X = B \).

See Also:

LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group

Syntax

```c
UTL_NLA.LAPACK_POSV(
    uplo IN flag,
    n IN POSITIVEN,
    nrhs IN POSITIVEN,
    a IN OUT UTL_NLA_ARRAY_DBL,
    lda IN POSITIVEN,
    b IN OUT UTL_NLA_ARRAY_DBL,
    ldb IN POSITIVEN,
    info OUT INTEGER,
    pack IN flag DEFAULT 'C');
```

```c
UTL_NLA.LAPACK_POSV(
    uplo IN flag,
    n IN POSITIVEN,
    nrhs IN POSITIVEN,
    a IN OUT UTL_NLA_ARRAY_FLT,
    lda IN POSITIVEN,
    b IN OUT UTL_NLA_ARRAY_FLT,
    ldb IN POSITIVEN,
    info OUT INTEGER,
    pack IN flag DEFAULT 'C');
```

Parameters

Table 272-48  LAPACK_POSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | \( \text{uplo} = 'U' \). Upper triangular of \( A \) is stored.  
|           | \( \text{uplo} = 'L' \). Lower triangular of \( A \) is stored. |
| n         | The number of linear equations, that is, the order of the matrix \( A \). \( n \geq 0 \) |
| nrhs      | The number of right-hand sides, which is the number of columns of the matrix \( B \). \( nrhs \geq 0 \). |
Table 272-48  (Cont.) LAPACK_POSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). If uplo = 'U', the leading NRHS n by n upper triangular part of a contains the upper NRHS triangular part of the matrix A, and the strictly lower NRHS triangular part of A is not referenced. If uplo = 'L', then rhs leading n by n lower triangular part of a contains the lower NRHS triangular part of the matrix a, and the strictly upper NRHS triangular part of a is not referenced. On exit, if info = 0, the factor U or L from the Cholesky factorization A = U<strong>T<em>U or A = L</em>L</strong>T.</td>
</tr>
<tr>
<td>lda</td>
<td>The leading dimension of the array a. lda &gt;= max (1, n)</td>
</tr>
<tr>
<td>b</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0, the n by nrhs solution matrix X.</td>
</tr>
<tr>
<td>ldb</td>
<td>The leading dimension of the array b. ldb &gt;= max (1, n)</td>
</tr>
<tr>
<td>info</td>
<td>・ = 0 : successful exit ・ &lt; 0 : if info = -i , the i-th argument had an illegal value ・ &gt; 0 : if info = i, the leading minor of order i of a is not positive definite, so the factorization could not be completed, and the solution has not been computed.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices: ・ 'C': column-major (default) ・ 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.43 LAPACK_PPSV Procedures

This procedure computes the solution to a real system of linear equations \( a * x = b \) where \( a \) is an \( n \) by \( n \) symmetric positive definite matrix stored in packed format and \( x \) and \( b \) are \( n \) by \( \text{nrhs} \) matrices.

The Cholesky decomposition is used to factor \( A \) as

\[
A = U**T* U \text{ if } \text{UPLO} = 'U'
\]

or

\[
A = L * L**T \text{ if } \text{UPLO} = 'L'
\]

where \( U \) is an upper triangular matrix and \( L \) is a lower triangular matrix. The factored form of \( A \) is then used to solve the system of equations \( A * X = B \).
## See Also:

LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group

### Syntax

```c
UTL_NLA.LAPACK_PPSV (  
  uplo IN flag,  
  n IN POSITIVEN,  
  nrhs IN POSITIVEN,  
  ap IN OUT UTL_NLA_ARRAY_DBL,  
  b IN OUT UTL_NLA_ARRAY_DBL,  
  ldb IN POSITIVEN,  
  info OUT INTEGER,  
  pack IN flag DEFAULT 'C');
```

```c
UTL_NLA.LAPACK_PPSV (  
  uplo IN flag,  
  n IN POSITIVEN,  
  nrhs IN POSITIVEN,  
  ap IN OUT UTL_NLA_ARRAY_FLT,  
  b IN OUT UTL_NLA_ARRAY_FLT,  
  ldb IN POSITIVEN,  
  info OUT INTEGER,  
  pack IN flag DEFAULT 'C');
```

### Parameters

Table 272-49  LAPACK_PPSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      |  • uplo = 'U'. Upper triangular of A is stored.  
           |  • uplo = 'L'. Lower triangular of A is stored.  |
| n         | The number of linear equations, that is, the order of the matrix a. n >= 0 |
| nrhs      | The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0. |
| ap        | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2).  
           | On entry, the upper or lower triangle of the symmetric matrix a, packed columnwise in a linear array. The j-th column of a is stored in the array ap as follows:  
           | If uplo = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j;  
           | If uplo = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n;  
           | On exit, if info = 0, the factor U or 'L' from the Cholesky factorization A = U**T*U or A = L*L**T in the same storage format as A. |

Oracle
Table 272-49  (Cont.) LAPACK_PPSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>UTL_NLA_ARRAY_FLT, DIMENSION (ldb, nrhs).&lt;br&gt;On entry, the n by nrhs matrix of right hand side matrix b.&lt;br&gt;On exit, if info = 0, the n by nrhs solution matrix X.</td>
</tr>
<tr>
<td>ldb</td>
<td>The leading dimension of the array b.&lt;br&gt;ldb &gt;= max(l,n)</td>
</tr>
<tr>
<td>info</td>
<td>• = 0 : successful exit&lt;br&gt;• &lt; 0 : if info = -i, the i-th argument had an illegal value&lt;br&gt;• &gt; 0 : if info = i, the leading minor of order i of a is not positive definite, so the factorization could not be completed, and the solution has not been computed.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:&lt;br&gt;• 'C': column-major (default)&lt;br&gt;• 'R': row-major</td>
</tr>
</tbody>
</table>

272.5.44 LAPACK_PTSV Procedures

This procedure computes the solution to a real system of linear equations $a \cdot x = b$, where $a$ is an $n$ by $n$ symmetric positive definite tridiagonal matrix, and $x$ and $b$ are $n$ by $nrhs$ matrices.

$a$ is factored as $A = L*D*L^T$, and the factored form of $a$ is then used to solve the system of equations.

See Also:
LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group

Syntax

```c
UTL_NLA_LAPACK_PTSV {
    n   IN   POSITIVEN,
    nrhs IN   POSITIVEN,
    d   IN OUT UTL_NLA_ARRAY_DBL,
    e   IN OUT UTL_NLA_ARRAY_DBL,
    b   IN OUT UTL_NLA_ARRAY_DBL,
    ldb IN   POSITIVEN,
    info OUT INTEGER,
    pack IN   flag DEFAULT 'C');
}

UTL_NLA_LAPACK_PTSV {
    n   IN   POSITIVEN,
    nrhs IN   POSITIVEN,
    d   IN OUT UTL_NLA_ARRAY_FLT,
    e   IN OUT UTL_NLA_ARRAY_FLT,
    b   IN OUT UTL_NLA_ARRAY_FLT,
}
```
Parameters

Table 272-50  LAPACK_PTSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The order of the matrix a. ( N \geq 0 ).</td>
</tr>
<tr>
<td>nrhs</td>
<td>The number of right-hand sides, which is the number of columns of the matrix ( b ). ( nrhs \geq 0 ).</td>
</tr>
<tr>
<td>d</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION ( (n) ). On entry, the ( n ) diagonal elements of the tridiagonal matrix ( a ). On exit, the ( n ) diagonal elements of the diagonal matrix ( d ) from the factorization ( A = L<em>D</em>L'^T ).</td>
</tr>
<tr>
<td>e</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION ( (n-1) ). On entry, the ( (n-1) ) subdiagonal elements of the tridiagonal matrix ( a ). On exit, the ( (n-1) ) diagonal elements of the unit bidiagonal factor ( L ) from the factorization ( A = L<em>D</em>L'^T ) of ( a ). ( e ) can also be regarded as the superdiagonal of the unit bidiagonal factor ( U ) from the ( U'^T<em>D</em>U ) factorization of ( a ).</td>
</tr>
<tr>
<td>b</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION ( (ldb, nrhs) ). On entry, the ( n ) by ( nrhs ) matrix of right hand side matrix ( b ). On exit, if ( info = 0 ), the ( n ) by ( nrhs ) solution matrix ( X ).</td>
</tr>
<tr>
<td>ldb</td>
<td>The leading dimension of the array ( b ). ( ldb \geq max(1,n) )</td>
</tr>
</tbody>
</table>
| info      | - 0 : successful exit  
- \( < 0 \) : if \( info = -i \), the \( i \)-th argument had an illegal value  
- \( > 0 \) : if \( info = i \), the leading minor of order \( i \) of \( a \) is not positive definite, so the factorization could not be completed, and the solution has not been computed. |
| pack      | (Optional) Flags the packing of the matrices:  
- 'C': column-major (default)  
- 'R': row-major |

272.5.45 LAPACK_SBEV Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix \( A \).
Syntax

```c
UTL_NLA.LAPACK_SBEV(
    jobz IN flag,
    uplo IN flag,
    n IN POSITIVEN,
    kd IN NATURALN,
    ab IN OUT UTL_NLA_ARRAY_DBL,
    ldab IN POSITIVEN,
    w IN OUT UTL_NLA_ARRAY_DBL,
    z IN OUT UTL_NLA_ARRAY_DBL,
    ldz IN POSITIVEN,
    info OUT INTEGER,
    pack IN flag DEFAULT 'C');
```

```c
UTL_NLA.LAPACK_SBEV(
    jobz IN flag,
    uplo IN flag,
    n IN POSITIVEN,
    kd IN NATURALN,
    ab IN OUT UTL_NLA_ARRAY_FLT,
    ldab IN POSITIVEN,
    w IN OUT UTL_NLA_ARRAY_FLT,
    z IN OUT UTL_NLA_ARRAY_FLT,
    ldz IN POSITIVEN,
    info OUT INTEGER,
    pack IN flag DEFAULT 'C');
```

Parameters

Table 272-51  LAPACK_SBEV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobz</td>
<td>'N': Compute eigenvalues only. 'V': Compute eigenvalues and eigenvectors.</td>
</tr>
<tr>
<td>uplo</td>
<td>'U': Upper triangle of A is stored. 'L': Lower triangle of A is stored.</td>
</tr>
<tr>
<td>n</td>
<td>The order of the matrix A. N &gt;= 0.</td>
</tr>
<tr>
<td>kd</td>
<td>The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if uplo = 'L'. kd &gt;= 0.</td>
</tr>
<tr>
<td>ab</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the upper or lower triangle of the symmetric band matrix A stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array ab:</td>
</tr>
<tr>
<td></td>
<td>• If uplo = 'U', ab(kd+i-i-j, j) = a(i, j) for max(1, j-kd) &lt;= i &lt;= j.</td>
</tr>
<tr>
<td></td>
<td>• If uplo = 'L', AB(l+i-j, j) = A(i, j) for j &lt;= i &lt;= min(n, j+kd).</td>
</tr>
<tr>
<td></td>
<td>On exit, ab is overwritten by values generated during the reduction to tridiagonal form:</td>
</tr>
<tr>
<td></td>
<td>• If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T are returned in rows kd and kd+1 of ab.</td>
</tr>
<tr>
<td></td>
<td>• If uplo = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of ab.</td>
</tr>
</tbody>
</table>
### Table 272-51  (Cont.) LAPACK_SBEV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldab</td>
<td>The leading dimension of the array ab. ldab &gt;= kd + 1.</td>
</tr>
<tr>
<td>w</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If info = 0, the eigenvalues in ascending order.</td>
</tr>
<tr>
<td>z</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). · If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i). · If jobz = 'N', then z is not referenced.</td>
</tr>
<tr>
<td>ldz</td>
<td>The leading dimension of the array z. ldz &gt;= 1, and if jobz = 'V', ldz &gt;= max(1,n).</td>
</tr>
<tr>
<td>info</td>
<td>= 0 : successful exit &lt; 0 : if info = -i, the i-th argument had an illegal value &gt; 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices: · 'C': column-major (default) · 'R': row-major</td>
</tr>
</tbody>
</table>

### 272.5.46 LAPACK_SBEVD Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

#### See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

#### Syntax

```
UTL_NLA.LAPACK_SBEVD (  
    jobz    IN     flag,  
    uplo    IN     flag,  
    n       IN     POSITIVEN,  
    kd      IN     NATURALN,  
    ab      IN OUT  UTL_NLA_ARRAY_DBL,  
    ldab    IN     POSITIVEN,  
    w       IN OUT  UTL_NLA_ARRAY_DBL,  
    z       IN OUT  UTL_NLA_ARRAY_DBL,  
    ldz     IN     POSITIVEN,  
    info    OUT    INTEGER,  
    pack    IN     flag DEFAULT 'C');
```
UTL_NLA.LAPACK_SBEVD (  
  jobz IN flag,  
  uplo IN flag,  
  n IN POSITIVEN,  
  kd IN NATURALN,  
  ab IN OUT UTL_NLA_ARRAY_FLT,  
  ldab IN POSITIVEN,  
  w IN OUT UTL_NLA_ARRAY_FLT,  
  z IN OUT UTL_NLA_ARRAY_FLT,  
  ldz IN POSITIVEN,  
  info OUT INTEGER,  
  pack IN flag DEFAULT 'C');

Parameters

Table 272-52  LAPACK_SBEVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| jobz      | 'N': Compute eigenvalues only.  
           | 'V': Compute eigenvalues and eigenvectors. |
| uplo      | 'U': Upper triangle of A is stored.  
           | 'L': Lower triangle of A is stored. |
| n         | The order of the matrix a. N >= 0. |
| kd        | The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if uplo = 'L'. kd >= 0. |
| ab        | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n). On entry, the upper or lower triangle of the symmetric band matrix A stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array ab:  
           | If uplo = 'U', ab(kd+1+i-j,j) = a(i,j) for max(1,j-kd)<=i<=j.  
           | If uplo = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). |
| ldab      | The leading dimension of the array ab. ldab >= kd + 1. |
| w         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz,n). If info = 0, the eigenvalues in ascending order. |
| z         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).  
           | If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i).  
           | If jobz = 'N', then z is not referenced. |
| ldz       | The leading dimension of the array z. ldz >= 1, and if jobz = 'v', ldz >= max(1,n). |
Table 272-52  (Cont.) LAPACK_SBEVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| info      | · = 0: successful exit  
           · < 0: if info = -i, the i-th argument had an illegal value  
           · > 0: if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero |
| pack      | (Optional) Flags the packing of the matrices:  
           · 'C': column-major (default)  
           · 'R': row-major |

272.5.47 LAPACK_SPEV Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix \( A \) in packed storage.

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```c
UTL_NLA.LAPACK_SPEV (  
  jobz IN flag,  
  uplo IN flag,  
  n IN POSITIVEN,  
  ap IN OUT UTL_NLA_ARRAY_DBL,  
  w IN OUT UTL_NLA_ARRAY_DBL,  
  z IN OUT UTL_NLA_ARRAY_DBL,  
  ldz IN POSITIVEN,  
  info OUT INTEGER,  
  pack IN flag DEFAULT 'C');
```

```c
UTL_NLA.LAPACK_SPEV (  
  jobz IN flag,  
  uplo IN flag,  
  n IN POSITIVEN,  
  ap IN OUT UTL_NLA_ARRAY_FLT,  
  w IN OUT UTL_NLA_ARRAY_FLT,  
  z IN OUT UTL_NLA_ARRAY_FLT,  
  ldz IN POSITIVEN,  
  info OUT INTEGER,  
  pack IN flag DEFAULT 'C');
```
272.5.48 LAPACK_SPEVD Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix \( \mathbf{A} \) in packed storage. If eigenvectors are desired, it uses a divide and
conquer algorithm. The divide and conquer algorithm makes very mild assumptions about floating point arithmetic.

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```plaintext
UTL_NLA.LAPACK_SPEVD (  
  jobz    IN      flag,  
  uplo    IN      flag,  
  n       IN      POSITIVEN,  
  ap      IN OUT  UTL_NLA_ARRAY_DBL,  
  w       IN OUT  UTL_NLA_ARRAY_DBL,  
  z       IN OUT  UTL_NLA_ARRAY_DBL,  
  ldz     IN      POSITIVEN,  
  info    OUT     INTEGER,  
  pack    IN      flag DEFAULT 'C');
```

```plaintext
UTL_NLA.LAPACK_SPEVD (  
  jobz    IN      flag,  
  uplo    IN      flag,  
  n       IN      POSITIVEN,  
  ap      IN OUT  UTL_NLA_ARRAY_FLT,  
  w       IN OUT  UTL_NLA_ARRAY_FLT,  
  z       IN OUT  UTL_NLA_ARRAY_FLT,  
  ldz     IN      POSITIVEN,  
  info    OUT     INTEGER,  
  pack    IN      flag DEFAULT 'C');
```

Parameters

Table 272-54  LAPACK_SPEVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| jobz      | • ’N’: Compute eigenvalues only.  
           | • ’V’: Compute eigenvalues and eigenvectors.  |
| uplo      | • ’U’: Upper triangle of A is stored.  
           | • ’L’: Lower triangle of A is stored.  |
| n         | The order of the matrix a. N >= 0.  |
Table 272-54  (Cont.) LAPACK_SPEVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ap**    | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2). On entry, the upper or lower triangle of the symmetric matrix A packed columnwise in a linear array. The j-th column of A is stored in the array ap:  
  - If uplo = 'U', ap(i + (j-1)*j/2) = a(i,j) for 1<=i<=j.  
  - If uplo = 'L', ap(i + (j-1)*(2*n-j)/2) = a(i,j) for j<=i<=n. On exit, ap is overwritten by values generated during the reduction to tridiagonal form:  
  - If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A.  
  - If uplo = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A. |
| **w**     | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If info = 0, the eigenvalues in ascending order. |
| **z**     | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz,n).  
  - If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i).  
  - If jobz = 'N', then z is not referenced. |
| **ldz**   | The leading dimension of the array z. ldz >= 1, and if jobz = 'V', ldz >= max(1,n). |
| **info**  | = 0 : successful exit  
  < 0 : if info = -i, the i-th argument had an illegal value  
  > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero |
| **pack**  | (Optional) Flags the packing of the matrices:  
  - 'C': column-major (default)  
  - 'R': row-major |

272.5.49 LAPACK_SPSV Procedures

This procedure computes the solution to a real system of linear equations A * x = b, where A is an n by n symmetric matrix stored in packed format, and x and b are n by nrhs matrices. The diagonal pivoting method is used to factor A as

A = U * D * U**T, if UPLO = 'U'

or

A = L * D * L**T, if UPLO = 'L'
where \( U \) (or \( L \)) is a product of permutation and unit upper (lower) triangular matrices, and \( D \) is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of \( A \) is then used to solve the system of equations \( A \times X = B \).

### See Also:

LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group

### Syntax

```c
UTL_NLA.LAPACK_SPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info, pack);
```

### Parameters

#### Table 272-55   LAPACK_SPSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `uplo`    | - `uplo = 'U'`. Upper triangular of \( A \) is stored.  
        | - `uplo = 'L'`. Lower triangular of \( A \) is stored. |
| `n`       | The number of linear equations, which is the order of the matrix \( a \). \( n \geq 0 \). |
| `nrhs`    | The number of right-hand sides, which is the number of columns of the matrix \( b \). \( nrhs \geq 0 \). |
### Table 272-55 (Cont.) LAPACK_SPSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ap</strong></td>
<td>UTL_NLA_ARRAYFLT, DIMENSION (n*(n+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array ap as follows:</td>
</tr>
<tr>
<td></td>
<td>- uplo = 'U': AP(i + (j-1)*j/2) = A(i,j) for 1&lt;=i&lt;=j</td>
</tr>
<tr>
<td></td>
<td>- uplo = 'L': AP(i + (j-1)*(2n-j)/2) = A(i,j) for j&lt;=i&lt;=n</td>
</tr>
<tr>
<td></td>
<td>See below for further details. On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U<em>D</em>U<strong>T or A = L<em>D</em>L</strong>T as computed by SSPTRF, stored as a packed triangular matrix in the same storage format as A.</td>
</tr>
<tr>
<td><strong>ipiv</strong></td>
<td>INTEGER array, DIMENSION (n). Details of the interchanges and the block structure of d, as determined by SSPTRF.</td>
</tr>
<tr>
<td></td>
<td>- If ipiv(k) &gt; 0, then rows and columns k and ipiv(k) were interchanged, and d(k,k) is a 1 by 1 diagonal block.</td>
</tr>
<tr>
<td></td>
<td>- If uplo = 'U' and ipiv(k) = ipiv(k-1) &lt; 0, then rows and columns k-1 and -ipiv(k) were interchanged and d(k-1:k,k-1:k) is a 2 by 2 diagonal block.</td>
</tr>
<tr>
<td></td>
<td>- If uplo = 'L' and ipiv(k) = ipiv(k+1) &lt; 0, then rows and columns k+1 and -ipiv(k) were interchanged and d(k:k+1,k:k+1) is a 2 by 2 diagonal block.</td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>UTL_NLA_ARRAYFLT, DIMENSION (ldb, nrhs). On entry, the n by nrhs right hand side matrix b. On exit, if info = 0, the n by nrhs solution matrix X.</td>
</tr>
<tr>
<td><strong>ldb</strong></td>
<td>The leading dimension of the array b. ldb &gt;= max(1,n)</td>
</tr>
<tr>
<td><strong>info</strong></td>
<td>- = 0 : successful exit</td>
</tr>
<tr>
<td></td>
<td>- &lt; 0 : if info = -i, the i-th argument had an illegal value</td>
</tr>
<tr>
<td></td>
<td>- &gt; 0 : if info = i, d(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix d is exactly singular, so the solution could not be computed.</td>
</tr>
<tr>
<td><strong>pack</strong></td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>- 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>- 'R': row-major</td>
</tr>
</tbody>
</table>
272.5.50 LAPACK_STEV Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix $A$.

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```
UTL_NLA.LAPACK_STEV (    
  jobz IN flag, 
  n IN POSITIVEN, 
  d IN OUT UTL_NLA_ARRAY_DBL, 
  e IN OUT UTL_NLA_ARRAY_DBL, 
  z IN OUT UTL_NLA_ARRAY_DBL, 
  ldz IN POSITIVEN, 
  info OUT INTEGER, 
  pack IN flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_STEV (    
  jobz IN flag, 
  n IN POSITIVEN, 
  d IN OUT UTL_NLA_ARRAY_FLT, 
  e IN OUT UTL_NLA_ARRAY_FLT, 
  z IN OUT UTL_NLA_ARRAY_FLT, 
  ldz IN POSITIVEN, 
  info OUT INTEGER, 
  pack IN flag DEFAULT 'C');
```

Parameters

Table 272-56  LAPACK_STEV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| jobz      | - 'N': Compute eigenvalues only.  
           | - 'V': Compute eigenvalues and eigenvectors.  
| n         | The order of the matrix $A$. $N \geq 0$.  
| d         | $\text{UTL\_NLA\_ARRAY\_FLT/DBL, DIMENSION (n)}$.  
           | - On entry, the $n$ diagonal elements of the tridiagonal matrix $A$.  
           | - On exit, if info = 0, the eigenvalues in ascending order.  
| e         | $\text{UTL\_NLA\_ARRAY\_FLT/DBL, DIMENSION (n)}$.  
           | - On entry, the $(n-1)$ subdiagonal elements of the tridiagonal matrix $A$, stored in elements 1 to $n-1$ of $e(n)$ need not be set, but is used by the subprogram.  
           | - On exit, the contents of $e$ are destroyed.  

### Table 272-56  (Cont.) LAPACK_STEV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| z | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).  
  - If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with d(i).  
  - If jobz = 'N', then z is not referenced. |
| ldz | The leading dimension of the array z. ldz >= 1, and if jobz = 'v', ldz >= max(1,n). |
| info | = 0 : successful exit  
  < 0 : if info = -i, the i-th argument had an illegal value  
  > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero |
| pack | (Optional) Flags the packing of the matrices:  
  - 'C': column-major (default)  
  - 'R': row-major |

### 272.5.51 LAPACK_STEVD Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

**See Also:**

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

**Syntax**

```c
UTL_NLA.LAPACK_STEVD ( 
  jobz     IN      flag, 
  n        IN      POSITIVEN, 
  d        IN OUT  UTL_NLA_ARRAY_DBL, 
  e        IN OUT  UTL_NLA_ARRAY_DBL, 
  z        IN OUT  UTL_NLA_ARRAY_DBL, 
  ldz      IN      POSITIVEN, 
  info     OUT     INTEGER, 
  pack     IN      flag DEFAULT 'C');
```

```c
UTL_NLA.LAPACK_STEVD( 
  jobz   IN    flag, 
  n     IN    POSITIVEN, 
  d     IN OUT  UTL_NLA_ARRAY_FLT, 
  e     IN OUT  UTL_NLA_ARRAY_FLT, 
  z     IN OUT  UTL_NLA_ARRAY_FLT, 
  ldz   IN    POSITIVEN, 
  info  OUT   INTEGER, 
  pack IN    flag DEFAULT 'C');
```
Parameters

Table 272-57  LAPACK_STEVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| jobz      | 'N': Compute eigenvalues only.  
           | 'V': Compute eigenvalues and eigenvectors. |
| n         | The order of the matrix \( A \). \( N \geq 0 \). |
| d         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).  
           | On entry, the \( n \) diagonal elements of the tridiagonal matrix \( A \).  
           | On exit, if info = 0, the eigenvalues in ascending order. |
| e         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).  
           | On entry, the \( (n-1) \) subdiagonal elements of the tridiagonal matrix \( A \), stored in elements \( 1 \) to \( n-1 \) of \( e \). \( e(n) \) need not be set, but is used by the subprogram.  
           | On exit, the contents of \( e \) are destroyed. |
| z         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).  
           | If jobz = 'V', then if info = 0, \( z \) contains the orthonormal eigenvectors of the matrix \( A \), with the \( i \)-th column of \( z \) holding the eigenvector associated with \( d(i) \).  
           | If jobz = 'N', then \( z \) is not referenced. |
| ldz       | The leading dimension of the array \( z \). \( ldz \geq 1 \), and if jobz = 'v', \( ldz \geq \max(1,n) \). |
| info      | = 0 : successful exit  
           | < 0 : if info = -i, the \( i \)-th argument had an illegal value  
           | > 0 : if info = i, the algorithm failed to converge; \( i \) off-diagonal elements of an intermediate tridiagonal form did not converge to zero |
| pack      | (Optional) Flags the packing of the matrices:  
           | 'C': column-major (default)  
           | 'R': row-major |

272.5.52 LAPACK_SYEV Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix \( A \).

See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

Syntax

```plaintext
UTL_NLA.LAPACK_SYEV (  
    jobz IN flag,  
    uplo IN flag,  
)```
UTL_NLA.LAPACK_SYEV (  
  jobz IN flag,  
  uplo IN flag,  
  n IN POSITIVEN,  
  a IN OUT UTL_NLA_ARRAY_FLT,  
  lda IN POSITIVEN,  
  w IN OUT UTL_NLA_ARRAY_FLT,  
  info OUT INTEGER,  
  pack IN flag DEFAULT 'C');

Parameters

Table 272-58  LAPACK_SYEV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| jobz      | · 'N': Compute eigenvalues only.  
            · 'V': Compute eigenvalues and eigenvectors.  
| uplo      | · 'U': Upper triangle of A is stored.  
            · 'L': Upper triangle of A is stored.  
| n         | The order of the matrix a. N >= 0.  
| a         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n).  
| lda       | The leading dimension of the array a. lda >= max(1,n).  
| w         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).  
| info      | · = 0 : successful exit  
            · < 0 : if info = -i, the i-th argument had an illegal value  
            · > 0 : if info = i, the algorithm failed to converge; i off-diagonal  
             elements of an intermediate tridiagonal form did not converge to  
             zero  
| pack      | (Optional) Flags the packing of the matrices:  
            · 'C': column-major (default)  
            · 'R': row-major  

Oracle
272.5.53 LAPACK_SYEVD Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

### See Also:

LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms for other subprograms in this group

### Syntax

[![Syntax image](image-url)](image-url)

### Parameters

#### Table 272-59  LAPACK_SYEVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| jobz      | 'N': Compute eigenvalues only.  
            | 'V': Compute eigenvalues and eigenvectors. |
| uplo      | 'U': Upper triangle of $A$ is stored.  
            | 'L': Upper triangle of $A$ is stored. |
| n         | The order of the matrix $a$. $N >= 0$. |
Table 272-59  (Cont.) LAPACK_SYEVD Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| a         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the symmetric matrix a:  
• If uplo = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a.  
• If uplo = 'L', the leading n by n lower triangular part of a contains the lower triangular part of the matrix a. On exit:  
• If jobz = 'V', then if info = 0, a contains the orthonormal eigenvectors of the matrix a.  
• If jobz = 'N', then on exit the lower triangle (if uplo = 'L') or the upper triangle (if uplo = 'U') of a, including the diagonal, is destroyed. |
| lda       | The leading dimension of the array a. lda >= max(1,n). |
| w         | UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If info = 0, the eigenvalues in ascending order. |
| info      |  
• = 0 : successful exit  
• < 0 : if info = -i, the i-th argument had an illegal value  
• > 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero |
| pack      | (Optional) Flags the packing of the matrices:  
• 'C': column-major (default)  
• 'R': row-major |

272.5.54 LAPACK_SYSV Procedures

This procedure computes the solution to a real system of linear equations a * x = b, where a is an n by n symmetric matrix, and x and b are n by nrhs matrices.

The diagonal pivoting method is used to factor A as

A = U * D * U**T, if UPLO = 'U'  

or  

A = L * D * L**T, if UPLO = 'L'  

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

See Also:  

LAPACK Driver Routines (Linear Equations) Subprograms for other subprograms in this group
Syntax

UTL_NLA.LAPACK_SYSV {
    uplo IN flag,
    n IN POSITIVEN,
    nrhs IN POSITIVEN,
    a IN OUT UTL_NLA_ARRAY_DBL,
    lda IN POSITIVEN,
    ipiv IN OUT UTL_NLA_ARRAY_INT,
    b IN OUT UTL_NLA_ARRAY_DBL,
    ldb IN POSITIVEN,
    info OUT INTEGER,
    pack IN flag DEFAULT 'C');

UTL_NLA.LAPACK_SYSV {
    uplo IN flag,
    n IN POSITIVEN,
    nrhs IN POSITIVEN,
    a IN OUT UTL_NLA_ARRAY_FLT,
    lda IN POSITIVEN,
    ipiv IN OUT UTL_NLA_ARRAY_INT,
    b IN OUT UTL_NLA_ARRAY_FLT,
    ldb IN POSITIVEN,
    info OUT INTEGER,
    pack IN flag DEFAULT 'C');

Parameters

Table 272-60  LAPACK_SYSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| uplo      | - uplo = 'U'. Upper triangular of A is stored.  
           - uplo = 'L'. Lower triangular of A is stored. |
| n         | The number of linear equations, which is the order of the matrix a. N >= 0. |
| nrhs      | The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0. |
| a         | UTL_NLA ARRAY_FLT/DBL, DIMENSION (n-1).  
            On entry, the symmetric matrix a. If UPLO = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a, and the strictly lower triangular part of a is not referenced.  
            If uplo = 'L', the leading n by n lower triangular part of a contains the lower triangular part of the matrix a, and the strictly upper triangular part of a is not referenced.  
            On exit, if info = 0, the block diagonal matrix d and the multipliers used to obtain the factor U or L from the factorization A = U*U**T or A = L*L**T as computed by SSYTRF. |
| lda       | The leading dimension of the array a.  
            lda >= max(1,n) |
Table 272-60  (Cont.) LAPACK_SYSV Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipiv</td>
<td>INTEGER array, DIMENSION (ldb, nrhs). Details of the interchanges and the block structure of d, as determined by SSYTRF.</td>
</tr>
<tr>
<td></td>
<td>• If ipiv(k) &gt; 0, then rows and columns k and ipiv(k) were interchanged, and d(k,k) is a 1 by 1 diagonal block.</td>
</tr>
<tr>
<td></td>
<td>• If uplo = 'U' and ipiv(k) = ipiv(k-1) &lt; 0, then rows and columns k-1 and -ipiv(k) were interchanged and d(k-1:k-1:k-1:k) is a 2 by 2 diagonal block.</td>
</tr>
<tr>
<td></td>
<td>• If uplo = 'L' and ipiv(k) = ipiv(k+1) &lt; 0, then rows and columns k+1 and -ipiv(k) were interchanged and d(k:k+1,k:k+1) is a 2 by 2 diagonal block.</td>
</tr>
<tr>
<td>b</td>
<td>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the n by nrhs matrix of right hand side matrix b. On exit, if info = 0, the n by nrhs solution matrix X.</td>
</tr>
<tr>
<td>ldb</td>
<td>The leading dimension of the array b. ldb &gt;= max(1,n)</td>
</tr>
<tr>
<td>info</td>
<td>• = 0 : successful exit</td>
</tr>
<tr>
<td></td>
<td>• &lt; 0 : if info = -i, the i-th argument had an illegal value</td>
</tr>
<tr>
<td></td>
<td>• &gt; 0 : if info = i, d(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix d is exactly singular, so the solution could not be computed.</td>
</tr>
<tr>
<td>pack</td>
<td>(Optional) Flags the packing of the matrices:</td>
</tr>
<tr>
<td></td>
<td>• 'C': column-major (default)</td>
</tr>
<tr>
<td></td>
<td>• 'R': row-major</td>
</tr>
</tbody>
</table>
273 UTL_RAW

The UTL_RAW package provides SQL functions for manipulating RAW datatypes.

This chapter contains the following topics:

- Overview
- Operational Notes
- Summary of UTL_RAW Subprograms

273.1 UTL_RAW Overview

This package is necessary because normal SQL functions do not operate on RAWs, and PL/SQL does not allow overloading between a RAW and a CHAR datatype. UTL_RAW also includes subprograms that convert various COBOL number formats to, and from, RAWs.

UTL_RAW is not specific to the database environment, and it may actually be used in other environments. For this reason, the prefix UTL has been given to the package, instead of DBMS.

273.2 UTL_RAW Operational Notes

UTL_RAW allows a RAW "record" to be composed of many elements. By using the RAW datatype, character set conversion will not be performed, keeping the RAW in its original format when being transferred through remote procedure calls.

With the RAW functions, you can manipulate binary data that was previously limited to the hextoraw and rawtohex functions.

Note:

Notes on datatypes:

- The PLS_INTEGER and BINARY_INTEGER datatypes are identical. This document uses BINARY_INTEGER to indicate datatypes in reference information (such as for table types, record types, subprogram parameters, or subprogram return values), but may use either in discussion and examples.
- The INTEGER and NUMBER(38) datatypes are also identical. This document uses INTEGER throughout.

273.3 Summary of UTL_RAW Subprograms

This table lists the UTL_RAW subprograms and briefly describes them.
### Table 273-1  **UTL_RAW Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIT_AND Function</strong></td>
<td>Performs bitwise logical &quot;and&quot; of the values in RAW r1 with RAW r2 and returns the &quot;anded&quot; result RAW</td>
</tr>
<tr>
<td><strong>BIT_COMPLEMENT Function</strong></td>
<td>Performs bitwise logical &quot;complement&quot; of the values in RAW r and returns the &quot;complemented&quot; result RAW</td>
</tr>
<tr>
<td><strong>BIT_OR Function</strong></td>
<td>Performs bitwise logical &quot;or&quot; of the values in RAW r1 with RAW r2 and returns the &quot;or'd&quot; result RAW</td>
</tr>
<tr>
<td><strong>BIT_XOR Function</strong></td>
<td>Performs bitwise logical &quot;exclusive or&quot; of the values in RAW r1 with RAW r2 and returns the &quot;xor'd&quot; result RAW</td>
</tr>
<tr>
<td><strong>CAST_FROM_BINARY_DOUBLE Function</strong></td>
<td>Returns the RAW binary representation of a BINARY_DOUBLE value</td>
</tr>
<tr>
<td><strong>CAST_FROM_BINARY_FLOAT Function</strong></td>
<td>Returns the RAW binary representation of a BINARY_FLOAT value</td>
</tr>
<tr>
<td><strong>CAST_FROM_BINARY_INTEGER Function</strong></td>
<td>Returns the RAW binary representation of a BINARY_INTEGER value</td>
</tr>
<tr>
<td><strong>CAST_FROM_NUMBER Function</strong></td>
<td>Returns the RAW binary representation of a NUMBER value</td>
</tr>
<tr>
<td><strong>CAST_TO_BINARY_DOUBLE Function</strong></td>
<td>Casts the RAW binary representation of a BINARY_DOUBLE into a BINARY_DOUBLE</td>
</tr>
<tr>
<td><strong>CAST_TO_BINARY_FLOAT Function</strong></td>
<td>Casts the RAW binary representation of a BINARY_FLOAT into a BINARY_FLOAT</td>
</tr>
<tr>
<td><strong>CAST_TO_BINARY_INTEGER Function</strong></td>
<td>Casts the RAW binary representation of a BINARY_INTEGER into a BINARY_INTEGER</td>
</tr>
<tr>
<td><strong>CAST_TO_NUMBER Function</strong></td>
<td>Casts the RAW binary representation of a NUMBER into a NUMBER</td>
</tr>
<tr>
<td><strong>CAST_TO_NVARCHAR2 Function</strong></td>
<td>Converts a RAW value into a VARCHAR2 value</td>
</tr>
<tr>
<td><strong>CAST_TO_RAW Function</strong></td>
<td>Converts a VARCHAR2 value into a RAW value</td>
</tr>
<tr>
<td><strong>CAST_TO_VARCHAR2 Function</strong></td>
<td>Converts a RAW value into a VARCHAR2 value</td>
</tr>
<tr>
<td><strong>COMPARE Function</strong></td>
<td>Compares RAW r1 against RAW r2</td>
</tr>
<tr>
<td><strong>CONCAT Function</strong></td>
<td>Concatenates up to 12 RAWs into a single RAW</td>
</tr>
<tr>
<td><strong>CONVERT Function</strong></td>
<td>Converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW</td>
</tr>
<tr>
<td><strong>COPIES Function</strong></td>
<td>Returns n copies of r concatenated together</td>
</tr>
<tr>
<td><strong>LENGTH Function</strong></td>
<td>Returns the length in bytes of a RAW r</td>
</tr>
<tr>
<td><strong>OVERLAY Function</strong></td>
<td>Overlays the specified portion of target RAW with overlay RAW, starting from byte position pos of target and proceeding for len bytes</td>
</tr>
<tr>
<td><strong>REVERSE Function</strong></td>
<td>Reverses a byte sequence in RAW r from end to end</td>
</tr>
<tr>
<td><strong>SUBSTR Function</strong></td>
<td>Returns len bytes, starting at pos from RAW r</td>
</tr>
<tr>
<td><strong>TRANSLATE Function</strong></td>
<td>Translates the bytes in the input RAW r according to the bytes in the translation RAWs from_set and to_set</td>
</tr>
</tbody>
</table>
Table 273-1 (Cont.) UTL_RAW Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSLITERATE Function</td>
<td>Converts the bytes in the input RAW r according to the bytes in the transliteration RAWs from_set and to_set</td>
</tr>
<tr>
<td>XRANGE Function</td>
<td>Returns a RAW containing all valid 1-byte encodings in succession, beginning with the value start_byte and ending with the value end_byte</td>
</tr>
</tbody>
</table>

273.3.1 BIT_AND Function

This function performs bitwise logical "and" of the values in RAW r1 with RAW r2 and returns the "anded" result RAW.

Syntax

```sql
UTL_RAW.BIT_AND (  
  r1 IN RAW,  
  r2 IN RAW)  
RETURN RAW;
```

Pragmas

```sql
pragma restrict_references(bit_and, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-2 BIT_AND Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>RAW to &quot;and&quot; with r2</td>
</tr>
<tr>
<td>r2</td>
<td>RAW to &quot;and&quot; with r1</td>
</tr>
</tbody>
</table>

Return Values

Table 273-3 BIT_AND Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the &quot;and&quot; of r1 and r2</td>
</tr>
<tr>
<td>NULL</td>
<td>Either r1 or r2 input parameter was NULL</td>
</tr>
</tbody>
</table>

Usage Notes

If r1 and r2 differ in length, the and operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.
273.3.2 BIT_COMPLEMENT Function

This function performs bitwise logical "complement" of the values in RAW r and returns the complemented result RAW. The result length equals the input RAW r length.

Syntax

```
UTL_RAW.BIT_COMPLEMENT (  
  r IN RAW)  
  RETURN RAW;  
```

Pragmas

```
pragma restrict_references(bit_complement, WNDS, RNDS, WNPS, RNPS);  
```

Parameters

Table 273-4 BIT_COMPLEMENT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW to perform &quot;complement&quot; operation</td>
</tr>
</tbody>
</table>

Return Values

Table 273-5 BIT_COMPLEMENT Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>The &quot;complement&quot; of r1</td>
</tr>
<tr>
<td>NULL</td>
<td>If r input parameter was NULL</td>
</tr>
</tbody>
</table>

273.3.3 BIT_OR Function

This function performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the or'd result RAW.

Syntax

```
UTL_RAW.BIT_OR (  
  r1 IN RAW,  
  r2 IN RAW)  
  RETURN RAW;  
```

Pragmas

```
pragma restrict_references(bit_or, WNDS, RNDS, WNPS, RNPS);  
```
Parameters

Table 273-6  BIT_OR Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>RAW to &quot;or&quot; with r2</td>
</tr>
<tr>
<td>r2</td>
<td>RAW to &quot;or&quot; with r1</td>
</tr>
</tbody>
</table>

Return Values

Table 273-7  BIT_OR Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the &quot;or&quot; of r1 and r2</td>
</tr>
<tr>
<td>NULL</td>
<td>Either r1 or r2 input parameter was NULL</td>
</tr>
</tbody>
</table>

Usage Notes

If r1 and r2 differ in length, then the "or" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

273.3.4 BIT_XOR Function

This function performs bitwise logical "exclusive or" of the values in RAW r1 with RAW r2 and returns the xor'd result RAW.

Syntax

UTL_RAW.BIT_XOR (  
r1 IN RAW,  
r2 IN RAW)  
RETURN RAW;  

Pragmas

pragma restrict_references(bit_xor, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 273-8  BIT_XOR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>RAW to &quot;xor&quot; with r2</td>
</tr>
<tr>
<td>r2</td>
<td>RAW to &quot;xor&quot; with r1</td>
</tr>
</tbody>
</table>
Return Values

Table 273-9  BIT_XOR Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the “xor” of r1 and r2</td>
</tr>
<tr>
<td>NULL</td>
<td>If either r1 or r2 input parameter was NULL</td>
</tr>
</tbody>
</table>

Usage Notes

If r1 and r2 differ in length, then the “xor” operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

273.3.5 CAST_FROM_BINARY_DOUBLE Function

This function returns the RAW binary representation of a BINARY_DOUBLE value.

Syntax

```
UTL_RAW.CAST_FROM_BINARY_DOUBLE(
    n          IN BINARY_DOUBLE,
    endianess IN PLS_INTEGER DEFAULT 1)
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_binary_double, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-10  CAST_FROM_BINARY_DOUBLE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>BINARY_DOUBLE value</td>
</tr>
<tr>
<td>endianess</td>
<td>A BINARY_INTEGER value indicating the endianess. The function recognizes the</td>
</tr>
<tr>
<td></td>
<td>defined constants big_endian (1), little_endian (2), and machine_endian (3).</td>
</tr>
<tr>
<td></td>
<td>The default is big_endian. A setting of machine_endian has the same effect</td>
</tr>
<tr>
<td></td>
<td>as big_endian on a big endian machine, or the same effect as little_endian on</td>
</tr>
<tr>
<td></td>
<td>a little endian machine.</td>
</tr>
</tbody>
</table>

Return Values

The binary representation of the BINARY_DOUBLE value, or NULL if the input is NULL.

Usage Notes

- An 8-byte binary_double value maps to the IEEE 754 double-precision format as follows:
  
  byte 0: bit 63 ~ bit 56
  byte 1: bit 55 ~ bit 48
• The parameter endianess describes how the bytes of BINARY_DOUBLE are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb7 refer to the bytes in raw and db0 ~ db7 refer to the bytes in BINARY_DOUBLE.

<table>
<thead>
<tr>
<th>endianess</th>
<th>rb0</th>
<th>rb1</th>
<th>rb2</th>
<th>rb3</th>
<th>rb4</th>
<th>rb5</th>
<th>rb6</th>
<th>rb7</th>
</tr>
</thead>
<tbody>
<tr>
<td>big_endian</td>
<td>db0</td>
<td>db1</td>
<td>db2</td>
<td>db3</td>
<td>db4</td>
<td>db5</td>
<td>db6</td>
<td>db7</td>
</tr>
<tr>
<td>little_endian</td>
<td>db7</td>
<td>db6</td>
<td>db5</td>
<td>db4</td>
<td>db3</td>
<td>db2</td>
<td>db1</td>
<td>db0</td>
</tr>
</tbody>
</table>

• In case of machine-endian, the 8 bytes of the BINARY_DOUBLE argument are copied straight across into the RAW return value. The effect is the same if the user has passed big_endian on a big-endian machine, or little_endian on a little-endian machine.

### 273.3.6 CAST_FROM_BINARY_FLOAT Function

This function returns the RAW binary representation of a BINARY_FLOAT value.

**Syntax**

```plaintext
UTL_RAW.CAST_FROM_BINARY_FLOAT(
    n          IN BINARY_FLOAT,
    endianess IN PLS_INTEGER DEFAULT 1)
RETURN RAW;
```

**Pragmas**

```plaintext
pragma restrict_references(cast_from_binary_float, WNDS, RNDS, WNPS, RNPS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Table 273-11</th>
<th>CAST_FROM_BINARY_FLOAT Function Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>n</td>
<td>BINARY_FLOAT value</td>
</tr>
<tr>
<td>endianess</td>
<td>A BINARY_INTEGER value indicating the endianess. The function recognizes the defined constants big_endian (1), little_endian (2), and machine_endian (3). The default is big_endian. A setting of machine_endian has the same effect as big_endian on a big endian machine, or the same effect as little_endian on a little endian machine.</td>
</tr>
</tbody>
</table>

**Return Values**

The binary representation (RAW) of the BINARY_FLOAT value, or NULL if the input is NULL.

**Usage Notes**

- A 4-byte binary_float value maps to the IEEE 754 single-precision format as follows:
The parameter endianess describes how the bytes of BINARY_FLOAT are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb3 refer to the bytes in RAW and fb0 ~ fb3 refer to the bytes in BINARY_FLOAT.

<table>
<thead>
<tr>
<th>Endianess</th>
<th>rb0</th>
<th>rb1</th>
<th>rb2</th>
<th>rb3</th>
</tr>
</thead>
<tbody>
<tr>
<td>big_endian</td>
<td>fb0</td>
<td>fb1</td>
<td>fb2</td>
<td>fb3</td>
</tr>
<tr>
<td>little_endian</td>
<td>fb3</td>
<td>fb2</td>
<td>fb1</td>
<td>fb0</td>
</tr>
</tbody>
</table>

In case of machine-endian, the 4 bytes of the BINARY_FLOAT argument are copied straight across into the RAW return value. The effect is the same if the user has passed big_endian on a big-endian machine, or little_endian on a little-endian machine.

273.3.7 CAST_FROM_BINARY_INTEGER Function

This function returns the RAW binary representation of a BINARY_INTEGER value.

Syntax

```
UTL_RAW.CAST_FROM_BINARY_INTEGER (n IN BINARY_INTEGER, endianess IN PLS_INTEGER DEFAULT BIG_ENDIAN) RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_from_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-12  CAST_FROM_BINARY_INTEGER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>BINARY_INTEGER value.</td>
</tr>
<tr>
<td>endianess</td>
<td>A BINARY_INTEGER value indicating the endianess. The function recognizes the defined constants big_endian (1), little_endian (2), and machine_endian (3). The default is big_endian. A setting of machine_endian has the same effect as big_endian on a big endian machine, or the same effect as little_endian on a little endian machine.</td>
</tr>
</tbody>
</table>

Return Values

The binary representation of the BINARY_INTEGER value.
273.3.8 CAST_FROM_NUMBER Function

This function returns the RAW binary representation of a NUMBER value.

Syntax

```sql
UTL_RAW.CAST_FROM_NUMBER (
    n  IN NUMBER)
RETURN RAW;
```

Pragmas

```sql
pragma restrict_references(cast_from_number, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-13  CAST_FROM_NUMBER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>NUMBER value</td>
</tr>
</tbody>
</table>

Return Values

The binary representation of the NUMBER value.

273.3.9 CAST_TO_BINARY_DOUBLE Function

This function casts the RAW binary representation of a BINARY_DOUBLE into a BINARY_DOUBLE.

Syntax

```sql
UTL_RAW.CAST_TO_BINARY_DOUBLE (
    r          IN RAW
    endianess  IN PLS_INTEGER DEFAULT 1)
RETURN BINARY_DOUBLE;
```

Pragmas

```sql
pragma restrict_references(cast_to_binary_double, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-14  CAST_TO_BINARY_DOUBLE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Binary representation of a BINARY_DOUBLE</td>
</tr>
<tr>
<td>endianess</td>
<td>A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.</td>
</tr>
</tbody>
</table>

Return Values

The BINARY_DOUBLE value.
Usage Notes

- If the RAW argument is more than 8 bytes, only the first 8 bytes are used and the rest of the bytes are ignored. If the result is -0, +0 is returned. If the result is NaN, the value BINARY_DOUBLE_NAN is returned.
- If the RAW argument is less than 8 bytes, a VALUE_ERROR exception is raised.
- An 8-byte binary_double value maps to the IEEE 754 double-precision format as follows:
  
  byte 0: bit 63 ~ bit 56
  byte 1: bit 55 ~ bit 48
  byte 2: bit 47 ~ bit 40
  byte 3: bit 39 ~ bit 32
  byte 4: bit 31 ~ bit 24
  byte 5: bit 23 ~ bit 16
  byte 6: bit 15 ~ bit  8
  byte 7: bit  7 ~ bit  0

- The parameter endianess describes how the bytes of BINARY_DOUBLE are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb7 refer to the bytes in raw and db0 ~ db7 refer to the bytes in BINARY_DOUBLE.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>rb0</th>
<th>rb1</th>
<th>rb2</th>
<th>rb3</th>
<th>rb4</th>
<th>rb5</th>
<th>rb6</th>
<th>rb7</th>
</tr>
</thead>
<tbody>
<tr>
<td>big_endian</td>
<td>db0</td>
<td>db1</td>
<td>db2</td>
<td>db3</td>
<td>db4</td>
<td>db5</td>
<td>db6</td>
<td>db7</td>
</tr>
<tr>
<td>little_endian</td>
<td>db7</td>
<td>db6</td>
<td>db5</td>
<td>db4</td>
<td>db3</td>
<td>db2</td>
<td>db1</td>
<td>db0</td>
</tr>
</tbody>
</table>

- In case of machine-endian, the 8 bytes of the RAW argument are copied straight across into the BINARY_DOUBLE return value. The effect is the same if the user has passed big_endian on a big-endian machine, or little_endian on a little-endian machine.

273.3.10 CAST_TO_BINARY_FLOAT Function

This function casts the RAW binary representation of a BINARY_FLOAT into a BINARY_FLOAT.

Syntax

UTL_RAW.CAST_TO_BINARY_FLOAT (  
  r              IN RAW  
  endianess     IN PLS_INTEGER DEFAULT 1)  
RETURN BINARY_FLOAT;

Pragmas

pragma restrict_references(cast_to_binary_float, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 273-15  CAST_TO_BINARY_FLOAT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Binary representation of a BINARY_FLOAT</td>
</tr>
</tbody>
</table>
### Table 273-15  (Cont.) CAST_TO_BINARY_FLOAT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endianess</td>
<td>A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.</td>
</tr>
</tbody>
</table>

#### Return Values

The **BINARY_FLOAT** value.

#### Usage Notes

- If the **RAW** argument is more than 4 bytes, only the first 4 bytes are used and the rest of the bytes are ignored. If the result is -0, +0 is returned. If the result is NaN, the value **BINARY_FLOAT_NAN** is returned.
- If the **RAW** argument is less than 4 bytes, a **VALUE_ERROR** exception is raised.
- A 4-byte **binary_float** value maps to the IEEE 754 single-precision format as follows:
  - byte 0: bit 31 ~ bit 24
  - byte 1: bit 23 ~ bit 16
  - byte 2: bit 15 ~ bit 8
  - byte 3: bit 7 ~ bit 0
- The parameter endianess describes how the bytes of **BINARY_FLOAT** are mapped to the bytes of **RAW**. In the following matrix, rb0 ~ rb3 refer to the bytes in **RAW** and fb0 ~ fb3 refer to the bytes in **BINARY_FLOAT**.

<table>
<thead>
<tr>
<th>Endianness</th>
<th>rb0</th>
<th>rb1</th>
<th>rb2</th>
<th>rb3</th>
</tr>
</thead>
<tbody>
<tr>
<td>big_endian</td>
<td>fbo</td>
<td>fb1</td>
<td>fb2</td>
<td>fb3</td>
</tr>
<tr>
<td>little_endian</td>
<td>fb3</td>
<td>fb2</td>
<td>fb1</td>
<td>fb0</td>
</tr>
</tbody>
</table>

- In case of machine-endian, the 4 bytes of the **RAW** argument are copied straight across into the **BINARY_FLOAT** return value. The effect is the same if the user has passed **big_endian** on a big-endian machine, or **little_endian** on a little-endian machine.

### 273.3.11 CAST_TO_BINARY_INTEGER Function

This function casts the **RAW** binary representation of a **BINARY_INTEGER** into a **BINARY_INTEGER**.

#### Syntax

```sql
UTL_RAW.CAST_TO_BINARY_INTEGER (  
    r          IN RAW  
    endianess  IN PLS_INTEGER DEFAULT BIG_ENDIAN)  
RETURN BINARY_INTEGER;
```

#### Pragmas

```sql
pragma restrict_references(cast_to_binary_integer, WNDS, RNDS, WNPS, RNPS);
```
Parameters

Table 273-16  CAST_TO_BINARY_INTEGER Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Binary representation of a BINARY_INTEGER</td>
</tr>
<tr>
<td>endianess</td>
<td>A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.</td>
</tr>
</tbody>
</table>

Return Values

The BINARY_INTEGER value

273.3.12 CAST_TO_NUMBER Function

This function casts the RAW binary representation of a NUMBER into a NUMBER.

Syntax

```
UTL_RAW.CAST_TO_NUMBER (r IN RAW)
RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(cast_to_number, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-17  CAST_TO_NUMBER function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Binary representation of a NUMBER</td>
</tr>
</tbody>
</table>

Return Values

The NUMBER value.

273.3.13 CAST_TO_NVARCHAR2 Function

This function converts a RAW value represented using some number of data bytes into an NVARCHAR2 value with that number of data bytes.

Note:

When casting to a NVARCHAR2, the current Globalization Support character set is used for the characters within that NVARCHAR2 value.
Syntax

```
UTL_RAW.CAST_TO_NVARCHAR2 (    
    r IN RAW) 
RETURN NVARCHAR2;
```

Pragmas

```
pragma restrict_references(cast_to_NVARCHAR2, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-18  CAST_TO_NVARCHAR2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW (without leading length field) to be changed to a NVARCHAR2</td>
</tr>
</tbody>
</table>

Return Values

Table 273-19  CAST_TO_NVARCHAR2 Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVARCHAR2</td>
<td>Containing having the same data as the input RAW</td>
</tr>
<tr>
<td>NULL</td>
<td>If r input parameter was NULL</td>
</tr>
</tbody>
</table>

273.3.14 CAST_TO_RAW Function

This function converts a VARCHAR2 value represented using some number of data bytes into a RAW value with that number of data bytes. The data itself is not modified in any way, but its datatype is recast to a RAW datatype.

Syntax

```
UTL_RAW.CAST_TO_RAW (    
    c IN VARCHAR2) 
RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_to_raw, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-20  CAST_TO_RAW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>VARCHAR2 to be changed to a RAW</td>
</tr>
</tbody>
</table>
Return Values

Table 273-21  CAST_TO_RAW Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the same data as the input VARCHAR2 and equal byte length as the input VARCHAR2 and without a leading length field</td>
</tr>
<tr>
<td>NULL</td>
<td>If input parameter was NULL</td>
</tr>
</tbody>
</table>

273.3.15 CAST_TO_VARCHAR2 Function

This function converts a RAW value represented using some number of data bytes into a VARCHAR2 value with that number of data bytes.

**Note:**
When casting to a VARCHAR2, the current Globalization Support character set is used for the characters within that VARCHAR2.

Syntax

```sql
UTL_RAW.CAST_TO_VARCHAR2 (
  r IN RAW)
RETURN VARCHAR2;
```

Pragmas

```sql
pragma restrict_references(cast_to_VARCHAR2, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-22  CAST_TO_VARCHAR2 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW (without leading length field) to be changed to a VARCHAR2</td>
</tr>
</tbody>
</table>

Return Values

Table 273-23  CAST_TO_VARCHAR2 Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>Containing having the same data as the input RAW</td>
</tr>
<tr>
<td>NULL</td>
<td>If input parameter was NULL</td>
</tr>
</tbody>
</table>
273.3.16 COMPARE Function

This function compares two RAW values. If they differ in length, then the shorter is extended on the right according to the optional pad parameter.

Syntax

```sql
UTL_RAW.COMPARE (  
r1 IN RAW,  
r2 IN RAW,  
pad IN RAW DEFAULT NULL)  
RETURN NUMBER;
```

Pragmas

```sql
pragma restrict_references(compare, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-24  COMPARE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>1st RAW to be compared, may be NULL or 0 length</td>
</tr>
<tr>
<td>r2</td>
<td>2nd RAW to be compared, may be NULL or 0 length</td>
</tr>
</tbody>
</table>
| pad       | This is an optional parameter. Byte to extend whichever of r1 or r2 is shorter. The default: x'00'

Return Values

Table 273-25  COMPARE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>Equals 0 if RAW byte strings are both NULL or identical; or,</td>
</tr>
<tr>
<td></td>
<td>Equals position (numbered from 1) of the first mismatched byte</td>
</tr>
</tbody>
</table>

273.3.17 CONCAT Function

This function concatenates up to 12 RAWs into a single RAW. If the concatenated size exceeds 32K, then an error is returned.

Syntax

```sql
UTL_RAW.CONCAT (  
r1 IN RAW DEFAULT NULL,  
r2 IN RAW DEFAULT NULL,  
r3 IN RAW DEFAULT NULL,  
r4 IN RAW DEFAULT NULL,  
r5 IN RAW DEFAULT NULL,  
r6 IN RAW DEFAULT NULL,  
r7 IN RAW DEFAULT NULL,  
r8 IN RAW DEFAULT NULL,  
r9 IN RAW DEFAULT NULL,  
r10 IN RAW DEFAULT NULL,  
r11 IN RAW DEFAULT NULL,  
r12 IN RAW DEFAULT NULL)  
RETURN NUMBER;
```

Chapter 273

Summary of UTL_RAW Subprograms
r10 IN RAW DEFAULT NULL,
r11 IN RAW DEFAULT NULL,
r12 IN RAW DEFAULT NULL)
RETURN RAW;

Pragmas
pragma restrict_references(concat, WNDS, RNDS, WNPS, RNPS);

Parameters
r1....r12 are the RAW items to concatenate.

Return Values

Table 273-26 CONCAT Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the items concatenated</td>
</tr>
</tbody>
</table>

Exceptions

There is an error if the sum of the lengths of the inputs exceeds the maximum allowable length for a RAW, which is 32767 bytes.

273.3.18 CONVERT Function

This function converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW.

Both from_charset and to_charset must be supported character sets defined to the Oracle server.

Syntax

UTL_RAW.CONVERT (  
r            IN RAW,
    to_charset   IN VARCHAR2,
    from_charset IN VARCHAR2)  
RETURN RAW;

Pragmas
pragma restrict_references(convert, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 273-27 CONVERT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW byte-string to be converted</td>
</tr>
<tr>
<td>to_charset</td>
<td>Name of the character set to which r is converted</td>
</tr>
<tr>
<td>from_charset</td>
<td>Name of the character set in which r is supplied</td>
</tr>
</tbody>
</table>
Return Values

Table 273-28  CONVERT Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Byte string r converted according to the specified character sets.</td>
</tr>
</tbody>
</table>

Exceptions

Table 273-29  CONVERT Function Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-06502</td>
<td>PL/SQL: numeric or value error</td>
</tr>
<tr>
<td>ORA-12703</td>
<td>This character set conversion is not supported</td>
</tr>
<tr>
<td>ORA-12705</td>
<td>Cannot access NLS data files or invalid environment specified</td>
</tr>
</tbody>
</table>

Usage Notes

- The NLS_LANG parameter form language_territory.character set is also accepted for to_charset and from_charset. However, this form is deprecated and should be avoided. Note that language and territory are ignored by this subprogram.
- The converted value is silently truncated if it exceeds the maximum length of a RAW value, which is 32767 bytes. Do not convert values longer than floor(32767/4) = 8191 bytes if you want to avoid this truncation for all possible combinations of to_charset and from_charset. You can use the maximum character width of the target character set to_charset, if known, to expand the limit to a less pessimistic value. For example, if the target character set is ZHS16GBK, the maximum safe source string length is floor(32767/2) = 16383 bytes. For single-byte target character sets, no truncation is ever necessary.

273.3.19 COPIES Function

This function returns n copies of r concatenated together.

Syntax

UTL_RAW.COPIES ( 
    r IN RAW, 
    n IN NUMBER) 
RETURN RAW;

Pragmas

pragma restrict_references(copies, WNDS, RNDS, WNPS, RNPS);
Parameters

Table 273-30  COPIES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW to be copied</td>
</tr>
<tr>
<td>n</td>
<td>Number of times to copy the RAW (must be positive)</td>
</tr>
</tbody>
</table>

Return Values

This returns the RAW copied n times.

Exceptions

Table 273-31  COPIES Function Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- r is missing, NULL or 0 length</td>
</tr>
<tr>
<td></td>
<td>- n &lt; 1</td>
</tr>
<tr>
<td></td>
<td>- Length of result exceeds maximum length of a RAW</td>
</tr>
</tbody>
</table>

273.3.20 LENGTH Function

This function returns the length in bytes of a RAW r.

Syntax

```
UTL_RAW.LENGTH (r IN RAW)
RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(length, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-32  LENGTH Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW byte stream to be measured</td>
</tr>
</tbody>
</table>

Return Values

Table 273-33  LENGTH Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>Current length of the RAW</td>
</tr>
</tbody>
</table>
273.3.21 OVERLAY Function

This function overlays the specified portion of target RAW with overlay_str RAW, starting from byte position pos of target and proceeding for len bytes.

Syntax

```sql
UTL_RAW.OVERLAY (
  overlay_str IN RAW,
  target      IN RAW,
  pos         IN BINARY_INTEGER DEFAULT 1,
  len         IN BINARY_INTEGER DEFAULT NULL,
  pad         IN RAW            DEFAULT NULL)
RETURN RAW;
```

Pragmas

```sql
pragma restrict_references(overlay, WNDS, RNDS, WNPS, RNPS);
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overlay_str</td>
<td>Byte-string used to overlay target</td>
</tr>
<tr>
<td>target</td>
<td>Byte-string which is to be overlaid</td>
</tr>
<tr>
<td>pos</td>
<td>Position in target (numbered from 1) to start overlay</td>
</tr>
<tr>
<td>len</td>
<td>The number of target bytes to overlay</td>
</tr>
<tr>
<td>pad</td>
<td>Pad byte used when overlay len exceeds overlay_str length or pos exceeds target length</td>
</tr>
</tbody>
</table>

Defaults and Optional Parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>1</td>
</tr>
<tr>
<td>len</td>
<td>To the length of overlay_str</td>
</tr>
<tr>
<td>pad</td>
<td>x'00'</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>The target byte_string overlaid as specified.</td>
</tr>
</tbody>
</table>
Usage Notes

If overlay_str has less than len bytes, then it is extended to len bytes using the pad byte. If overlay_str exceeds len bytes, then the extra bytes in overlay_str are ignored. If len bytes beginning at position pos of target exceeds the length of target, then target is extended to contain the entire length of overlay_str.

If len is specified, it must be greater than or equal to 0. If pos is specified, it must be greater than or equal to 1. If pos exceeds the length of target, then target is padded with pad bytes to position pos, and target is further extended with overlay_str bytes.

Exceptions

Table 273-37  OVERLAY Function Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- Overlay_str is NULL or has 0 length</td>
</tr>
<tr>
<td></td>
<td>- Target is missing or undefined</td>
</tr>
<tr>
<td></td>
<td>- Length of target exceeds maximum length of a RAW</td>
</tr>
<tr>
<td></td>
<td>- len &lt; 0</td>
</tr>
<tr>
<td></td>
<td>- pos &lt; 1</td>
</tr>
</tbody>
</table>

273.3.22 REVERSE Function

This function reverses a byte sequence in RAW r from end to end.

For example, x'0102F3' would be reversed to x'F30201', and 'xyz' would be reversed to 'zyx'. The result length is the same as the input RAW length.

Syntax

```
UTL_RAW.REVERSE (  
  r IN RAW)  
RETURN RAW;
```

Pragmas

```
pragma restrict_references(reverse, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-38  REVERSE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW to reverse</td>
</tr>
</tbody>
</table>
Return Values

Table 273-39 REVERSE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing the &quot;reverse&quot; of ( r )</td>
</tr>
</tbody>
</table>

Exceptions

Table 273-40 REVERSE Function Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>( r ) is NULL or has 0 length</td>
</tr>
</tbody>
</table>

273.3.23 SUBSTR Function

This function returns \( len \) bytes, starting at \( pos \) from RAW \( r \).

Syntax

\[
\text{UTL_RAW.SUBSTR (}
\begin{array}{ll}
\text{r IN RAW,} \\
\text{pos IN BINARY_INTEGER,} \\
\text{len IN BINARY_INTEGER DEFAULT NULL)} \\
\end{array}
\text{RETURN RAW;}
\]

Pragmas

\text{pragma restrict_references(substr, WNDS, RNDS, WNPS, RNPS);} 

Parameters

Table 273-41 SUBSTR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r )</td>
<td>RAW byte-string from which a portion is extracted</td>
</tr>
<tr>
<td>( pos )</td>
<td>Byte position in ( r ) at which to begin extraction</td>
</tr>
<tr>
<td>( len )</td>
<td>Number of bytes from ( pos ) to extract from ( r ) (optional)</td>
</tr>
</tbody>
</table>

Defaults and Optional Parameters

Table 273-42 SUBSTR Function Optional Parameter

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( len )</td>
<td>Position ( pos ) through to the end of ( r )</td>
</tr>
</tbody>
</table>
Return Values

Table 273-43  SUBSTR Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>portion of r</td>
<td>Beginning at pos for len bytes long</td>
</tr>
<tr>
<td>NULL</td>
<td>r input parameter was NULL</td>
</tr>
</tbody>
</table>

Usage Notes

• If pos is positive, then SUBSTR counts from the beginning of r to find the first byte. If pos is negative, then SUBSTR counts backward from the end of the r. The value pos cannot be 0.
• If len is omitted, then SUBSTR returns all bytes to the end of r. The value len cannot be less than 1.

Exceptions

Table 273-44  SUBSTR Function Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>VALUE_ERROR is returned if:</td>
</tr>
<tr>
<td></td>
<td>• pos = 0 or &gt; length of r</td>
</tr>
<tr>
<td></td>
<td>• len &lt; 1 or &gt; length of r - (pos-1)</td>
</tr>
</tbody>
</table>

273.3.24 TRANSLATE Function

This function translates the bytes in the input RAW r according to the bytes in the translation RAWs from_set and to_set.

If a byte in r has a matching byte in from_set, then it is replaced by the byte in the corresponding position in to_set, or deleted.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored.

Syntax

```sql
UTL_RAW.TRANSLATE ( 
  r IN RAW, 
  from_set IN RAW, 
  to_set IN RAW) 
RETURN RAW;
```

Note:

Be aware that to_set and from_set are reversed in the calling sequence compared to TRANSLITERATE.
Pragmas
pragma restrict_references(translate, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 273-45  TRANSLATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW source byte-string to be translated</td>
</tr>
<tr>
<td>from_set</td>
<td>RAW byte-codes to be translated, if present in r</td>
</tr>
<tr>
<td>to_set</td>
<td>RAW byte-codes to which corresponding from_set bytes are translated</td>
</tr>
</tbody>
</table>

Return Values

Table 273-46  TRANSLATE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Translated byte-string</td>
</tr>
</tbody>
</table>

Usage Notes

- If to_set is shorter than from_set, the extra from_set bytes have no corresponding translation bytes. Bytes from the input RAW that match any such from_set bytes are not translated or included in the result. They are effectively translated to NULL.
- If to_set is longer than from_set, the extra to_set bytes are ignored.
- If a byte value is repeated in from_set, the repeated occurrence is ignored.

Note:

Differences from the TRANSLITERATE Function:

- The from_set parameter comes before the to_set parameter in the calling sequence.
- Bytes from r that appear in from_set but have no corresponding values in to_set are not translated or included in the result.
- The resulting RAW value may be shorter than the input RAW value.

Note that TRANSLATE and TRANSLITERATE only differ in functionality when to_set has fewer bytes than from_set.
Exceptions

Table 273-47 TRANSFORM Function Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>Either:</td>
</tr>
<tr>
<td></td>
<td>- r is NULL or has 0 length</td>
</tr>
<tr>
<td></td>
<td>- from_set is NULL or has 0 length</td>
</tr>
<tr>
<td></td>
<td>- to_set is NULL or has 0 length</td>
</tr>
</tbody>
</table>

273.3.25 TRANSFORM Function

This function converts the bytes in the input RAW r according to the bytes in the transliteration RAWs from_set and to_set.

Successive bytes in r are looked up in the from_set, and, if not found, copied unaltered to the result RAW. If found, then they are replaced in the result RAW by either corresponding bytes in the to_set, or the pad byte when no correspondence exists.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored. The result RAW is always the same length as r.

Syntax

UTL_RAW.TRANSLITERATE (r IN RAW, to_set IN RAW DEFAULT NULL, from_set IN RAW DEFAULT NULL, pad IN RAW DEFAULT NULL) RETURN RAW;

Note:

Be aware that to_set and from_set are reversed in the calling sequence compared to TRANSFORM.

Pragmas

pragma restrict_references(transliterate, WNDS, RNDS, WNPS, RNPS);

Parameters

Table 273-48 TRANSFORM Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>RAW input byte-string to be converted</td>
</tr>
<tr>
<td>to_set</td>
<td>RAW byte-codes to which corresponding from_set bytes are converted (any length)</td>
</tr>
</tbody>
</table>
Table 273-48  (Cont.) TRANSLITERATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from_set</td>
<td>RAW byte-codes to be converted, if presenting r (any length)</td>
</tr>
<tr>
<td>pad</td>
<td>1 byte used when to-set is shorter than the from_set</td>
</tr>
</tbody>
</table>

Defaults and Optional Parameters

Table 273-49  TRANSLITERATE Function Optional Parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_set</td>
<td>To the NULL string and effectively extended with pad to the length of from_set as necessary</td>
</tr>
<tr>
<td>from_set</td>
<td>x'00' through x'fff'</td>
</tr>
<tr>
<td>pad</td>
<td>x'00'</td>
</tr>
</tbody>
</table>

Return Values

Table 273-50  TRANSLITERATE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Converted byte-string.</td>
</tr>
</tbody>
</table>

Usage Notes

- If to_set is shorter than from_set, the extra from_set bytes have no corresponding conversion bytes. Bytes from the input RAW that match any such from_set bytes are converted in the result to the pad byte instead.
- If to_set is longer than from_set, the extra to_set bytes are ignored.
- If a byte value is repeated in from_set, the repeated occurrence is ignored.

Note:

Differences from the TRANSLATE Function:

- The to_set parameter comes before the from_set parameter in the calling sequence.
- Bytes from r that appear in from_set but have no corresponding values in to_set are replaced by pad in the result.
- The resulting RAW value always has the same length as the input RAW value.

Note that TRANSLATE and TRANSLITERATE only differ in functionality when to_set has fewer bytes than from_set.
Exceptions

Table 273-51  TRANSLITERATE Function Exceptions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ERROR</td>
<td>R is NULL or has 0 length</td>
</tr>
</tbody>
</table>

273.3.26 XRANGE Function

This function returns a RAW value containing the succession of one-byte encodings beginning and ending with the specified byte-codes. The specified byte-codes must be single-byte RAW values. If the start_byte value is greater than the end_byte value, then the succession of resulting bytes begins with start_byte, wraps through x'FF' back to x'00', then ends at end_byte.

Syntax

```sql
UTL_RAW.XRANGE (  
    start_byte IN RAW DEFAULT NULL,  
    end_byte   IN RAW DEFAULT NULL)  
RETURN RAW;
```

Pragmas

```sql
pragma restrict_references(xrange, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 273-52  XRANGE Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_byte</td>
<td>Beginning byte-code value of resulting sequence. The default is x'00'.</td>
</tr>
<tr>
<td>end_byte</td>
<td>Ending byte-code value of resulting sequence. The default is x'FF'.</td>
</tr>
</tbody>
</table>

Return Values

Table 273-53  XRANGE Function Return Values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Containing succession of 1-byte hexadecimal encodings</td>
</tr>
</tbody>
</table>
The UTL_RECOMP package recompiles invalid PL/SQL modules, invalid views, Java classes, indextypes and operators in a database, either sequentially or in parallel.

This chapter contains the following topics:

- Overview
- Operational Notes
- Examples
- Summary of UTL_RECOMP Subprograms

274.1 UTL_RECOMP Overview

This script is particularly useful after a major-version upgrade that typically invalidates all PL/SQL and Java objects. Although invalid objects are recompiled automatically on use, it is useful to run this script prior to operation because this will either eliminate or minimize subsequent latencies due to on-demand automatic recompilation at runtime.

Parallel recompilation can exploit multiple CPUs to reduce the time taken to recompile invalid objects. The degree of parallelism is specified by the first argument to `RECOMP_PARALLEL Procedure`.

In general, a parallelism setting of one thread for each available CPU provides a good initial setting. However, please note that the process of recompiling an invalid object writes a significant amount of data to system tables and is fairly I/O intensive. A slow disk system may be a significant bottleneck and limit speedups available from a higher degree of parallelism.

274.2 UTL_RECOMP Operational Notes

UTL_RECOMP has several operational notes.

- This package uses the job queue for parallel recompilation.
- This package must be run using SQL*PLUS.
- You must be connected as SYSDBA to run this script.
- This package expects the following packages to have been created with VALID status:
  - STANDARD (standard.sql)
  - DBMS_STANDARD (dbmsstdx.sql)
  - DBMS_JOB (dbmsjob.sql)
  - DBMS_RANDOM (dbmsrand.sql)
- There should be no other DDL on the database while running entries in this package. Not following this recommendation may lead to deadlocks.
274.3 UTL_RECOMP Examples

These examples show various ways that UTL_RECOMP can recompile objects.

- Recompile all objects sequentially:
  ```sql
  EXECUTE UTL_RECOMP.RECOMP_SERIAL();
  ```

- Recompile objects in schema SCOTT sequentially:
  ```sql
  EXECUTE UTL_RECOMP.RECOMP_SERIAL('SCOTT');
  ```

- Recompile all objects using 4 parallel threads:
  ```sql
  EXECUTE UTL_RECOMP.RECOMP_PARALLEL(4);
  ```

- Recompile objects in schema JOE using the number of threads specified in the parameter JOB_QUEUE_PROCESSES:
  ```sql
  EXECUTE UTL_RECOMP.RECOMP_PARALLEL(NULL, 'JOE');
  ```

274.4 Summary of UTL_RECOMP Subprograms

This table lists the UTL_RECOMP subprograms and briefly describes them.

Table 274-1  UTL_RECOMP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECOMP_PARALLEL</td>
<td>Recompiles invalid objects in a given schema, or all invalid objects in the database, in parallel</td>
</tr>
<tr>
<td>RECOMP_SERIAL</td>
<td>Recompiles invalid objects in a given schema or all invalid objects in the database</td>
</tr>
</tbody>
</table>

274.4.1 RECOMP_PARALLEL Procedure

This procedure uses the information exposed in the DBA_Dependencies view to recompile invalid objects in the database, or in a given schema, in parallel.

Syntax

```sql
UTL_RECOMP.RECOMP_PARALLEL(
  threads IN   PLS_INTEGER DEFAULT NULL,
  schema   IN   VARCHAR2    DEFAULT NULL,
  flags    IN   PLS_INTEGER DEFAULT 0);
```

Parameters

Table 274-2  RECOMP_PARALLEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threads</td>
<td>The number of recompile threads to run in parallel. If NULL, use the value of 'job_queue_processes'.</td>
</tr>
</tbody>
</table>
Table 274-2  (Cont.) RECOMP_PARALLEL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>The schema in which to recompile invalid objects. If NULL, all invalid objects in the database are recompiled.</td>
</tr>
<tr>
<td>flags</td>
<td>Flag values are intended for internal testing and diagnosability only.</td>
</tr>
</tbody>
</table>

Usage Notes

The parallel recompile exploits multiple CPUs to reduce the time taken to recompile invalid objects. However, please note that recompilation writes significant amounts of data to system tables, so the disk system may be a bottleneck and prevent significant speedups.

274.4.2 RECOMP_SERIAL Procedure

This procedure recompiles invalid objects in a given schema or all invalid objects in the database.

Syntax

```sql
UTL_RECOMP.RECOMP_SERIAL(
   schema   IN   VARCHAR2    DEFAULT NULL,
   flags    IN   PLS_INTEGER DEFAULT 0);
```

Parameters

Table 274-3  RECOMP_SERIAL Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>The schema in which to recompile invalid objects. If NULL, all invalid objects in the database are recompiled.</td>
</tr>
<tr>
<td>flags</td>
<td>Flag values are intended for internal testing and diagnosability only.</td>
</tr>
</tbody>
</table>
UTL_REF

The UTL_REF package provides PL/SQL procedures to support reference-based operations. Unlike SQL, UTL_REF procedures enable you to write generic type methods without knowing the object table name.

This chapter contains the following topics:

- Overview
- Security Model
- Types
- Exceptions
- Summary of UTL_REF Subprograms

275.1 UTL_REF Overview

UTL_REF procedures enable you to write generic type methods without knowing the object table name.

Oracle supports user-defined composite type or object type. Any instance of an object type is called an object. An object type can be used as the type of a column or as the type of a table.

In an object table, each row of the table stores an object. You can uniquely identify an object in an object table with an object identifier.

A reference is a persistent pointer to an object, and each reference can contain an object identifier. The reference can be an attribute of an object type, or it can be stored in a column of a table. Given a reference, an object can be retrieved.

275.2 UTL_REF Security Model

The procedural option is needed to use this package. This package must be created under SYS (CONNECT /AS SYSDBA). Operations provided by this package are performed under the current calling user, not under the package owner SYS.

You can use the UTL_REF package from stored PL/SQL procedures/packages on the server, as well as from client/side PL/SQL code.

When invoked from PL/SQL procedures/packages on the server, UTL_REF verifies that the invoker has the appropriate privileges to access the object pointed to by the REF.
Note:

This is in contrast to PL/SQL packages/procedures on the server which operate with definer's privileges, where the package owner must have the appropriate privileges to perform the desired operations.

Thus, if UTL_REF is defined under user SYS, and user A invokes UTL_REF.SELECT to select an object from a reference, then user A (the invoker) requires the privileges to check.

When invoked from client-side PL/SQL code, UTL_REF operates with the privileges of the client session under which the PL/SQL execution is being done.

275.3 UTL_REF Types

An object type is a composite datatype defined by the user or supplied as a library type.

You can create the object type employee_type using the following syntax:

```
CREATE TYPE employee_type AS OBJECT (
    name   VARCHAR2(20),
    id     NUMBER,

    member function GET_ID
        (name VARCHAR2)
    RETURN MEMBER);
```

The object type employee_type is a user-defined type that contains two attributes, name and id, and a member function, GET_ID().

You can create an object table using the following SQL syntax:

```
CREATE TABLE employee_table OF employee_type;
```

275.4 UTL_REF Exceptions

Exceptions can be returned during execution of UTL_REF functions for various reasons.

For example, the following scenarios would result in exceptions:

- The object selected does not exist. This could be because either:
  1. The object has been deleted, or the given reference is dangling (invalid).
  2. The object table was dropped or does not exist.
- The object cannot be modified or locked in a serializable transaction. The object was modified by another transaction after the serializable transaction started.
- You do not have the privilege to select or modify the object. The caller of the UTL_REF subprogram must have the proper privilege on the object that is being selected or modified.
Table 275-1  UTL_REF Exceptions

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errnum == 942</td>
<td>Insufficient privileges.</td>
</tr>
<tr>
<td>errnum == 1031</td>
<td>Insufficient privileges.</td>
</tr>
<tr>
<td>errnum == 8177</td>
<td>Unable to serialize, if in a serializable transaction.</td>
</tr>
<tr>
<td>errnum == 60</td>
<td>Deadlock detected.</td>
</tr>
<tr>
<td>errnum == 1403</td>
<td>No data found (if the REF is NULL, and so on.).</td>
</tr>
</tbody>
</table>

The UTL_REF package does not define any named exceptions. You may define exception handling blocks to catch specific exceptions and to handle them appropriately.

275.5 Summary of UTL_REF Subprograms

This table lists the UTL_REF subprograms and briefly describes them.

Table 275-2  UTL_REF Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE_OBJECT Procedure</td>
<td>Deletes an object given a reference</td>
</tr>
<tr>
<td>LOCK_OBJECT Procedure</td>
<td>Locks an object given a reference</td>
</tr>
<tr>
<td>SELECT_OBJECT Procedure</td>
<td>Selects an object given a reference</td>
</tr>
<tr>
<td>UPDATE_OBJECT Procedure</td>
<td>Updates an object given a reference</td>
</tr>
</tbody>
</table>

275.5.1 DELETE_OBJECT Procedure

This procedure deletes an object given a reference.

The semantic of this subprogram is similar to the following SQL statement:

```
DELETE FROM object_table
WHERE REF(t) = reference;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.DELETE_OBJECT (
    reference IN REF "<typename>");
```

Parameters

Table 275-3  DELETE_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference of the object to delete.</td>
</tr>
</tbody>
</table>
Exceptions

May be raised.

Examples

The following example illustrates usage of the UTL_REF package to implement this scenario: if an employee of a company changes their address, their manager should be notified.

... declarations of Address_t and others...

CREATE OR REPLACE TYPE Person_t (  
    name VARCHAR2(64),  
    gender CHAR(1),  
    address Address_t,  
    MEMBER PROCEDURE setAddress(addr IN Address_t)  
);  

CREATE OR REPLACE TYPE BODY Person_t (  
    MEMBER PROCEDURE setAddress(addr IN Address_t) IS  
    BEGIN  
        address := addr;  
    END;  
);  

CREATE OR REPLACE TYPE Employee_t (  
    Under Person_t: Simulate implementation of inheritance using a REF to Person_t and delegation of setAddress to it.  
    thePerson REF Person_t,  
    empno NUMBER(5),  
    deptREF Department_t,  
    mgrREF Employee_t,  
    reminders StringArray_t,  
    MEMBER PROCEDURE setAddress(addr IN Address_t),  
    MEMBER procedure addReminder(reminder VARCHAR2);  
);  

CREATE TYPE BODY Employee_t (  
    MEMBER PROCEDURE setAddress(addr IN Address_t) IS  
    myMgr Employee_t;  
    meAsPerson Person_t;  
    BEGIN  
        Update the address by delegating the responsibility to thePerson. Lock the Person object from the reference, and also select it:  
        UTL_REF.LOCK_OBJECT(thePerson, meAsPerson);  
        meAsPerson.setAddress(addr);  
        Delegate to thePerson:  
        UTL_REF.UPDATE_OBJECT(thePerson, meAsPerson);  
        if mgr is NOT NULL THEN  
            Give the manager a reminder:
UTL_REF.LOCK_OBJECT(mgr);
UTL_REF.SELECT_OBJECT(mgr, myMgr);
myMgr.addReminder
    ('Update address in the employee directory for' ||
    thePerson.name || ', new address: ' || addr.asString);
UTL_REF.UPDATE_OBJECT(mgr, myMgr);
END IF;
EXCEPTION
    WHEN OTHERS THEN
        errnum := SQLCODE;
        errmsg := SUBSTR(SQLERRM, 1, 200);

275.5.2 LOCK_OBJECT Procedure

This procedure locks an object given a reference. In addition, this procedure lets the program
select the locked object.

The semantic of this subprogram is similar to the following SQL statement:

SELECT VALUE(t)
    INTO object
    FROM object_table t
    WHERE REF(t) = reference
    FOR UPDATE;

Unlike the preceding SQL statement, this subprogram does not require you to specify the
object table name where the object resides. It is not necessary to lock an object before
updating/deleting it.

Syntax

UTL_REF.LOCK_OBJECT {
    reference IN REF "<typename>";
}

UTL_REF.LOCK_OBJECT {
    reference IN REF "<typename>",
    object IN OUT "<typename>";
}

Parameters

Table 275-4  LOCK_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference of the object to lock.</td>
</tr>
</tbody>
</table>
| object    | The PL/SQL variable that stores the locked object. This variable should be of
            the same object type as the locked object. |

Exceptions

May be raised.

275.5.3 SELECT_OBJECT Procedure

This procedure selects an object given its reference. The selected object is retrieved from the
database and its value is put into the PL/SQL variable 'object'.

The semantic of this subprogram is similar to the following SQL statement:
SELECT VALUE(t) INTO object 
FROM object_table t 
WHERE REF(t) = reference;

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

UTL_REF.SELECT_OBJECT ( 
   reference IN REF "<typename>",
   object    IN OUT "<typename>");

Parameters

Table 275-5  SELECT_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference to the object to select or retrieve.</td>
</tr>
<tr>
<td>object</td>
<td>The PL/SQL variable that stores the selected object; this variable should be of the same object type as the referenced object.</td>
</tr>
</tbody>
</table>

Exceptions

May be raised.

275.5.4 UPDATE_OBJECT Procedure

This procedure updates an object given a reference. The referenced object is updated with the value contained in the PL/SQL variable 'object'.

The semantic of this subprogram is similar to the following SQL statement:

UPDATE object_table t 
SET VALUE(t) = object 
WHERE REF(t) = reference;

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

UTL_REF.UPDATE_OBJECT ( 
   reference IN REF "<typename>",
   object    IN "<typename>");

Parameters

Table 275-6  UPDATE_OBJECT Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>Reference of the object to update.</td>
</tr>
<tr>
<td>object</td>
<td>The PL/SQL variable that contains the new value of the object. This variable should be of the same object type as the object to update.</td>
</tr>
</tbody>
</table>
Exceptions

May be raised.
276

UTL_SMTP

The UTL_SMTP package is designed for sending electronic mails (e-mails) over Simple Mail Transfer Protocol (SMTP) as specified by RFC821.

This chapter contains the following topics:

• Overview
• Security Model
• Constants
• Types
• Reply Codes
• UTL_SMTP Operational Notes
• Exceptions
• Rules and Limits
• Examples
• Summary of UTL_SMTP Subprograms

See Also:
Oracle Database Development Guide

276.1 UTL_SMTP Overview

The UTL_SMTP protocol consists of a set of commands for an e-mail client to dispatch e-mails to an SMTP server. The UTL_SMTP package provides interfaces to the SMTP commands.

For many of the commands, the package provides both a procedural and a functional interface. The functional form returns the reply from the server for processing by the client. The procedural form checks the reply and raises an exception if the reply indicates a transient (400-range reply code) or permanent error (500-range reply code). Otherwise, it discards the reply.

Note that the original SMTP protocol communicates using 7-bit ASCII. Using UTL_SMTP, all text data (in other words, those in VARCHAR2) is converted to US7ASCII before it is sent to the server. Some implementations of SMTP servers that support SMTP extension 8BITMIME [RFC1652] support full 8-bit communication between client and server. The body of the DATA command can be transferred in full 8 bits, but the rest of the SMTP command and response must be in 7 bits. When the target SMTP server supports 8BITMIME extension, users of multibyte databases may convert their non-US7ASCII, multibyte VARCHAR2 data to RAW and use the WRITE_RAW_DATA subprogram to send multibyte data using 8-bit MIME encoding.
UTL_SMTP provides for SMTP communication as specified in RFC821, but does not provide an API to format the content of the message according to RFC 822 (for example, setting the subject of an electronic mail). You must format the message appropriately. In addition, UTL_SMTP does not have the functionality to implement an SMTP server for an e-mail clients to send e-mails using SMTP.

### 276.2 UTL_SMTP Security Model

This package is an invoker's rights package. The invoking user must have the `connect` privilege granted in the access control list assigned to the remote network host to which the user must connect.

> **Note:**

For more information on managing fine-grained access, see *Oracle Database Security Guide*

### 276.3 UTL_SMTP Constants

UTL_SMTP defines several constants to use when specifying parameter values. These are shown in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL_SCHEMES</td>
<td>VARCHAR2(256)</td>
<td>'CRAM-MD5 PLAIN LOGIN'</td>
<td>List of all authentication schemes UTL_SMTP supports, in order of their relative security strength. The subset of the schemes in ALL_SCHEMES (namely, PLAIN and LOGIN) in which cleartext passwords are sent over SMTP must be used only in SMTP connections that are secured by Secure Socket Layer / Transport Layer Security (SSL/TLS).</td>
</tr>
<tr>
<td>NON_CLEARTEXT_PASSWORD_SCHEMES</td>
<td>VARCHAR2(256)</td>
<td>'CRAM-MD5'</td>
<td>List of authentication schemes that UTL_SMTP supports and in which no cleartext passwords are sent over SMTP. They can be used in SMTP connections that are not secured by SSL/TLS. Note that these schemes may still be weak when used in an insecure SMTP connection.</td>
</tr>
</tbody>
</table>
276.4 UTL_SMTP Types

UTL_SMTP uses a CONNECTION record type and REPLY_REPLIES record types.

CONNECTION Record Type

This is a PL/SQL record type used to represent an SMTP connection.

Syntax

```plsql
TYPE connection IS RECORD (
    host              VARCHAR2(255),
    port              PLS_INTEGER,
    tx_timeout        PLS_INTEGER,
    private_tcp_con   utl_tcp.connection,
    private_state     PLS_INTEGER);
```

Fields

Table 276-2  CONNECTION Record Type Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>Name of the remote host when connection is established. NULL when no connection is established.</td>
</tr>
<tr>
<td>port</td>
<td>Port number of the remote SMTP server connected. NULL when no connection is established.</td>
</tr>
<tr>
<td>tx_timeout</td>
<td>Time in seconds that the UTL_SMTP package waits before timing out in a read or write operation in this connection. In read operations, this package times out if no data is available for reading immediately. In write operations, this package times out if the output buffer is full and no data is to be sent into the network without being blocked. 0 indicates not to wait at all. NULL indicates to wait forever.</td>
</tr>
<tr>
<td>private_tcp_con</td>
<td>Private, for implementation use only. You should not modify this field.</td>
</tr>
<tr>
<td>private_state</td>
<td>Private, for implementation use only. You should not modify this field.</td>
</tr>
</tbody>
</table>

Usage Notes

The read-only fields in a connection record are used to return information about the SMTP connection after the connection is successfully made with the OPEN_CONNECTION Functions. Changing the values of these fields has no effect on the connection. The fields private_tcp_con and private_state for implementation use only. You should not modify these fields.

REPLY_REPLIES Record Types

These are PL/SQL record types used to represent an SMTP reply line. Each SMTP reply line consists of a reply code followed by a text message. While a single reply line is expected for most SMTP commands, some SMTP commands expect multiple reply lines. For those situations, a PL/SQL table of reply records is used to represent multiple reply lines.
Syntax

```plsql
TYPE reply IS RECORD (    code   PLS_INTEGER,    text   VARCHAR2(508));

TYPE replies IS TABLE OF reply INDEX BY BINARY_INTEGER;
```

Fields

**Table 276-3  REPLY, REPLEIS Record Type Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>3-digit reply code</td>
</tr>
<tr>
<td>text</td>
<td>Text message of the reply</td>
</tr>
</tbody>
</table>

### 276.5 UTL_SMTP Reply Codes

SMTP servers send reply codes that indicate message or server status.

The following is a list of the SMTP reply codes.

**Table 276-4  SMTP Reply Codes**

<table>
<thead>
<tr>
<th>Reply Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>211</td>
<td>System status, or system help reply</td>
</tr>
<tr>
<td>214</td>
<td>Help message [Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]</td>
</tr>
<tr>
<td>220</td>
<td>&lt;domain&gt; Service ready</td>
</tr>
<tr>
<td>221</td>
<td>&lt;domain&gt; Service closing transmission channel</td>
</tr>
<tr>
<td>250</td>
<td>Requested mail action okay, completed</td>
</tr>
<tr>
<td>251</td>
<td>User not local; forwards to &lt;forward-path&gt;</td>
</tr>
<tr>
<td>252</td>
<td>OK, pending messages for node &lt;node&gt; started. Cannot VRFY user (for example, info is not local), but takes message for this user and attempts delivery.</td>
</tr>
<tr>
<td>253</td>
<td>OK, &lt;messages&gt; pending messages for node &lt;node&gt; started</td>
</tr>
<tr>
<td>354</td>
<td>Start mail input; end with &lt;CRLF.CRLF&gt;</td>
</tr>
<tr>
<td>355</td>
<td>Octet-offset is the transaction offset</td>
</tr>
<tr>
<td>421</td>
<td>&lt;domain&gt; Service not available, closing transmission channel (This can be a reply to any command if the service knows it must shut down.)</td>
</tr>
<tr>
<td>450</td>
<td>Requested mail action not taken: mailbox unavailable [for example, mailbox busy]</td>
</tr>
<tr>
<td>451</td>
<td>Requested action terminated: local error in processing</td>
</tr>
<tr>
<td>452</td>
<td>Requested action not taken: insufficient system storage</td>
</tr>
<tr>
<td>453</td>
<td>You have no mail.</td>
</tr>
</tbody>
</table>
### Table 276-4 (Cont.) SMTP Reply Codes

<table>
<thead>
<tr>
<th>Reply Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>454</td>
<td>TLS not available due to temporary reason. Encryption required for requested authentication mechanism.</td>
</tr>
<tr>
<td>458</td>
<td>Unable to queue messages for node <code>&lt;node&gt;</code></td>
</tr>
<tr>
<td>459</td>
<td>Node <code>&lt;node&gt;</code> not allowed: reason</td>
</tr>
<tr>
<td>500</td>
<td>Syntax error, command unrecognized (This may include errors such as command line too long.)</td>
</tr>
<tr>
<td>501</td>
<td>Syntax error in parameters or arguments</td>
</tr>
<tr>
<td>502</td>
<td>Command not implemented</td>
</tr>
<tr>
<td>503</td>
<td>Bad sequence of commands</td>
</tr>
<tr>
<td>504</td>
<td>Command parameter not implemented</td>
</tr>
<tr>
<td>521</td>
<td><code>&lt;Machine&gt;</code> does not accept mail.</td>
</tr>
<tr>
<td>530</td>
<td>Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.</td>
</tr>
<tr>
<td>534</td>
<td>Authentication mechanism is too weak.</td>
</tr>
<tr>
<td>538</td>
<td>Encryption required for requested authentication mechanism.</td>
</tr>
<tr>
<td>550</td>
<td>Requested action not taken: mailbox unavailable [for, mailbox not found, no access]</td>
</tr>
<tr>
<td>551</td>
<td>User not local; please try <code>&lt;forward-path&gt;</code></td>
</tr>
<tr>
<td>552</td>
<td>Requested mail action terminated: exceeded storage allocation</td>
</tr>
<tr>
<td>553</td>
<td>Requested action not taken: mailbox name not allowed [for example, mailbox syntax incorrect]</td>
</tr>
<tr>
<td>554</td>
<td>Transaction failed</td>
</tr>
</tbody>
</table>

### 276.6 UTL_SMTP Operational Notes

An SMTP connection is initiated by a call to OPEN_CONNECTION Functions which returns a SMTP connection.

After a connection is established, the following calls are required to send a mail:

- **HELO Function and Procedure** - identify the domain of the sender
- **MAIL Function and Procedure** - start a mail, specify the sender
- **RCPT Function** - specify the recipient
- **OPEN_DATA Function and Procedure** - start the mail body
- **WRITE_RAW_DATA Procedure** - write the mail body (multiple calls allowed)
- **CLOSE_DATA Function and Procedure** - close the mail body and send the mail

The SMTP connection is closed by calling the **QUIT Function and Procedure**.
Related Topics

- **OPEN_CONNECTION Functions**
  These functions open a connection to an SMTP server.

- **QUIT Function and Procedure**
  This subprogram terminates an SMTP session and disconnects from the server.

### 276.7 UTL_SMTP Exceptions

This table lists the exceptions that can be raised by the interface of the UTL_SMTP package.

The network error is transferred to a reply code of 421- service not available.

**Table 276-5  UTL_SMTP Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_OPERATION</td>
<td>Raised when an invalid operation is made. In other words, calling API other than the WRITE_DATA Procedure, the WRITE_RAW_DATA Procedure or the CLOSE_DATA Function and Procedure after the OPEN_DATA Function and Procedure is called, or calling WRITE_DATA, WRITE_RAW_DATA or CLOSE_DATA without first calling OPEN_DATA.</td>
</tr>
<tr>
<td>TRANSIENT_ERROR</td>
<td>Raised when receiving a reply code in 400 range</td>
</tr>
<tr>
<td>PERMANENT_ERROR</td>
<td>Raised when receiving a reply code in 500 range</td>
</tr>
</tbody>
</table>

### 276.8 UTL_SMTP Rules and Limits

The API imposes no imitation or range-checking. However, you must be aware of the limitations on various elements of SMTP. Sending data that exceed these limits may result in errors returned by the server.

The following table describes the size limitations encountered by the UTL_SMTP subprograms.

**Table 276-6  SMTP Size Limitation**

<table>
<thead>
<tr>
<th>Element</th>
<th>Size Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Maximum total length of a user name is 64 characters</td>
</tr>
<tr>
<td>domain</td>
<td>Maximum total length of a domain name or number is 64 characters</td>
</tr>
<tr>
<td>path</td>
<td>Maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators)</td>
</tr>
<tr>
<td>command line</td>
<td>Maximum total length of a command line including the command word and the &lt;CRLF&gt; is 512 characters</td>
</tr>
<tr>
<td>reply line</td>
<td>Maximum total length of a reply line including the reply code and the &lt;CRLF&gt; is 512 characters</td>
</tr>
<tr>
<td>text line</td>
<td>Maximum total length of a text line including the &lt;CRLF&gt; is 1000 characters (but not counting the leading dot duplicated for transparency)</td>
</tr>
</tbody>
</table>
Table 276-6  (Cont.) SMTP Size Limitation

<table>
<thead>
<tr>
<th>Element</th>
<th>Size Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipients buffer</td>
<td>Maximum total number of recipients that must be buffered is 100 recipients</td>
</tr>
</tbody>
</table>

276.9 UTL_SMTP Examples

This example illustrates how UTL_SMTP is used by an application to send e-mail. The application connects to an SMTP server at port 25 and sends a simple text message.

```
DECLARE
    c UTL_SMTP.CONNECTION;
BEGIN
    PROCEDURE send_header(name IN VARCHAR2, header IN VARCHAR2) AS
    BEGIN
        UTL_SMTP.WRITE_DATA(c, name || ': ' || header || UTL_TCP.CRLF);
    END;

    c := UTL_SMTP.OPEN_CONNECTION('smtp-server.acme.com');
    UTL_SMTP.HELO(c, 'foo.com');
    UTL_SMTP.MAIL(c, 'sender@foo.com');
    UTL_SMTP.RCPT(c, 'recipient@foo.com');
    UTL_SMTP.OPEN_DATA(c);
    send_header('From',    '"Sender" <sender@foo.com>');
    send_header('To',      '"Recipient" <recipient@foo.com>');
    send_header('Subject', 'Hello');
    UTL_SMTP.WRITE_DATA(c, UTL_TCP.CRLF || 'Hello, world!');
    UTL_SMTP.CLOSE_DATA(c);
    UTL_SMTP.QUIT(c);
EXCEPTION
    WHEN utl_smtp.transient_error OR utl_smtp.permanent_error THEN
        BEGIN
            UTL_SMTP.QUIT(c);
        EXCEPTION
            WHEN UTL_SMTP.TRANSIENT_ERROR OR UTL_SMTP.PERMANENT_ERROR THEN
                NULL; -- When the SMTP server is down or unavailable, we don't have
                -- a connection to the server. The QUIT call raises an
                -- exception that we can ignore.
        END;
        raise_application_error(-20000,
            'Failed to send mail due to the following error: ' || sqlerrm);
    END;
END;
```

276.10 Summary of UTL_SMTP Subprograms

This table lists the UTL_SMTP subprograms and briefly describes them.

Table 276-7  UTL_SMTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH Function and Procedure</td>
<td>Sends the AUTH command to authenticate to the SMTP server</td>
</tr>
</tbody>
</table>
Table 276-7 (Cont.) UTL_SMTP Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE_CONNECTION Procedure</td>
<td>Closes the SMTP connection, causing the current SMTP operation to terminate</td>
</tr>
<tr>
<td>CLOSE_DATA Function and Procedure</td>
<td>Closes the data session</td>
</tr>
<tr>
<td>COMMAND Function and Procedure</td>
<td>Performs a generic SMTP command</td>
</tr>
<tr>
<td>COMMAND_REPLIES Function</td>
<td>Performs a generic SMTP command and retrieves multiple reply lines</td>
</tr>
<tr>
<td>DATA Function and Procedure</td>
<td>Sends the e-mail body</td>
</tr>
<tr>
<td>EHLO Function and Procedure</td>
<td>Performs the initial handshake with SMTP server using the EHLO command</td>
</tr>
<tr>
<td>HELO Function and Procedure</td>
<td>Performs the initial handshake with SMTP server using the HELO command</td>
</tr>
<tr>
<td>HELP Function</td>
<td>Sends HELP command</td>
</tr>
<tr>
<td>MAIL Function and Procedure</td>
<td>Initiates an e-mail transaction with the server, the destination is a mailbox</td>
</tr>
<tr>
<td>NOOP Function and Procedure</td>
<td>NULL command</td>
</tr>
<tr>
<td>OPEN_CONNECTION Functions</td>
<td>Opens a connection to an SMTP server</td>
</tr>
<tr>
<td>OPEN_DATA Function and Procedure</td>
<td>Sends the DATA command</td>
</tr>
<tr>
<td>QUIT Function and Procedure</td>
<td>Terminates an SMTP session and disconnects from the server</td>
</tr>
<tr>
<td>RCPT Function</td>
<td>Specifies the recipient of an e-mail message</td>
</tr>
<tr>
<td>RSET Function and Procedure</td>
<td>Terminates the current e-mail transaction</td>
</tr>
<tr>
<td>STARTTLS Function and Procedure</td>
<td>Sends STARTTLS command to secure the SMTP connection using SSL/TLS</td>
</tr>
<tr>
<td>VRFY Function</td>
<td>Verifies the validity of a destination e-mail address</td>
</tr>
<tr>
<td>WRITE_DATA Procedure</td>
<td>Writes a portion of the e-mail message</td>
</tr>
<tr>
<td>WRITE_RAW_DATA Procedure</td>
<td>Writes a portion of the e-mail message withRAW data</td>
</tr>
</tbody>
</table>

276.10.1 AUTH Function and Procedure

This subprogram sends the AUTH command to authenticate to the SMTP server. The UTL_SMTP package goes through the user's choices of authentication schemes, skips any that is not supported by the SMTP server and uses the first supported.

To determine the schemes the SMTP server supports from its EHLO reply, the user must call the EHLO Function and Procedure. Otherwise, UTL_SMTP uses the first scheme in the list.
Syntax

UTL_SMTP.AUTH (  
    c IN OUT NOCOPY connection,  
    username   IN            VARCHAR2,  
    password   IN            VARCHAR2,  
    schemes    IN            VARCHAR2 DEFAULT NON_CLEARTEXT_PASSWORD_SCHEMES)  
RETURN reply;

UTL_SMTP.AUTH (  
    c IN OUT NOCOPY connection,  
    username   IN            VARCHAR2,  
    password   IN            VARCHAR2,  
    schemes    IN            VARCHAR2 DEFAULT NON_CLEARTEXT_PASSWORD_SCHEMES);

Parameters

Table 276-8  AUTH Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>username</td>
<td>Username</td>
</tr>
<tr>
<td>password</td>
<td>Password</td>
</tr>
<tr>
<td>schemes</td>
<td>Space-separated list of authentication schemes. UTL_SMTP is allowed to use in the preferred order. See the ALL_SCHEMES and NON_CLEARTEXT_PASSWORD_SCHEMES constants for suggestions.</td>
</tr>
</tbody>
</table>

Return Values

Table 276-9  AUTH Function and Procedure Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_REP IES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

- Currently only PLAIN, LOGIN and CRAM-MD5 authentication schemes are supported by UTL_SMTP.
- Since the SMTP server may change the authentication schemes it supports after the SMTP connection is secured by SSL/TLS after the STARTTLS command (for example, adding PLAIN and LOGIN), the caller must call the EHLO Function and Procedure again for UTL_SMTP to update the list after the STARTTLS Function and Procedure is called.

Examples

DECLARE  
c utl_smtp.connection;
BEGIN  
c := utl_smtp.open_connection(  
    host => 'smtp.example.com',  
    port => 25,  
);
wallet_path => 'file:/oracle/wallets/smtp_wallet',
wallet_password => 'password',
secure_connection_before_smtp => FALSE);
UTL_SMTP.STARTTLS(c);
UTL_SMTP.AUTH(
  c => c,
  username => 'scott',
  password => 'password'
  schemes => ucl_smtp.all_schemes);
END;

276.10.2 CLOSE_CONNECTION Procedure

This procedure closes the SMTP connection, causing the current SMTP operation to terminate. Use this procedure only to cancel an e-mail in the middle of the data session.

To end the SMTP connection properly, use the QUIT Function and Procedure.

Syntax

UTL_SMTP.CLOSE_CONNECTION (c IN OUT NOCOPY connection);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
</tbody>
</table>

276.10.3 CLOSE_DATA Function and Procedure

This subprogram ends the e-mail message by sending the sequence <CR><LF>.<CR><LF> (a single period at the beginning of a line).

Syntax

UTL_SMTP.CLOSE_DATA (c IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.CLOSE_DATA (c IN OUT NOCOPY connection);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
</tbody>
</table>
Return Values

Table 276-12 CLOSE_DATA Function and Procedure Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_REPLES Record Types in UTIl_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

The calls to OPEN_DATA, WRITE_DATA, WRITE_RAW_DATA and CLOSE_DATA must be made in the right order. A program calls OPEN_DATA to send the DATA command to the SMTP server. After that, it can call WRITE_DATA or WRITE_RAW_DATA repeatedly to send the actual data. The data is terminated by calling CLOSE_DATA. After OPEN_DATA is called, the only subprograms that can be called are WRITE_DATA, WRITE_RAW_DATA, or CLOSE_DATA. A call to other subprograms results in an INVALID_OPERATION exception being raised.

CLOSE_DATA must be called only after OPEN_CONNECTION, HELO or EHLO, MAIL, and RCPT have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

Note that there is no function form of WRITE_DATA because the SMTP server does not respond until the data-terminator is sent during the call to CLOSE_DATA.

276.10.4 COMMAND Function and Procedure

This subprogram performs a generic SMTP command.

Syntax

UTL_SMTP.COMMAND (c IN OUT NOCOPY connection, cmd IN VARCHAR2, arg IN VARCHAR2 DEFAULT NULL) RETURN reply;

UTL_SMTP.COMMAND (c IN OUT NOCOPY connection, cmd IN VARCHAR2, arg IN VARCHAR2 DEFAULT NULL);

Parameters

Table 276-13 COMMAND Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>cmd</td>
<td>SMTP command to send to the server</td>
</tr>
<tr>
<td>arg</td>
<td>Optional argument to the SMTP argument. A space is inserted between cmd and arg.</td>
</tr>
</tbody>
</table>
Return Values

Table 276-14  COMMAND Function and Procedure Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPL_Y_REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

This function is used to invoke generic SMTP commands. Use COMMAND if only a single reply line is expected. Use COMMAND_REPLIES if multiple reply lines are expected.

For COMMAND, if multiple reply lines are returned from the SMTP server, it returns the last reply line only.

276.10.5 COMMAND_REPLIES Function

This function performs a generic SMTP command and retrieves multiple reply lines.

Syntax

```sql
UTL_SMTP.COMMAND_REPLIES (  
c IN OUT NOCOPY connection,  
cmd IN VARCHAR2,  
arg IN VARCHAR2 DEFAULT NULL)  
RETURN replies;
```

Parameters

Table 276-15  COMMAND_REPLIES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>cmd</td>
<td>SMTP command to send to the server</td>
</tr>
<tr>
<td>arg</td>
<td>Optional argument to the SMTP argument. A space is inserted between cmd and arg.</td>
</tr>
</tbody>
</table>

Return Values

Table 276-16  COMMAND_REPLIES Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replies</td>
<td>Reply of the command (see REPL_Y_REPLIES Record Types in UTL_SMTP Types)</td>
</tr>
</tbody>
</table>

Usage Notes

This function is used to invoke generic SMTP commands. Use COMMAND if only a single reply line is expected. Use COMMAND_REPLIES if multiple reply lines are expected.
For **COMMAND**, if multiple reply lines are returned from the SMTP server, it returns the last reply line only.

### 276.10.6 DATA Function and Procedure

This subprogram specifies the body of an e-mail message.

**Syntax**

```sql
UTL_SMTP.DATA (  
    c     IN OUT NOCOPY connection  
    body  IN  VARCHAR2 CHARACTER SET ANY_CS)  
RETURN reply;
```

```sql
UTL_SMTP.DATA (  
    c     IN OUT NOCOPY connection  
    body  IN VARCHAR2 CHARACTER SET ANY_CS);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP Connection</td>
</tr>
<tr>
<td>body</td>
<td>Text of the message to be sent, including headers, in [RFC822] format</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

**Usage Notes**

The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The DATA routine terminates the message with a `<CR><LF>.<CR><LF>` sequence (a single period at the beginning of a line), as required by RFC821. It also translates any sequence of `<CR><LF>.<CR><LF>` (single period) in body to `<CR><LF>..<CR><LF>` (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

The **DATA** subprogram must be called only after **OPEN_CONNECTION**, **HELO** or **EHLO**, **MAIL** and **RCPT** have been called. The connection to the SMTP server must be open, and a mail transaction must be active when this routine is called.

The expected response from the server is a message beginning with status code 250. The 354 response received from the initial DATA command is not returned to the caller.
276.10.7 EHLO Function and Procedure

This subprogram performs the initial handshake with SMTP server using the EHLO command.

Syntax

```plsql
UTL_SMTP.EHLO (  
  c       IN OUT NOCOPY connection,  
  domain  IN)  
RETURN replies;
```

```plsql
UTL_SMTP.EHLO (  
  c       IN OUT NOCOPY connection,  
  domain  IN);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>domain</td>
<td>Domain name of the local (sending) host. Used for identification purposes.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replies</td>
<td>Reply of the command (see REPL_ REPLIES Record Types in UTI_SMTP Types).</td>
</tr>
</tbody>
</table>

Usage Notes

The EHLO interface is identical to HELO except that it allows the server to return more descriptive information about its configuration. [RFC1869] specifies the format of the information returned, which the PL/SQL application can retrieve using the functional form of this call. For compatibility with HELO, each line of text returned by the server begins with status code 250.

Related Functions

HELO Function and Procedure
276.10.8 HELO Function and Procedure

This subprogram performs the initial handshake with SMTP server using the `HELO` command.

**Syntax**

```sql
UTL_SMTP.HELO (  
c       IN OUT NOCOPY   connection,
domain  IN              VARCHAR2)  
RETURN reply;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>domain</td>
<td>Domain name of the local (sending) host. Used for identification purposes.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see <code>REPLY_REPLIES</code> Record Types in <code>UTL_SMTP Types</code>). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

**Usage Notes**

RFC 821 specifies that the client must identify itself to the server after connecting. This routine performs that identification. The connection must have been opened through a call to `OPEN_CONNECTION` Functions before calling this routine.

The expected response from the server is a message beginning with status code 250.

**Related Functions**

EHLO Function and Procedure

276.10.9 HELP Function

This function sends the `HELP` command.

**Syntax**

```sql
UTL_SMTP.HELP (  
c       IN OUT NOCOPY   connection,
```
command IN VARCHAR2 DEFAULT NULL)
RETURN replies;

Parameters

Table 276-23 HELP Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>command</td>
<td>Command to get the help message</td>
</tr>
</tbody>
</table>

Return Values

Table 276-24 HELP Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replies</td>
<td>Reply of the command (see REPL_Y_REPLIES Record Types in UTL_SMTP Types)</td>
</tr>
</tbody>
</table>

276.10.10 MAIL Function and Procedure

This subprogram initiate a mail transaction with the server. The destination is a mailbox.

Syntax

UTL_SMTP.MAIL (  
  c IN OUT NOCOPY connection,  
  sender IN VARCHAR2,  
  parameters IN VARCHAR2 DEFAULT NULL)  
RETURN reply;

UTL_SMTP.MAIL (  
  c IN OUT NOCOPY connection,  
  sender IN VARCHAR2,  
  parameters IN VARCHAR2 DEFAULT NULL);

Parameters

Table 276-25 MAIL Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>sender</td>
<td>E-mail address of the user sending the message.</td>
</tr>
<tr>
<td>parameters</td>
<td>Additional parameters to mail command as defined in Section 6 of [RFC1869]. It must follow the format of &quot;XXX=XXX (XXX=XXX ....)&quot;.</td>
</tr>
</tbody>
</table>
Return Values

Table 276-26  MAIL Function and Procedure Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

This command does not send the message; it simply begins its preparation. It must be followed by calls to RCPT and DATA to complete the transaction. The connection to the SMTP server must be open and a HELO or EHLO command must have already been sent.

The expected response from the server is a message beginning with status code 250.

276.10.11 NOOP Function and Procedure

This subprogram issues the NULL command.

Syntax

UTL_SMTP.NOOP (c IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.NOOP (c IN OUT NOCOPY connection);

Parameter

Table 276-27  NOOP Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
</tbody>
</table>

Return Values

Table 276-28  NOOP Function and Procedure Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

- This command has no effect except to elicit a successful reply from the server. It can be issued at any time after the connection to the server has been established with OPEN_CONNECTION. The NOOP command can be used to verify that the server is still connected and is listening properly.
• This command replies with a single line beginning with status code 250.

276.10.12 OPEN_CONNECTION Functions

These functions open a connection to an SMTP server.

Syntax

```sql
UTL_SMTP.OPEN_CONNECTION (
    host                            IN  VARCHAR2,
    port                            IN  PLS_INTEGER DEFAULT 25,
    c                               OUT connection,
    tx_timeout                      IN  PLS_INTEGER DEFAULT NULL,
    wallet_path                     IN  VARCHAR2 DEFAULT NULL,
    wallet_password                 IN  VARCHAR2 DEFAULT NULL,
    secure_connection_before_smtp   IN  BOOLEAN DEFAULT FALSE,
    secure_host                     IN  VARCHAR2 DEFAULT NULL)
RETURN reply;
```

```sql
UTL_SMTP.OPEN_CONNECTION (
    host                           IN  VARCHAR2,
    port                           IN  PLS_INTEGER DEFAULT 25,
    tx_timeout                     IN  PLS_INTEGER DEFAULT NULL,
    wallet_path                    IN  VARCHAR2 DEFAULT NULL,
    wallet_password                IN  VARCHAR2 DEFAULT NULL,
    secure_connection_before_smtp  IN  BOOLEAN DEFAULT FALSE,
    secure_host                    IN  VARCHAR2 DEFAULT NULL)
RETURN connection;
```

Parameters

Table 276-29  OPEN_CONNECTION Functions Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>Name of the SMTP server host</td>
</tr>
<tr>
<td>port</td>
<td>Port number on which SMTP server is listening (usually 25)</td>
</tr>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>tx_timeout</td>
<td>Time in seconds that the UTL_SMTP package waits before timing out in a read or write operation for this connection. In read operations, this package times out if no data is available for reading immediately. In write operations, this package times out if the output buffer is full and no data is to be sent into the network without being blocked. 0 indicates not to wait at all. NULL indicates to wait forever.</td>
</tr>
<tr>
<td>wallet_path</td>
<td>Directory path that contains the Oracle wallet for SSL/TLS. The format is file:&lt;directory-path&gt;</td>
</tr>
<tr>
<td>wallet_password</td>
<td>Password to open the wallet. When the wallet is auto-login enabled, the password can be set to NULL.</td>
</tr>
<tr>
<td>secure_connection_before_smtp</td>
<td>If TRUE, a secure connection with SSL/TLS is made before SMTP communication. If FALSE, no connection is made.</td>
</tr>
<tr>
<td>secure_host</td>
<td>The host name to be matched against the common name (CN) of the SMTP server’s certificate when a secure connection is used. It can also be a domain name like &quot;*.example.com&quot;. If NULL, the SMTP host name to connect to will be used.</td>
</tr>
</tbody>
</table>
Return Values

Table 276-30  OPEN_CONNECTION Functions Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

- The expected response from the server is a message beginning with status code 220.
- The version of OPEN_CONNECTION that returns UTL_SMTP.CONNECTION record checks the reply code returned by an SMTP server when the connection is first established. It raises an exception when the reply indicates an error. Otherwise, it discards the reply. If you want to examine the reply, invoke the version of OPEN_CONNECTION that returns REPLY.
- tx_timeout is intended to govern both the read operations and the write operations. However, an implementation restriction prevents tx_timeout from governing write operations in the current release.

Examples

DECLARE
  c utl_smtp.connection;
BEGIN
  c := UTL_SMTP.OPEN_CONNECTION(
      host => 'smtp.example.com',
      port => 465,
      wallet_path => 'file:/oracle/wallets/smtp_wallet',
      wallet_password => 'password',
      secure_connection_before_smtp => TRUE);
END;

276.10.13 OPEN_DATA Function and Procedure

This subprogram sends the DATA command after which you can use WRITE_DATA and WRITE_RAW_DATA to write a portion of the e-mail message.

Syntax

UTL_SMTP.OPEN_DATA (  
  c IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.OPEN_DATA (  
  c IN OUT NOCOPY connection);}
Parameters

Table 276-31 OPEN_DATA Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>data</td>
<td>Portion of the text of the message to be sent, including headers, in RFC822 format.</td>
</tr>
</tbody>
</table>

Return Values

Table 276-32 OPEN_DATA Function and Procedure Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPL_ REPLY_ REPIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

• The calls to OPEN_DATA, WRITE_DATA, WRITE_RAW_DATA and CLOSE_DATA must be made in the right order. A program calls OPEN_DATA to send the DATA command to the SMTP server. After that, it can call WRITE_DATA or WRITE_RAW_DATA repeatedly to send the actual data. The data is terminated by calling CLOSE_DATA. After OPEN_DATA is called, the only subprograms that can be called are WRITE_DATA, WRITE_RAW_DATA, or CLOSE_DATA. A call to other subprograms results in an INVALID_OPERATION exception being raised.

• OPEN_DATA must be called only after OPEN_CONNECTION, HELO or EHLO, MAIL, and RCPT have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

276.10.14 QUIT Function and Procedure

This subprogram terminates an SMTP session and disconnects from the server.

Syntax

```sql
UTL_SMTP.QUIT (c IN OUT NOCOPY connection)
RETURN reply;
```

Parameter

Table 276-33 QUIT Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
</tbody>
</table>
Return Values

Table 276-34 QUIT Function and Procedure Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_ REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

The QUIT command informs the SMTP server of the client's intent to terminate the session. It then closes the connection established by OPEN_CONNECTION which must have been called before executing this command. If a mail transaction is in progress when QUIT is issued, it is canceled in the same manner as RSET.

The function form of this command returns a single line beginning with the status code 221 on successful termination. In all cases, the connection to the SMTP server is closed. The fields REMOTE_HOST and REMOTE_PORT of c are reset.

Related Functions

RSET Function and Procedure

276.10.15 RCPT Function

This subprogram specifies the recipient of an e-mail message.

Syntax

```sql
UTL_SMTP.RCPT (c           IN OUT NOCOPY     connection,
recipient   IN                VARCHAR2,
parameters  IN                VARCHAR2 DEFAULT NULL)
RETURN reply;
```

Table 276-35 RCPT Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>recipient</td>
<td>E-mail address of the user to which the message is being sent</td>
</tr>
<tr>
<td>parameters</td>
<td>Additional parameters to RCPT command as defined in Section 6 of [RFC1869]. It must follow the format of &quot;XXX=XXX (XXX=XXX ....)&quot;.</td>
</tr>
</tbody>
</table>
Return Values

Table 276-36  RCPT Function and Procedure Function Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

To send a message to multiple recipients, call this routine multiple times. Each invocation schedules delivery to a single e-mail address. The message transaction must have been begun by a prior call to MAIL, and the connection to the mail server must have been opened and initialized by prior calls to OPEN_CONNECTION and HELO or EHLO respectively.

The expected response from the server is a message beginning with status code 250 or 251.

276.10.16 RSET Function and Procedure

This subprogram terminates the current mail transaction.

Syntax

```
UTL_SMTP.RSET (c  IN OUT NOCOPY connection)
RETURN reply;
```

Parameters

Table 276-37  RSET Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
</tbody>
</table>

Return Values

Table 276-38  RSET Function and Procedure Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see REPLY_REPLIES Record Types in UTL_SMTP Types). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>
Usage Notes

- This command allows the client to cancel an e-mail message it was in the process of composing. No mail is sent. The client can call RSET at any time after the connection to the SMTP server has been opened by means of OPEN_CONNECTION until DATA or OPEN_DATA is called. Once the e-mail data has been sent, it is too late to prevent the e-mail from being sent.
- The server responds to RSET with a message beginning with status code 250.

Related Functions
QUIT Function and Procedure

276.10.17 STARTTLS Function and Procedure

This subprogram sends the STARTTLS command to secure the SMTP connection using SSL/TLS.

SSL/TLS requires an Oracle wallet which must be specified when the connection was opened by the OPEN_CONNECTION Functions.

Syntax

UTL_SMTP.STARTTLS (c IN OUT NOCOPY connection,
                        secure_host IN VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.STARTTLS (c IN OUT NOCOPY connection,
                        secure_host IN VARCHAR2 DEFAULT NULL);

Parameters

Table 276-39 STARTTLS Function and Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>secure_host</td>
<td>The host name to be matched against the common name (CN) of the SMTP server's certificate. It can also be a domain name like &quot;*.example.com&quot;. If NULL, the SMTP host name to connect to will be used.</td>
</tr>
</tbody>
</table>

Return Values

Table 276-40 STARTTLS Function and Procedure Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>SMTP reply</td>
</tr>
</tbody>
</table>
Usage Notes

The **STARTTLS** command must only be issued on an unencrypted connection and when the SMTP server indicates the support of the command in the reply of the **EHLO** command. The wallet to be used for encryption must have been specified when the initial SMTP connection was opened by the **OPEN_CONNECTION** function.

Examples

```sql
DECLARE
  c utl_smtp.connection;
BEGIN
  c := utl_smtp.open_connection(
    host => 'smtp.example.com',
    port => 25,
    wallet_path => 'file:/oracle/wallets/smtp_wallet',
    wallet_password => 'password',
    secure_connection_before_smtp => FALSE);
  utl_smtp.starttls(c);
END;
```

276.10.18 VRFY Function

This function verifies the validity of a destination e-mail address.

Syntax

```sql
UTL_SMTP.VRFY (
    c          IN OUT NOCOPY connection,
    recipient  IN VARCHAR2)
RETURN reply;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>recipient</td>
<td>E-mail address to be verified</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reply</td>
<td>Reply of the command (see <strong>REPLY_REPLEIES Record Types in UTL_SMTP Types</strong>). In cases where there are multiple replies, the last reply is returned.</td>
</tr>
</tbody>
</table>

Usage Notes

The server attempts to resolve the destination address **recipient**. If successful, it returns the recipient's full name and fully qualified mailbox path. The connection to the
server must have already been established by means of \texttt{OPEN\_CONNECTION} and \texttt{HELO} or \texttt{EHLO} before making this request.

Successful verification returns one or more lines beginning with status code 250 or 251.

276.10.19 WRITE\_DATA Procedure

This procedure writes a portion of the e-mail message. A repeat call to \texttt{WRITE\_DATA} appends data to the e-mail message.

Syntax

\begin{verbatim}
UTL\_SMTP.WRITE\_DATA (  
c     IN OUT NOCOPY connection,  
data  IN VARCHAR2 CHARACTER SET ANY\_CS);
\end{verbatim}

Parameters

\begin{table}[ht]
\centering
\caption{WRITE\_DATA Procedure Parameters}
\begin{tabular}{ll}
\hline
Parameter & Description \\
\hline
c & SMTP connection \\
data & Portion of the text of the message to be sent, including headers, in \texttt{[RFC822]} format \\
\hline
\end{tabular}
\end{table}

Usage Notes

- The calls to the \texttt{OPEN\_DATA Function and Procedure}, \texttt{WRITE\_DATA Procedure}, \texttt{WRITE\_RAW\_DATA Procedure} and \texttt{CLOSE\_DATA Function and Procedure} must be made in the correct order. A program calls \texttt{OPEN\_DATA} to send the \texttt{DATA} command to the SMTP server. After that, it can call \texttt{WRITE\_DATA} or \texttt{WRITE\_RAW\_DATA} repeatedly to send the actual data. The data is terminated by calling \texttt{CLOSE\_DATA}. After \texttt{OPEN\_DATA} is called, the only subprograms that can be called are \texttt{WRITE\_DATA}, \texttt{WRITE\_RAW\_DATA}, or \texttt{CLOSE\_DATA}. A call to other subprograms results in an \texttt{INVALID\_OPERATION} exception being raised.

- The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The \texttt{DATA} routine terminates the message with a \texttt{<CR><LF>.<CR><LF>} sequence (a single period at the beginning of a line), as required by RFC821. It also translates any sequence of \texttt{<CR><LF>.<CR><LF>} (single period) in the body to \texttt{<CR><LF>.<CR><LF>} (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

- The \texttt{OPEN\_DATA Function and Procedure}, \texttt{WRITE\_DATA Procedure}, \texttt{WRITE\_RAW\_DATA Procedure} and \texttt{CLOSE\_DATA Function and Procedure} must be called only after \texttt{OPEN\_CONNECTION Functions}, \texttt{HELO Function and Procedure}, or \texttt{EHLO Function and Procedure}, \texttt{MAIL Function and Procedure}, and \texttt{RCPT Function} have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

- Note that there is no function form of the \texttt{WRITE\_DATA Procedure} because the SMTP server does not respond until the data-terminator is sent during the call to \texttt{CLOSE\_DATA Function and Procedure}.

- Text (:\texttt{VARCHAR2}) data sent using \texttt{WRITE\_DATA} is converted to US7ASCII before it is sent. If the text contains multibyte characters, each multibyte character in the text that cannot be converted to US7ASCII is replaced by a “?” character. If 8BITMIME extension is
negotiated with the SMTP server using the **EHLO** subprogram, multibyte **VARCHAR2** data can be sent by first converting the text to **RAW** using the **UTL_RAW** package, and then sending the **RAW** data using **WRITE_RAW_DATA**.

### 276.10.20 WRITE_RAW_DATA Procedure

This procedure writes a portion of the e-mail message. A repeat call to **WRITE_RAW_DATA** appends data to the e-mail message.

#### Syntax

```sql
UTL_SMTP.WRITE_RAW_DATA (
  c     IN OUT NOCOPY connection
  data  IN RAW);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>SMTP connection</td>
</tr>
<tr>
<td>data</td>
<td>Portion of the text of the message to be sent, including headers, in [RFC822] format</td>
</tr>
</tbody>
</table>

#### Usage Notes

- The calls to the **OPEN_DATA Function and Procedure**, **WRITE_DATA Procedure**, **WRITE_RAW_DATA Procedure** and **CLOSE_DATA Function and Procedure** must be made in the correct order. A program calls **OPEN_DATA** to send the **DATA** command to the SMTP server. After that, it can call **WRITE_DATA** or **WRITE_RAW_DATA** repeatedly to send the actual data. The data is terminated by calling **CLOSE_DATA**. After **OPEN_DATA** is called, the only subprograms that can be called are **WRITE_DATA**, **WRITE_RAW_DATA**, or **CLOSE_DATA**. A call to other subprograms results in an **INVALID_OPERATION** exception being raised.

- The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The **DATA** routine terminates the message with a `<CR><LF>.<CR><LF>` sequence (a single period at the beginning of a line), as required by RFC821. It also translates any sequence of `<CR><LF>.<CR><LF>` (single period) in the body to `<CR><LF><CR><LF>` (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

- The **OPEN_DATA Function and Procedure**, **WRITE_DATA Procedure**, **WRITE_RAW_DATA Procedure** and **CLOSE_DATA Function and Procedure** must be called only after **OPEN_CONNECTION Functions**, **HELO Function and Procedure**, or **EHLO Function and Procedure**, **MAIL Function and Procedure**, and **RCPT Function** have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

- Note that there is no function form of the **WRITE_DATA Procedure** because the SMTP server does not respond until the data-terminator is sent during the call to **CLOSE_DATA Function and Procedure**.
The **UTL_SPADV** package, one of a set of Oracle Streams packages, provides subprograms to collect and analyze statistics for the Oracle Streams components in a distributed database environment. This package uses the Oracle Streams Performance Advisor to gather statistics.

This chapter contains the following topic:

- Overview
- Security Model
- Operational Notes
- Summary of UTL_SPADV Subprograms

### 277.1 UTL_SPADV Overview

This package enables you to collect and analyze statistics about the performance of Oracle Streams components. You can either collect statistics on demand or you can create a monitoring job that continually monitors Oracle Streams performance.

When this package is used on an Oracle Database 11g Release 2 (11.2) database, it can monitor Oracle Database 10g Release 2 (10.2) and later databases. It cannot monitor databases before release 10.2.

### See Also:

*Oracle Streams Concepts and Administration* for more information about this package and the Oracle Streams Performance Advisor

### 277.2 DBMS_COMPARISON Security Model

Security on this package can be controlled by either granting `EXECUTE` on this package to selected users or roles, or by granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.
To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

See Also:

Oracle Streams Replication Administrator’s Guide for information about configuring an Oracle Streams administrator

277.3 UTL_SPADV Operational Notes

To use this package, you must connect to an Oracle database as an Oracle Streams administrator and run the utlspadv.sql script in the rdbms/admin directory in ORACLE_HOME.

The utlspadv.sql script creates the following tables:

- STREAMS$ _PA_COMPONENT Table
- STREAMS$ _PA_COMPONENT_LINK Table
- STREAMS$ _PA_COMPONENT_PROP Table
- STREAMS$ _PA_COMPONENT_STAT Table
- STREAMS$ _PA_CONTROL Table
- STREAMS$ _PA_DATABASE Table
- STREAMS$ _PA_DATABASE_PROP Table
- STREAMS$ _PA_MONITORING Table
- STREAMS$ _PA_PATH_BOTTLENECK Table
- STREAMS$ _PA_PATH_STAT Table
- STREAMS$ _PA_SHOW_COMP_STAT Table
- STREAMS$ _PA_SHOW_PATH_STAT Table

The Oracle Streams Performance Advisor populates these tables when it is run.

See Also:

Oracle Streams Concepts and Administration

STREAMS$ _PA_COMPONENT Table

The STREAMS$ _PA_COMPONENT table displays information about the Oracle Streams components at each database.
Table 277-1  STREAMS$ PA_COMPONENT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Identification number assigned to the component by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>COMPONENT_NAME</td>
<td>VARCHAR2(194)</td>
<td></td>
<td>Name of the component</td>
</tr>
<tr>
<td>COMPONENT_DB</td>
<td>VARCHAR2(128)</td>
<td></td>
<td>Name of the database that contains the component</td>
</tr>
</tbody>
</table>
| COMPONENT_TYPE     | VARCHAR2(20)      |      | Type of the component
The following types are possible:
  - CAPTURE for a capture process
  - PROPAGATION SENDER for a propagation sender
  - PROPAGATION RECEIVER for a propagation receiver
  - APPLY for an apply process
  - QUEUE for a queue
| COMPONENT_CHANGED_TIME | DATE               |      | Time when the component was last changed                                   |

STREAMS$ PA_COMPONENT_LINK Table

The STREAMS$ PA_COMPONENT_LINK table displays information about how information flows between Oracle Streams components.

Table 277-2  STREAMS$ PA_COMPONENT_LINK Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Identification number assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>PATH_KEY</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>Unique key assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
</tbody>
</table>
| SOURCE_COMPONENT_ID| NUMBER            | NOT NULL | Source component ID for the path
The path starts with this component. |
| DESTINATION_COMPONENT_ID | NUMBER            | NOT NULL | Destination component ID for the path
The path ends with this component. |
| POSITION           | NUMBER            |      | Position of the component in the path                                       |
STREAMS$_PA_COMPONENT_PROP Table

The STREAMS$_PA_COMPONENT_PROP table displays information about capture processes and apply processes necessary for analysis by the Streams Performance Advisor.

Table 277-3  STREAMS$_PA_COMPONENT_PROP Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Identification number assigned to the component by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>PROP_NAME</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>Property name</td>
</tr>
</tbody>
</table>

For a capture process, the component properties include the following:

- **SOURCE_DATABASE** - The source database for the changes captured by the capture process
- **PARALLELISM** - The setting for the parallelism capture process parameter
- **OPTIMIZATION_MODE** - Indicates whether the capture process uses combined capture and apply (greater than zero) or does not use combined capture and apply (0)

For an apply process, the component properties include the following:

- **SOURCE_DATABASE** - The source database for the messages applied by the apply process
- **PARALLELISM** - The setting for the parallelism apply process parameter
- **APPLY_CAPTURED** - Indicates whether the apply process applies captured messages (YES) persistent messages (NO)
- **MESSAGE_DELIVERY_MODE** - Either buffered or persistent

<table>
<thead>
<tr>
<th>PROP_VALUE</th>
<th>VARCHAR2(30)</th>
<th>Property value</th>
</tr>
</thead>
</table>
STREAMS$PA_COMPONENT_STAT Table

The STREAMS$PA_COMPONENT_STAT table displays performance statistics and session statistics about each Oracle Streams component.

Table 277-4  STREAMS$PA_COMPONENT_STAT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVISOR_RUN_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number of the Oracle Streams Performance Advisor run</td>
</tr>
<tr>
<td>ADVISOR_RUN_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the Oracle Streams Performance Advisor was run for the advisor run ID</td>
</tr>
<tr>
<td>COMPONENT_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number assigned to the component by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>STATISTIC_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the statistic was recorded</td>
</tr>
<tr>
<td>STATISTIC_NAME</td>
<td>VARCHAR2(64)</td>
<td></td>
<td>Name of the statistic</td>
</tr>
<tr>
<td>STATISTIC_VALUE</td>
<td>NUMBER</td>
<td></td>
<td>Value recorded for the statistic</td>
</tr>
<tr>
<td>STATISTIC_UNIT</td>
<td>VARCHAR2(64)</td>
<td></td>
<td>Unit of measurement for the statistic</td>
</tr>
<tr>
<td>SUB_COMPONENT_TYPE</td>
<td>VARCHAR2(64)</td>
<td></td>
<td>Type of the subcomponent</td>
</tr>
</tbody>
</table>

Only capture processes and apply processes have subcomponents.

The following capture process subcomponent types are possible:

- LOGMINER READER for a builder server of a capture process
- LOGMINER PREPARER for a preparer server of a capture process
- LOGMINER BUILDER for a reader server of a capture process
- CAPTURE SESSION for a capture process session

The following apply process subcomponent types are possible:

- PROPAGATION SENDER+RECEIVER for sending LCRs from a capture process directly to an apply process in a combined capture and apply configuration in which both the capture process and apply process run on a single database
- APPLY READER for a reader server of an apply process
- APPLY COORDINATOR for a coordinator process of an apply process
- APPLY SERVER for a reader server of an apply process
### Table 277-4  (Cont.) STREAMS$_PA_COMPONENT_STAT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number of the session for the component. Query the V$SESSION view for information about the session.</td>
</tr>
<tr>
<td>SESSION_SERIAL#</td>
<td>NUMBER</td>
<td></td>
<td>Session serial number of the session for the component. Query the V$SESSION view for information about the session.</td>
</tr>
</tbody>
</table>

### STREAMS$_PA_CONTROL Table

The STREAMS$_PA_CONTROL table displays the parameters set for the COLLECT_STATS procedure in this package. The parameters control the monitoring behavior.

### Table 277-5  STREAMS$_PA_CONTROL Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVISOR_RUN_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number of the Oracle Streams Performance Advisor run</td>
</tr>
<tr>
<td>ADVISOR_RUN_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the Oracle Streams Performance Advisor was last run</td>
</tr>
<tr>
<td>PARAM_NAME</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>The name of the parameter</td>
</tr>
<tr>
<td>PARAM_VALUE</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>The value set for the parameter</td>
</tr>
<tr>
<td>PARAM_UNIT</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>The unit of the parameter</td>
</tr>
</tbody>
</table>

### STREAMS$_PA_DATABASE Table

The STREAMS$_PA_DATABASE table displays information about each database that contains Oracle Streams components.

### Table 277-6  STREAMS$_PA_DATABASE Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_NAME</td>
<td>VARCHAR2(128)</td>
<td>NOT NULL</td>
<td>Global name of the database analyzed by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>LAST_QUERIED</td>
<td>DATE</td>
<td></td>
<td>The time when the Performance Advisor successfully collected information from a database in its last run</td>
</tr>
<tr>
<td>ERROR_NUMBER</td>
<td>NUMBER</td>
<td></td>
<td>The error number of the error encountered when the database was last queried</td>
</tr>
<tr>
<td>ERROR_MESSAGE</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>The error message of the error encountered when the database was last queried</td>
</tr>
</tbody>
</table>
STREAMS$_PA_DATABASE_PROP Table

The STREAMS$_PA_DATABASE_PROP table displays Oracle Streams database property information necessary for analysis by the Streams Performance Advisor.

Table 277-7  STREAMS$_PA_DATABASE_PROP Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_NAME</td>
<td>VARCHAR2(128)</td>
<td>NOT NULL</td>
<td>Global name of the database analyzed by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>PROP_NAME</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>Property name</td>
</tr>
<tr>
<td>PROP_VALUE</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>The database properties include the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· VERSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· COMPATIBILITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· MANAGEMENT_PACK_ACCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>· DB_UNIQUE_NAME</td>
</tr>
</tbody>
</table>

STREAMS$_PA_MONITORING Table

The STREAMS$_PA_MONITORING table displays information about each monitoring job running in a database.

Table 277-8  STREAMS$_PA_MONITORING Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_NAME</td>
<td>VARCHAR2(30)</td>
<td>NOT NULL</td>
<td>Name of the monitoring job</td>
</tr>
<tr>
<td>CLIENT_NAME</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>Name of the client that submitted the job</td>
</tr>
<tr>
<td>QUERY_USER_NAME</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>User granted privileges to view the monitoring results</td>
</tr>
<tr>
<td>SHOW_STATS_TABLE</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>Name of the table used by the SHOW_STATS procedure to display statistics</td>
</tr>
<tr>
<td>STARTED_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time the monitoring job started</td>
</tr>
<tr>
<td>STOPPED_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time the monitoring job last stopped</td>
</tr>
<tr>
<td>ALTERED_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time the monitoring job was last altered</td>
</tr>
<tr>
<td>STATE</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>State of the monitoring job, either ENABLED or STOPPED</td>
</tr>
</tbody>
</table>

STREAMS$_PA_PATH_BOTTLENECK Table

The STREAMS$_PA_PATH_BOTTLENECK table displays information about Oracle Streams components that might be slowing down the flow of messages.
Table 277-9  STREAMS$_PA_PATH_BOTTLENECK Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVISOR_RUN_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number of the Oracle Streams Performance Advisor run</td>
</tr>
<tr>
<td>ADVISOR_RUN_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the Oracle Streams Performance Advisor was last run</td>
</tr>
<tr>
<td>ADVISOR_RUN_REASON</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>Reason for the bottleneck</td>
</tr>
<tr>
<td>PATH_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>PATH_KEY</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>Unique key assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>COMPONENT_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number assigned to the component by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>TOP_SESSION_ID</td>
<td>NUMBER</td>
<td></td>
<td>Session ID of the top component. Query the V$SESSION view for information about the session.</td>
</tr>
<tr>
<td>TOP_SESSION_SERIAL #</td>
<td>NUMBER</td>
<td></td>
<td>Session serial number of the top component. Query the V$SESSION view for information about the session.</td>
</tr>
<tr>
<td>ACTION_NAME</td>
<td>VARCHAR2(32)</td>
<td></td>
<td>Action name for the top session</td>
</tr>
<tr>
<td>BOTTLENECK_IDENTIFIED</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>Whether a bottleneck was identified</td>
</tr>
</tbody>
</table>

STREAMS$_PA_PATH_STAT Table

The STREAMS$_PA_PATH_STAT table displays performance statistics about each stream path.

Table 277-10  STREAMS$_PA_PATH_STAT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVISOR_RUN_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number of the Oracle Streams Performance Advisor run</td>
</tr>
<tr>
<td>ADVISOR_RUN_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the Oracle Streams Performance Advisor was run for the advisor run ID</td>
</tr>
<tr>
<td>PATH_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>PATH_KEY</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>Unique key assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>STATISTIC_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the statistic was recorded</td>
</tr>
</tbody>
</table>
Table 277-10  (Cont.) STREAMS$_PA_PATH_STAT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATISTIC_NAME</td>
<td>VARCHAR2(64)</td>
<td></td>
<td>Name of the statistic</td>
</tr>
<tr>
<td>STATISTIC_VALUE</td>
<td>NUMBER</td>
<td></td>
<td>Value recorded for the statistic</td>
</tr>
<tr>
<td>STATISTIC_UNIT</td>
<td>VARCHAR2(64)</td>
<td></td>
<td>Unit of measurement for the statistic</td>
</tr>
</tbody>
</table>

STREAMS$_PA_SHOW_COMP_STAT Table

The STREAMS$_PA_SHOW_COMP_STAT table displays statistics for Oracle Streams components.

Table 277-11  STREAMS$_PA_SHOW_COMP_STAT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVISOR_RUN_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number of the Oracle Streams Performance Advisor run</td>
</tr>
<tr>
<td>ADVISOR_RUN_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the Oracle Streams Performance Advisor was last run</td>
</tr>
<tr>
<td>PATH_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>POSITION</td>
<td>NUMBER</td>
<td></td>
<td>Position of the component in the path</td>
</tr>
<tr>
<td>COMPONENT_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number assigned to the component by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>COMPONENT_NAME</td>
<td>VARCHAR2(194)</td>
<td></td>
<td>Name of the component</td>
</tr>
<tr>
<td>COMPONENT_TYPE</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>Type of the component</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The following types are possible:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CAPTURE for a capture process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- PROPAGATION SENDER for a propagation sender</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- PROPAGATION RECEIVER for a propagation receiver</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- APPLY for an apply process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- QUEUE for a queue</td>
</tr>
</tbody>
</table>
### Table 277-11  (Cont.) STREAMS$_PA_SHOW_COMP_STAT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB_COMPONENT_TY</td>
<td>VARCHAR2(30)</td>
<td>PE</td>
<td>Type of the subcomponent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Only capture processes and apply processes have subcomponents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The following capture process subcomponent types are possible:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• LOGMINER READER for a builder server of a capture process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• LOGMINER PREPARER for a preparer server of a capture process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• LOGMINER BUILDER for a reader server of a capture process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CAPTURE SESSION for a capture process session</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The following apply process subcomponent types are possible:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• PROPAGATION SENDER+RECEIVER for sending LCRs from a capture process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>directly to an apply process in a combined capture and apply configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in which both the capture process and apply process run on a single database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• APPLY READER for a reader server of an apply process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• APPLY COORDINATOR for a coordinator process of an apply process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• APPLY SERVER for a reader server of an apply process</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>NUMBER</td>
<td></td>
<td>Identification number of the session for the component. Query the V$SESSION view for information about the session.</td>
</tr>
<tr>
<td>SESSION_SERIAL#</td>
<td>NUMBER</td>
<td></td>
<td>Session serial number of the session for the component. Query the V$SESSION view for information about the session.</td>
</tr>
<tr>
<td>STATISTIC_ALIAS</td>
<td>VARCHAR2(30)</td>
<td></td>
<td>Name of the statistic</td>
</tr>
<tr>
<td>STATISTIC_NAME</td>
<td>VARCHAR2(128)</td>
<td></td>
<td>Name of the statistic</td>
</tr>
<tr>
<td>STATISTIC_VALUE</td>
<td>NUMBER</td>
<td></td>
<td>Value recorded for the statistic</td>
</tr>
<tr>
<td>STATISTIC_UNIT</td>
<td>VARCHAR2(128)</td>
<td></td>
<td>Unit of measurement for the statistic</td>
</tr>
</tbody>
</table>

**STREAMS$_PA_SHOW_PATH_STAT Table**

The **STREAMS$_PA_SHOW_PATH_STAT** table displays statistics for the stream paths in an Oracle Streams configuration. A monitoring job uses this table as the default table for the statistics collected for stream paths.
Table 277-12  STREAMS$_PA_SHOW_PATH_STAT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH_ID</td>
<td>NUMBER</td>
<td>NULL</td>
<td>Identification number assigned to the path by the Oracle Streams Performance Advisor</td>
</tr>
<tr>
<td>ADVISOR_RUN_ID</td>
<td>NUMBER</td>
<td>NULL</td>
<td>Identification number of the Oracle Streams Performance Advisor run</td>
</tr>
<tr>
<td>ADVISOR_RUN_TIME</td>
<td>DATE</td>
<td></td>
<td>Time when the Oracle Streams Performance Advisor was last run</td>
</tr>
<tr>
<td>SETTING</td>
<td>VARCHAR2(2000)</td>
<td></td>
<td>Setting for the Oracle Streams Performance Advisor Run</td>
</tr>
<tr>
<td>STATISTICS</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>Component-level statistics</td>
</tr>
<tr>
<td>SESSION_STATISTICS</td>
<td>VARCHAR2(4000)</td>
<td></td>
<td>Session-level statistics</td>
</tr>
<tr>
<td>OPTIMIZATION</td>
<td>NUMBER</td>
<td></td>
<td>Whether the path uses the combined capture and apply optimization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 (zero) means that the path does not use the combined capture and apply optimization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 means that the path uses the combined capture and apply optimization.</td>
</tr>
</tbody>
</table>

277.4 Summary of UTL_SPADV Subprograms

This table lists the UTL_SPADV subprograms and briefly describes them.

Table 277-13  UTL_SPADV Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER_MONITORING Procedure</td>
<td>Alters the monitoring job submitted by the current user.</td>
</tr>
<tr>
<td>COLLECT_STATS Procedure</td>
<td>Uses the Oracle Streams Performance Advisor to gather statistics about the Oracle Streams components and subcomponents in a distributed database environment.</td>
</tr>
<tr>
<td>IS_MONITORING Function</td>
<td>Checks whether a monitoring job is currently running.</td>
</tr>
<tr>
<td>SHOW_STATS Procedure</td>
<td>Generates output that includes the statistics gathered by the COLLECT_STATS procedure.</td>
</tr>
<tr>
<td>SHOW_STATS_HTML Procedure</td>
<td>Generates HTML output that includes the statistics gathered by the COLLECT_STATS procedure.</td>
</tr>
<tr>
<td>START_MONITORING Procedure</td>
<td>Starts a monitoring job.</td>
</tr>
<tr>
<td>STOP_MONITORING Procedure</td>
<td>Stops a monitoring job.</td>
</tr>
</tbody>
</table>
277.4.1 ALTER_MONITORING Procedure

This procedure alters the monitoring job submitted by the current user.

Syntax

```
UTL_SPADV.ALTER_MONITORING(
  interval                      IN NUMBER  DEFAULT NULL,
  top_event_threshold           IN NUMBER  DEFAULT NULL,
  bottleneck_idle_threshold     IN NUMBER  DEFAULT NULL,
  bottleneck_flowctrl_threshold IN NUMBER  DEFAULT NULL,
  retention_time                IN NUMBER  DEFAULT NULL);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval</td>
<td>The amount of time, in seconds, between each Performance Advisor run. The maximum is 3600 seconds.</td>
</tr>
<tr>
<td></td>
<td>For example, if the value is NULL, then the current value is not changed.</td>
</tr>
<tr>
<td>top_event_threshold</td>
<td>A percentage that determines whether a top wait event statistic is collected.</td>
</tr>
<tr>
<td></td>
<td>The percentage for a wait event must be greater than the value specified in this parameter for the procedure to collect the wait event statistic. For example, if 15 is specified, then only wait events with a value larger than 15% are collected.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the current value is not changed.</td>
</tr>
<tr>
<td>bottleneck_idle_threshold</td>
<td>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its IDLE percentage.</td>
</tr>
<tr>
<td></td>
<td>The IDLE percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are idle 50% of the time or less are eligible for bottleneck analysis.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the current value is not changed.</td>
</tr>
<tr>
<td>bottleneck_flowctrl_threshold</td>
<td>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its FLOW CONTROL percentage.</td>
</tr>
<tr>
<td></td>
<td>The FLOW CONTROL percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are paused for flow control 50% of the time or less are eligible for bottleneck analysis.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the current value is not changed.</td>
</tr>
</tbody>
</table>
Table 277-14  (Cont.) ALTER_MONITORING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retention_time</td>
<td>The number of hours to retain monitoring results. If NULL, then the current value is not changed.</td>
</tr>
</tbody>
</table>

Exceptions

Table 277-15  ALTER_MONITORING Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-20113</td>
<td>no active monitoring job found</td>
</tr>
</tbody>
</table>

277.4.2 COLLECT_STATS Procedure

This procedure uses the Oracle Streams Performance Advisor to gather statistics about the Oracle Streams components and subcomponents in a distributed database environment.

**Note:**

This procedure commits.

**See Also:**

*Oracle Streams Concepts and Administration* for more information about the Oracle Streams Performance Advisor.

Syntax

```sql
UTL_SPADV.COLLECT_STATS(
   interval                      IN NUMBER  DEFAULT 60,
   num_runs                      IN NUMBER  DEFAULT 10,
   comp_stat_table             IN VARCHAR2  DEFAULT 'STREAMS$ADVISOR_COMP_STAT',
   path_stat_table             IN VARCHAR2  DEFAULT 'STREAMS$ADVISOR_PATH_STAT',
   top_event_threshold           IN NUMBER  DEFAULT 15,
   bottleneck_idle_threshold     IN NUMBER  DEFAULT 50,
   bottleneck_flowctrl_threshold IN NUMBER  DEFAULT 50);
```

Parameters

Table 277-16  COLLECT_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval</td>
<td>The amount of time, in seconds, between each Performance Advisor run. The maximum is 3600 seconds.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>num_runs</td>
<td>The number of times that the Oracle Streams Performance Advisor is run by the procedure.</td>
</tr>
<tr>
<td>comp_stat_table</td>
<td>The name of the table that stores the statistics collected for Oracle Streams components and subcomponents. Specify the table name as [schema_name.]object_name. If the schema is not specified, then the current user is the default. The procedure creates the specified table if it does not exist. Oracle recommends that you use the default table STREAMS$_ADVISOR_COMP_STAT. See &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
<tr>
<td>path_stat_table</td>
<td>The name of the table that stores the statistics collected for stream paths. Specify the table name as [schema_name.]object_name. If the schema is not specified, then the current user is the default. The procedure creates the specified table if it does not exist. Oracle recommends that you use the default table STREAMS$_ADVISOR_PATH_STAT. See &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
<tr>
<td>top_event_threshold</td>
<td>A percentage that determines whether a top wait event statistic is collected. The percentage for a wait event must be greater than the value specified in this parameter for the procedure to collect the wait event statistic. For example, if 15 is specified, then only wait events with a value larger than 15% are collected.</td>
</tr>
<tr>
<td>bottleneck_idle_threshold</td>
<td>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its IDLE percentage. The IDLE percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are idle 50% of the time or less are eligible for bottleneck analysis.</td>
</tr>
<tr>
<td>bottleneck_flowctrl_threshold</td>
<td>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its FLOW CONTROL percentage. The FLOW CONTROL percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are paused for flow control 50% of the time or less are eligible for bottleneck analysis.</td>
</tr>
</tbody>
</table>
Usage Notes

The table specified in the \texttt{path\_stat\_table} parameter stores stream path statistics. This table also concatenates the component and subcomponent statistics stored in the table specified in the \texttt{comp\_stat\_table} parameter. The \texttt{SHOW\_STATS} procedure in this package shows only the statistics stored in the table specified in the \texttt{path\_stat\_table} parameter.

277.4.3 IS\_MONITORING Function

This function checks whether a monitoring job is currently running. This function either returns \texttt{TRUE} if a monitoring job is currently running or \texttt{FALSE} if a monitoring job is not currently running.

A monitoring job is submitted using the \texttt{START\_MONITORING} procedure.

\begin{verbatim}
See Also:

"START\_MONITORING Procedure"
\end{verbatim}

Syntax

\begin{verbatim}
UTL\_SPADV.IS\_MONITORING(
    job_name    IN VARCHAR2  DEFAULT 'STREAMS\$_MONITORING\_JOB',
    client_name IN VARCHAR2  DEFAULT NULL)
RETURN BOOLEAN;
\end{verbatim}

Parameters

\begin{table}
\centering
\begin{tabular}{|l|p{0.7\textwidth}|}
\hline
Parameter & Description																																																																																																																																																																																																																																																																																																																																																																																																																																						\hline
job\_name & The name of the job for which to check. \\
client\_name & The name of the client that submitted the job. \\
\hline
\end{tabular}
\end{table}

277.4.4 SHOW\_STATS Procedure

This procedure generates output that includes the statistics gathered by the \texttt{COLLECT\_STATS} and \texttt{START\_MONITORING} procedures.

The output is formatted so that it can be imported into a spreadsheet for analysis.

\begin{verbatim}
 Note: 

This procedure does not commit.
\end{verbatim}
See Also:

- "COLLECT_STATS Procedure"
- "START_MONITORING Procedure"
- Oracle Streams Concepts and Administration for more information about the Oracle Streams Performance Advisor

Syntax

```sql
UTL_SPADV.SHOW_STATS(
    path_stat_table IN VARCHAR2  DEFAULT 'STREAMS$_ADVISOR_PATH_STAT',
    path_id IN NUMBER  DEFAULT NULL,
    bgn_run_id IN NUMBER  DEFAULT -1,
    end_run_id IN NUMBER  DEFAULT -10,
    show_path_id IN BOOLEAN  DEFAULT TRUE,
    show_run_id IN BOOLEAN  DEFAULT TRUE,
    show_run_time IN BOOLEAN  DEFAULT TRUE,
    show_optimization IN BOOLEAN  DEFAULT TRUE,
    show_setting IN BOOLEAN  DEFAULT FALSE,
    show_stat IN BOOLEAN  DEFAULT TRUE,
    show_sess IN BOOLEAN  DEFAULT TRUE,
    show_legend IN BOOLEAN  DEFAULT TRUE);
```

Parameters

Table 277-18  SHOW_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path_stat_table</td>
<td>The name of the table that contains the stream path statistics. Specify the table name as [schema_name.object_name]. If the schema is not specified, then the current user is the default. When you gather statistics using the COLLECT_STATS procedure, this table is specified in the path_stat_table parameter in the COLLECT_STATS procedure. The default table is STREAMS$_ADVISOR_PATH_STAT. When you gather statistics using the START_MONITORING procedure, you can determine the name for this table by querying the SHOW_STATS_TABLE column in the STREAMS$_PA_MONITORING view. The default table for a monitoring job is STREAMS$_PA_SHOW_PATH_STAT.</td>
</tr>
<tr>
<td>path_id</td>
<td>A stream path ID. If non-NULL, then the procedure shows output for the specified stream path only. If NULL, then the procedure shows output for all active stream paths.</td>
</tr>
<tr>
<td>bgn_run_id</td>
<td>The first Oracle Streams Performance Advisor run ID to show in the range of runs. See &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
</tbody>
</table>
### Table 277-18 (Cont.) SHOW_STATS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end_run_id</td>
<td>The last Oracle Streams Performance Advisor run ID to show in the range of runs. See &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
<tr>
<td>show_path_id</td>
<td>If TRUE, then the path ID for each stream path is included in the output. If FALSE, then the path ID for each stream path is not included in the output.</td>
</tr>
<tr>
<td>show_run_id</td>
<td>If TRUE, then the Oracle Streams Performance Advisor run ID is included in the output. If FALSE, then the Oracle Streams Performance Advisor run ID is not included in the output.</td>
</tr>
<tr>
<td>show_run_time</td>
<td>If TRUE, then the Oracle Streams Performance Advisor run time is included in the output. If FALSE, then the Oracle Streams Performance Advisor run time is not included in the output.</td>
</tr>
<tr>
<td>show_optimization</td>
<td>If TRUE, then path output includes information pertaining to the combined capture and apply optimization. If FALSE, then path output does not include information pertaining to the combined capture and apply optimization.</td>
</tr>
<tr>
<td>show_setting</td>
<td>If TRUE, then the settings for the threshold parameters are included in the output. The threshold parameters are the <code>top_event_threshold</code>, <code>bottleneck_idle_threshold</code>, and <code>bottleneck_flowctrl_threshold</code> parameters in the <code>COLLECT_STATS</code> procedure. If FALSE, then the settings for the threshold parameters are not included in the output.</td>
</tr>
<tr>
<td>show_stat</td>
<td>If TRUE, then the component-level and subcomponent-level statistics are included in the output. These components include capture processes, queues, propagation senders, propagation receivers, and apply processes. The subcomponents are the subcomponents for capture processes and apply processes. If FALSE, then the component-level and subcomponent-level statistics are not included in the output.</td>
</tr>
<tr>
<td>show_sess</td>
<td>If TRUE, then the session-level statistics are included in the output. Session-level statistics include IDLE, FLOW CONTROL, and EVENT statistics. If FALSE, then the session-level statistics are not included in the output.</td>
</tr>
<tr>
<td>show_legend</td>
<td>If TRUE, then the legend is included in the output. The legend describes the abbreviations used in the output. If FALSE, then the legend is not included in the output.</td>
</tr>
</tbody>
</table>

#### Usage Notes

Use the `bgn_run_id` and `end_run_id` together to specify the range of Oracle Streams Performance Advisor runs to display. Positive numbers show statistics from an earlier run forward. Negative numbers show statistics from a later run backward.
For example, if `bgn_run_id` is set to 1 and `end_run_id` is set to 10, then the procedure shows statistics for the first ten Oracle Streams Performance Advisor runs.

However, if `bgn_run_id` is set to -1 and `end_run_id` is set to -10, then the procedure shows statistics for the last ten Oracle Streams Performance Advisor runs.

See Also:

*Oracle Streams Concepts and Administration* for information about the combined capture and apply optimization.

### 277.4.5 SHOW_STATS_HTML Procedure

This procedure generates HTML output that includes the statistics gathered by the `COLLECT_STATS` and `START_MONITORING` procedures.

**Note:**

This procedure does not commit.

See Also:

- "COLLECT_STATS Procedure"
- "START_MONITORING Procedure"
- *Oracle Streams Concepts and Administration* for more information about the Oracle Streams Performance Advisor

**Syntax**

```sql
UTL_SPADV.SHOW_STATS_HTML(
    directory       IN VARCHAR2,
    reportname      IN VARCHAR2  DEFAULT 'SPADVREPORT.HTML',
    comp_stat_table IN VARCHAR2  DEFAULT 'STREAMS$ _ADVISOR_COMP_STAT',
    path_id         IN NUMBER    DEFAULT NULL,
    bgn_run_id      IN NUMBER    DEFAULT -1,
    end_run_id      IN NUMBER    DEFAULT -10,
    detailed        IN BOOLEAN   DEFAULT TRUE);
```
### Parameters

#### Table 277-19  SHOW_STATS_HTML Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>The directory object for the directory on the local computer system into which the generated HTML report is placed. The specified directory object must be created using the SQL statement <code>CREATE DIRECTORY</code>, and the user who invokes the procedure must have <code>READ</code> and <code>WRITE</code> privilege on each one.</td>
</tr>
<tr>
<td>reportname</td>
<td>The name of the HTML report.</td>
</tr>
<tr>
<td>comp_stat_table</td>
<td>The name of the table that stores the statistics collected for Oracle Streams components and subcomponents. Specify the table name as <code>[schema_name].object_name</code>. If the schema is not specified, then the current user is the default.</td>
</tr>
<tr>
<td></td>
<td>When you gather statistics using the <code>COLLECT_STATS</code> procedure, this table is specified in the <code>comp_stat_table</code> parameter in the <code>COLLECT_STATS</code> procedure. The default table is <code>STREAMS$_ADVISOR_COMP_STAT</code>.</td>
</tr>
<tr>
<td></td>
<td>When you gather statistics using the <code>START_MONITORING</code> procedure, you can determine the name for this table by querying the <code>SHOW_STATS_TABLE</code> column in the <code>STREAMS$_PA_MONITORING</code> view. The default table for a monitoring job is <code>STREAMS$_PA_SHOW_PATH_STAT</code>.</td>
</tr>
<tr>
<td></td>
<td>Oracle recommends that you start a monitoring job with the <code>START_MONITORING</code> procedure in this package and use the appropriate the <code>STREAMS$_PA_SHOW_PATH_STAT</code> table.</td>
</tr>
<tr>
<td>path_id</td>
<td>A stream path ID.</td>
</tr>
<tr>
<td></td>
<td>If non-NULL, then the procedure shows output for the specified stream path only.</td>
</tr>
<tr>
<td></td>
<td>If NULL, then the procedure shows output for all active stream paths.</td>
</tr>
<tr>
<td>bgn_run_id</td>
<td>The first Oracle Streams Performance Advisor run ID to show in the range of runs.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
<tr>
<td>end_run_id</td>
<td>The last Oracle Streams Performance Advisor run ID to show in the range of runs.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Usage Notes&quot; for more information about this parameter.</td>
</tr>
<tr>
<td>detailed</td>
<td>If <code>TRUE</code>, then the procedure generates component-level statistics.</td>
</tr>
<tr>
<td></td>
<td>If <code>FALSE</code>, then the procedure does not generate component-level statistics.</td>
</tr>
</tbody>
</table>

### Usage Notes

Use the `bgn_run_id` and `end_run_id` together to specify the range of Oracle Streams Performance Advisor runs to display. Positive numbers show statistics from an earlier run forward. Negative numbers show statistics from a later run backward.

For example, if `bgn_run_id` is set to 1 and `end_run_id` is set to 10, then the procedure shows statistics for the first ten Oracle Streams Performance Advisor runs.
However, if \( \text{bgn\_run\_id} \) is set to \(-1\) and \( \text{end\_run\_id} \) is set to \(-10\), then the procedure shows statistics for the last ten Oracle Streams Performance Advisor runs.

### 277.4.6 START_MONITORING Procedure

This procedure starts a monitoring job.

This procedure runs the `COLLECT_STATS` procedure to gather statistics about the Oracle Streams components and subcomponents in a distributed database environment.

#### Note:

This procedure commits.

#### See Also:

- "COLLECT_STATS Procedure"
- *Oracle Streams Concepts and Administration* for more information about the Oracle Streams Performance Advisor

#### Syntax

```sql
UTL_SPADV.START_MONITORING(
    job_name                      IN VARCHAR2  DEFAULT 'STREAMS$$_MONITORING_JOB',
    client_name                   IN VARCHAR2  DEFAULT NULL,
    query_user_name               IN VARCHAR2  DEFAULT NULL,
    interval                      IN NUMBER    DEFAULT 60,
    top_event_threshold           IN NUMBER    DEFAULT 15,
    bottleneck_idle_threshold     IN NUMBER    DEFAULT 50,
    bottleneck_flowctrl_threshold IN NUMBER    DEFAULT 50,
    retention_time                IN NUMBER    DEFAULT 24);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>job_name</code></td>
<td>The name of the monitoring job to create.</td>
</tr>
<tr>
<td><code>client_name</code></td>
<td>The name of the client.</td>
</tr>
<tr>
<td><code>query_user_name</code></td>
<td>The user who will query the result tables.</td>
</tr>
<tr>
<td><code>interval</code></td>
<td>This procedure grants privileges to the specified user to enable the user to query the result tables. This specified interval is used for the interval parameter in the <code>COLLECT_STATS</code> procedure.</td>
</tr>
<tr>
<td><code>top_event_threshold</code></td>
<td>The amount of time, in seconds, between each Performance Advisor run. The maximum is 3600 seconds.</td>
</tr>
<tr>
<td><code>bottleneck_idle_threshold</code></td>
<td>The specified interval is used for the interval parameter in the <code>COLLECT_STATS</code> procedure.</td>
</tr>
<tr>
<td><code>bottleneck_flowctrl_threshold</code></td>
<td>The specified interval is used for the interval parameter in the <code>COLLECT_STATS</code> procedure.</td>
</tr>
<tr>
<td><code>retention_time</code></td>
<td>The specified interval is used for the interval parameter in the <code>COLLECT_STATS</code> procedure.</td>
</tr>
</tbody>
</table>
Table 277-20  (Cont.) START_MONITORING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top_event_threshold</td>
<td>A percentage that determines whether a top wait event statistic is collected. The percentage for a wait event must be greater than the value specified in this parameter for the procedure to collect the wait event statistic. For example, if 15 is specified, then only wait events with a value larger than 15% are collected.</td>
</tr>
<tr>
<td>bottleneck_idle_threshold</td>
<td>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its IDLE percentage. The IDLE percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are idle 50% of the time or less are eligible for bottleneck analysis.</td>
</tr>
<tr>
<td>bottleneck_flowctrl1_threshold</td>
<td>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its FLOW CONTROL percentage. The FLOW CONTROL percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are paused for flow control 50% of the time or less are eligible for bottleneck analysis.</td>
</tr>
<tr>
<td>retention_time</td>
<td>The number of hours to retain monitoring results.</td>
</tr>
</tbody>
</table>

Exceptions

Table 277-21  START_MONITORING Procedure Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
</table>
| ORA-20111  | cannot start monitoring due to active EM monitoring job
Stop the Oracle Enterprise Manager (EM) monitoring job, and run the START_MONITORING procedure again. |
| ORA-20112  | cannot start monitoring due to active Streams monitoring job
Stop the Streams monitoring job, and run the START_MONITORING procedure again. |

Usage Notes

The following are usage notes for the START_MONITORING procedure:

- Requirements for the User Running the Procedure
- Full Monitoring Job Names
- Restrictions on Monitoring Jobs
Requirements for the User Running the Procedure

The user who runs the `START_MONITORING` procedure must meet the following requirements:

- The user must have access to a database link to each database that contains Oracle Streams components.
- The user must have been granted privileges using the `DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE` procedure, and each database link must connect to a user at the remote database that has been granted privileges using the `DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE` procedure.

Full Monitoring Job Names

When you submit a monitoring job, the client name and job name are concatenated to form the full monitoring job name. You specify the client name using the `client_name` parameter and the job name using the `job_name` parameter when you run the `START_MONITORING` procedure. The client name for a monitoring job submitted by Oracle Enterprise Manager is always `EM`.

The following table show examples of full monitoring job names:

<table>
<thead>
<tr>
<th>Setting for client_name parameter</th>
<th>Setting for job_name parameter</th>
<th>Full Monitoring Job Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>STREAMS$$_MONITORING_JOB</td>
<td>STREAMS$$_MONITORING_JOB</td>
</tr>
<tr>
<td>EM</td>
<td>STREAMS$$_MONITORING_JOB</td>
<td>EMSTREAMS$$_MONITORING_JOB</td>
</tr>
<tr>
<td>strm</td>
<td>STREAMS$$_MONITORING_JOB</td>
<td>strmSTREAMS$$_MONITORING_JOB</td>
</tr>
<tr>
<td>strm</td>
<td>mjob1</td>
<td>strmjob1</td>
</tr>
</tbody>
</table>

Restrictions on Monitoring Jobs

The following restrictions apply to monitoring jobs:

- The limit for the length of the full monitoring job name is 30 bytes.
- Two monitoring jobs cannot have the same full monitoring job name, even if the monitoring jobs were submitted by different schemas. The name check is not case-sensitive. For example, `strmSTREAMS$$_MONITORING_JOB` and `strmSTREAMS$$_MONITORING_JOB` are considered to be the same name.
- Oracle Enterprise Manager can have at most one monitoring job for each database.
- Each schema can have at most one monitoring job.

277.4.7 STOP_MONITORING Procedure

This procedure stops a monitoring job that was submitted by the current user.

Syntax

```sql
UTL_SPADV.STOP_MONITORING(
    purge IN BOOLEAN DEFAULT FALSE);
```
Parameters

Table 277-22  STOP_MONITORING Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| purge     | If **TRUE**, then the procedure purges information about the monitoring job from the result tables.  
|           | If **FALSE**, then the procedure retains information about the monitoring job in the result tables. |
With the UTL_TCP package and its procedures and functions, PL/SQL applications can communicate with external TCP/IP-based servers using TCP/IP. Because many Internet application protocols are based on TCP/IP, this package is useful to PL/SQL applications that use Internet protocols and e-mail.

This chapter contains the following topics:

- Overview
- Security Model
- Types
- Exceptions
- Rules and Limits
- Examples
- Summary of UTL_TCP Subprograms

278.1 UTL_TCP Overview

The UTL_TCP package provides TCP/IP client-side access functionality in PL/SQL.

278.2 UTL_TCP Security Model

This package is an invoker’s rights package and the invoking user needs the connect privilege granted in the access control list assigned to the remote network host to which he wants to connect.

**Note:**

For more information about managing fine-grained access, see Oracle Database Security Guide.

278.3 UTL_TCP Types

The UTL_TCP package includes a CONNECTION type and a carriage-return line-feed (CRLF) type.

**CONNECTION Type**

This is a PL/SQL record type used to represent a TCP/IP connection.
Syntax

```sql
TYPE connection IS RECORD (
    remote_host VARCHAR2(255),
    remote_port PLS_INTEGER,
    local_host VARCHAR2(255),
    local_port PLS_INTEGER,
    charset VARCHAR2(30),
    newline VARCHAR2(2),
    tx_timeout PLS_INTEGER,
    private_sd PLS_INTEGER);
```

Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_host</td>
<td>Name of the remote host when connection is established. NULL when no connection is established.</td>
</tr>
<tr>
<td>remote_port</td>
<td>Port number of the remote host connected. NULL when no connection is established.</td>
</tr>
<tr>
<td>local_host</td>
<td>Name of the local host used to establish the connection. NULL when no connection is established.</td>
</tr>
<tr>
<td>local_port</td>
<td>Port number of the local host used to establish the connection. NULL when no connection is established.</td>
</tr>
<tr>
<td>charset</td>
<td>The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database are converted to and from the on-the-wire character set as they are sent and received on the network.</td>
</tr>
<tr>
<td>newline</td>
<td>Newline character sequence. This newline character sequence is appended to the text line sent by WRITE_LINE API.</td>
</tr>
<tr>
<td>tx_timeout</td>
<td>Time in seconds that the UTL_TCP package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. NULL indicates to wait forever.</td>
</tr>
</tbody>
</table>

Usage Notes

The fields in a connection record are used to return information about the connection, which is often made using OPEN_CONNECTION. Changing the values of those fields has no effect on the connection. The fields private_XXXX are for implementation use only. You should not modify the values.

In the current release of the UTL_TCP package, the parameters local_host and local_port are ignored when open_connection makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The local_host and local_port fields are not set in the connection record returned by the function.
Time out on write operations is not supported in the current release of the UTL_TCP package.

CRLF

The character sequence carriage-return line-feed. It is the newline sequence commonly used by many communication standards.

Syntax

CRLF CONSTANT VARCHAR2(2 CHAR);

Usage Notes

This package variable defines the newline character sequence commonly used in many Internet protocols. This is the default value of the newline character sequence for WRITE_LINE, specified when a connection is opened. While such protocols use \textless CR\textgreater LF to denote a new line, some implementations may choose to use just line-feed to denote a new line. In such cases, users can specify a different newline character sequence when a connection is opened.

278.4 UTL_TCP Exceptions

UTL_TCP will raise an exception when it encounters a processing issue.

The exceptions raised by the TCP/IP package are listed in the following table.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER_TOO_SMALL</td>
<td>Buffer is too small for input that requires look-ahead</td>
</tr>
<tr>
<td>END_OF_INPUT</td>
<td>Raised when no more data is available to read from the connection</td>
</tr>
<tr>
<td>NETWORK_ERROR</td>
<td>Generic network error</td>
</tr>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument passed in an API call (for example, a negative buffer size)</td>
</tr>
<tr>
<td>TRANSFER_TIMEOUT</td>
<td>No data is read and a read time out occurred</td>
</tr>
<tr>
<td>PARTIAL_MULTIBYTE_CHAR</td>
<td>No complete character is read and a partial multibyte character is found at the end of the input</td>
</tr>
</tbody>
</table>

278.5 UTL_TCP Rules and Limits

The interface provided in the package only allows connections to be initiated by the PL/SQL program. It does not allow the PL/SQL program to accept connections initiated outside the program.
278.6 UTL_TCP Examples

Some possible uses for UTL_TCP include retrieving a Web page over HTTP or sending an e-mail.

The following code example illustrates how the TCP/IP package can be used to retrieve a Web page over HTTP. It connects to a Web server listening at port 80 (standard port for HTTP) and requests the root document.

```sql
DECLARE
c  utl_tcp.connection;  -- TCP/IP connection to the Web server
ret_val pls_integer;
BEGIN
  c := utl_tcp.open_connection(remote_host => 'www.acme.com',
                              remote_port => 80,
                              charset     => 'US7ASCII');  -- open connection
  ret_val := utl_tcp.write_line(c, 'GET / HTTP/1.0');    -- send HTTP request
  ret_val := utl_tcp.write_line(c);
  BEGIN
    LOOP
      dbms_output.put_line(utl_tcp.get_line(c, TRUE));  -- read result
    END LOOP;
  EXCEPTION
    WHEN utl_tcp.end_of_input THEN
      NULL; -- end of input
  END;
  utl_tcp.close_connection(c);
END;
```

The following code example illustrates how the TCP/IP package can be used by an application to send e-mail (also known as email from PL/SQL). The application connects to an SMTP server at port 25 and sends a simple text message.

```sql
PROCEDURE send_mail (sender    IN VARCHAR2,
                     recipient IN VARCHAR2,
                     message   IN VARCHAR2) IS
  mailhost   VARCHAR2(30) := 'mailhost.mydomain.com';
  smtp_error  EXCEPTION;
  mail_conn   utl_tcp.connection;
  PROCEDURE smtp_command(command IN VARCHAR2,
                           ok      IN VARCHAR2 DEFAULT '250')
  IS
    response varchar2(3);
    len pls_integer;
  BEGIN
    len := utl_tcp.write_line(mail_conn, command);
    response := substr(utl_tcp.get_line(mail_conn), 1, 3);
    IF (response <> ok) THEN
      RAISE smtp_error;
    END IF;
  END;
  BEGIN
    mail_conn := utl_tcp.open_connection(remote_host => mailhost,
                                          remote_port => 25,
                                          charset     => 'US7ASCII');
    smtp_command('HELO ' || mailhost);
    smtp_command('MAIL FROM: ' || sender);
    smtp_command('RCPT TO: ' || recipient);
```

Chapter 278
UTL_TCP Examples
### 278.7 Summary of UTL_TCP Subprograms

This table lists the UTL_TCP subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVAILABLE Function</td>
<td>Determines the number of bytes available for reading from a TCP/IP connection</td>
</tr>
<tr>
<td>CLOSE_ALL_CONNECTIONS Procedure</td>
<td>Closes all open TCP/IP connections</td>
</tr>
<tr>
<td>CLOSE_CONNECTION Procedure</td>
<td>Closes an open TCP/IP connection</td>
</tr>
<tr>
<td>FLUSH Procedure</td>
<td>Transmits immediately to the server all data in the output buffer, if a buffer is used</td>
</tr>
<tr>
<td>GET_LINE Function</td>
<td>Returns the line of data read</td>
</tr>
<tr>
<td>GET_LINE_NCHAR Function</td>
<td>Returns the line of data read in NCHAR form</td>
</tr>
<tr>
<td>GET_RAW Function</td>
<td>Return the data read instead of the amount of data read</td>
</tr>
<tr>
<td>GET_TEXT Function</td>
<td>Returns the text data read</td>
</tr>
<tr>
<td>GET_TEXT_NCHAR Function</td>
<td>Returns the text data read in NCHAR form</td>
</tr>
<tr>
<td>OPEN_CONNECTION Function</td>
<td>Opens a TCP/IP connection to a specified service</td>
</tr>
<tr>
<td>READ_LINE Function</td>
<td>Receives a text line from a service on an open connection</td>
</tr>
<tr>
<td>READ_RAW Function</td>
<td>Receives binary data from a service on an open connection</td>
</tr>
<tr>
<td>READ_TEXT Function</td>
<td>Receives text data from a service on an open connection</td>
</tr>
<tr>
<td>SECURE_CONNECTION Procedure</td>
<td>Secures a TCP/IP connection using SSL/TLS</td>
</tr>
<tr>
<td>WRITE_LINE Function</td>
<td>Transmits a text line to a service on an open connection</td>
</tr>
<tr>
<td>WRITE_RAW Function</td>
<td>Transmits a binary message to a service on an open connection</td>
</tr>
<tr>
<td>WRITE_TEXT Function</td>
<td>Transmits a text message to a service on an open connection</td>
</tr>
</tbody>
</table>

#### 278.7.1 AVAILABLE Function

This function determines the number of bytes available for reading from a TCP/IP connection. It is the number of bytes that can be read immediately without blocking. Determines if data is ready to be read from the connection.

**Syntax**

```sql
UTL_TCP.AVAILABLE (c IN OUT NOCOPY connection,
```
timeout  IN PLS_INTEGER DEFAULT 0)
RETURN PLS_INTEGER;

Parameters

Table 278-4 AVAILABLE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection to determine the amount of data that is available to be read</td>
</tr>
<tr>
<td>timeout</td>
<td>Time in seconds to wait before giving up and reporting that no data is available. Zero (0) indicates not to wait at all. NULL indicates to wait forever.</td>
</tr>
</tbody>
</table>

Return Values

The number of bytes available for reading without blocking

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. Users may use this API to determine if data is available to be read before calling the read API so that the program are not blocked because data is not ready to be read from the input.

The number of bytes available for reading returned by this function may be less than what is actually available. On some platforms, this function may only return 1, to indicate that some data is available. If you are concerned about the portability of your application, then assume that this function returns a positive value when data is available for reading, and 0 when no data is available. This function returns a positive value when all the data at a particular connection has been read and the next read result in the END_OF_INPUT exception.

The following example illustrates using this function in a portable manner:

```sql
DECLARE
  c   utl_tcp.connection
  data VARCHAR2(256);
  len  PLS_INTEGER;
BEGIN
  c := utl_tcp.open_connection(...);
  LOOP
    IF (utl_tcp.available(c) > 0) THEN
      len := utl_tcp.read_text(c, data, 256);
    ELSE
      ---do some other things
      . . . .
    END IF
  END LOOP;
END;
```

278.7.2 CLOSE_ALL_CONNECTIONS Procedure

This procedure closes all open TCP/IP connections.

Syntax

\[ \text{UTL_TCP.CLOSE_ALL_CONNECTIONS;} \]

Usage Notes

This call is provided to close all connections before a PL/SQL program ends to avoid dangling connections.

278.7.3 CLOSE_CONNECTION Procedure

This procedure closes an open TCP/IP connection.

Syntax

\[ \text{UTL_TCP.CLOSE_CONNECTION (c IN OUT NOCOPY connection);} \]

Parameters

Table 278-5  CLOSE_CONNECTION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection to close</td>
</tr>
</tbody>
</table>

Usage Notes

Connection must have been opened by a previous call to OPEN_CONNECTION. The fields remote_host, remote_port, local_host, local_port and charset of c are reset after the connection is closed.

An open connection must be closed explicitly. An open connection remains open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.

278.7.4 FLUSH Procedure

This procedure transfers immediately to the server all data in the output buffer, if a buffer is used.

Syntax

\[ \text{UTL_TCP.FLUSH (c IN OUT NOCOPY connection);} \]
Parameters

Table 278-6   FLUSH Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection to which to send data</td>
</tr>
</tbody>
</table>

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION.

278.7.5 GET_LINE Function

This function returns the line of data read.

Syntax

```sql
UTL_TCP.GET_LINE (  
c IN OUT NOCOPY connection,  
remove_crlf IN BOOLEAN DEFAULT FALSE,  
peek IN BOOLEAN DEFAULT FALSE)  
RETURN VARCHAR2;
```

Parameters

Table 278-7   GET_LINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>remove_crlf</td>
<td>If TRUE, then one or more trailing CRLF characters are removed from the received message.</td>
</tr>
<tr>
<td>peek</td>
<td>Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
</tbody>
</table>

Return Values

The text line read

Usage Notes

- The connection must have already been opened through a call to OPEN_CONNECTION.
- See READ_LINE for the read time out, character set conversion, buffer size, and multibyte character issues.
278.7.6 GET_LINE_NCHAR Function

This function returns the line of data read in NCHAR form.

Syntax

```
UTL_TCP.GET_LINE_NCHAR ( 
    c           IN OUT NOCOPY connection, 
    remove_crlf IN            BOOLEAN DEFAULT FALSE, 
    peek        IN            BOOLEAN DEFAULT FALSE) 
RETURN NVARCHAR2;
```

Parameters

Table 278-8  GET_LINE_NCHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>remove_crlf</td>
<td>If TRUE, then one or more trailing CRLF characters are removed from the received message.</td>
</tr>
<tr>
<td>peek</td>
<td>Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
</tbody>
</table>

Return Values

The text line read

Usage Notes

- The connection must have already been opened through a call to `OPEN_CONNECTION`.
- See `READ_LINE` for the read time out, character set conversion, buffer size, and multibyte character issues.

278.7.7 GET_RAW Function

This function returns the data read instead of the amount of data read.

Syntax

```
UTL_TCP.GET_RAW ( 
    c           IN OUT NOCOPY connection, 
    len         IN            PLS_INTEGER DEFAULT 1, 
    peek        IN            BOOLEAN     DEFAULT FALSE) 
RETURN RAW;
```
Parameters

Table 278-9  GET_RAW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.</td>
</tr>
<tr>
<td>peek</td>
<td>Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
<tr>
<td>remove_crlf</td>
<td>If TRUE, then one or more trailing CRLF characters are removed from the received message.</td>
</tr>
</tbody>
</table>

Return Values

The binary data read

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION.

For all the get_* APIs described in this section, see the corresponding READ_* API for the read time out issue. For GET_TEXT and GET_LINE, see the corresponding READ_* API for character set conversion, buffer size, and multibyte character issues.

278.7.8 GET_TEXT Function

This function returns the text data read.

Syntax

```
UTL_TCP.GET_TEXT (  
  c    IN OUT NOCOPY connection,  
  len IN PLS_INTEGER DEFAULT 1,  
  peek IN BOOLEAN DEFAULT FALSE) 
RETURN VARCHAR2;
```

Parameters

Table 278-10  GET_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>len</td>
<td>Number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.</td>
</tr>
</tbody>
</table>
Table 278-10 (Cont.) GET_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peek</td>
<td>Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
<tr>
<td>remove_crlf</td>
<td>If TRUE, then one or more trailing CRLF characters are removed from the received message.</td>
</tr>
</tbody>
</table>

Return Values
The text data read

Usage Notes
The connection must have already been opened through a call to OPEN_CONNECTION.

For all the get_* APIs described in this section, see the corresponding read_* API for the read time out issue. For GET_TEXT and GET_LINE, see the corresponding READ_* API for character set conversion, buffer size, and multibyte character issues.

278.7.9 GET_TEXT_NCHAR Function

This function returns the text data read in NCHAR form.

Syntax

UTL_TCP.GET_TEXT_NCHAR (  
c IN OUT NOCOPY connection,  
len IN PLS_INTEGER DEFAULT 1,  
peek IN BOOLEAN DEFAULT FALSE)  
RETURN NVARCHAR2;

Parameters

Table 278-11 GET_TEXT_NCHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.</td>
</tr>
</tbody>
</table>
Table 278-11 (Cont.) GET_TEXT_NCHAR Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peek</td>
<td>Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
<tr>
<td>remove_crlf</td>
<td>If TRUE, then one or more trailing CRLF characters are removed from the received message.</td>
</tr>
</tbody>
</table>

Return Values

The text data read

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION.

For all the get_* APIs described in this section, see the corresponding read_* API for the read time out issue. For GET_TEXT and GET_LINE, see the corresponding READ_* API for character set conversion, buffer size, and multibyte character issues.

278.7.10 OPEN_CONNECTION Function

This function opens a TCP/IP connection to a specified service.

Syntax

UTL_TCP.OPEN_CONNECTION  (  
remote_host IN VARCHAR2,  
remote_port IN PLS_INTEGER,  
local_host IN VARCHAR2 DEFAULT NULL,  
local_port IN PLS_INTEGER DEFAULT NULL,  
in_buffer_size IN PLS_INTEGER DEFAULT NULL,  
out_buffer_size IN PLS_INTEGER DEFAULT NULL,  
charset IN VARCHAR2 DEFAULT NULL,  
newline IN VARCHAR2 DEFAULT CRLF,  
tx_timeout IN PLS_INTEGER DEFAULT NULL,  
wallet_path IN VARCHAR2 DEFAULT NULL,  
wallet_password IN VARCHAR2 DEFAULT NULL,  
RETURN connection;  
)

Parameters

Table 278-12 OPEN_CONNECTION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_host</td>
<td>Name of the host providing the service. When remote_host is NULL, it connects to the local host.</td>
</tr>
</tbody>
</table>
Table 278-12 (Cont.) OPEN_CONNECTION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_port</td>
<td>Port number on which the service is listening for connections.</td>
</tr>
<tr>
<td>local_host</td>
<td>Name of the host providing the service. NULL means does not care.</td>
</tr>
<tr>
<td>local_port</td>
<td>Port number on which the service is listening for connections. NULL means don’t care.</td>
</tr>
<tr>
<td>in_buffer_size</td>
<td>The size of input buffer. The use of an input buffer can speed up execution performance in receiving data from the server. The appropriate size of the buffer depends on the flow of data between the client and the server, and the traffic/latency on the network. A zero value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the input buffer is 32767 bytes.</td>
</tr>
<tr>
<td>out_buffer_size</td>
<td>The size of output buffer. The use of an output buffer can speed up execution performance in sending data to the server. The appropriate size of buffer depends on the flow of data between the client and the server, and the network condition. A zero value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the output buffer is 32767 bytes.</td>
</tr>
<tr>
<td>charset</td>
<td>The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database are converted to and from the on-the-wire character set as they are sent and received on the network using READ_TEXT, READ_LINE, WRITE_TEXT and WRITE_LINE. Set this parameter to NULL when no conversion is needed.</td>
</tr>
<tr>
<td>newline</td>
<td>Newline character sequence. This newline character sequence is appended to the text line sent by WRITE_LINE API.</td>
</tr>
<tr>
<td>tx_timeout</td>
<td>Time in seconds that the UTL_TCP package should wait before giving up in a read or write operations in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. NULL indicates to wait forever.</td>
</tr>
<tr>
<td>wallet_path</td>
<td>Directory path that contains the Oracle wallet for SSL/TLS. The format is file:directory-path</td>
</tr>
<tr>
<td>wallet_password</td>
<td>Password to open the wallet. When the wallet is auto-login enabled, the password may be set to NULL.</td>
</tr>
</tbody>
</table>

Return Values
A connection to the targeted TCP/IP service

Usage Notes
• Note that connections opened by this UTL_TCP package can remain open and be passed from one database call to another in a shared server configuration. However, the
connection must be closed explicitly. The connection remains open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.

- In the current release of the UTL_TCP package, the parameters `local_host` and `local_port` are ignored when `open_connection` makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The `local_host` and `local_port` fields is not set in the connection record returned by the function.

- `tx_timeout` is intended to govern both the read operations and the write operations. However, an implementation restriction prevents `tx_timeout` from governing write operations in the current release.

Examples

```plsql
DECLARE
    c UTL_TCP.CONNECTION;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION(
        host            => 'www.example.com',
        port            => 443,
        wallet_path     => 'file:/oracle/wallets/smtp_wallet',
        wallet_password => '****');
    UTL_TCP.SECURE_CONNECTION (c => c);
END;
```

278.7.11 READ_LINE Function

This function receives a text line from a service on an open connection.

A line is terminated by a line-feed, a carriage-return or a carriage-return followed by a line-feed.

Syntax

```plsql
UTL_TCP.READ_LINE (  
    c IN OUT NOCOPY connection,  
    data IN OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,  
    peek IN BOOLEAN DEFAULT FALSE)  
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>data</td>
<td>Data received.</td>
</tr>
<tr>
<td>remove_crlf</td>
<td>If <code>TRUE</code>, then one ore more trailing CRLF characters are removed from the received message.</td>
</tr>
</tbody>
</table>
Table 278-13   (Cont.) READ_LINE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peek</td>
<td>Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to \texttt{TRUE} and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
</tbody>
</table>

Return Values

The number of characters of data received

Usage Notes

The connection must have already been opened through a call to \texttt{OPEN\_CONNECTION}. This function does not return until the end-of-line have been reached, or the end of input has been reached. Text messages is converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

If transfer time out is set when the connection is opened, then this function waits for each data packet to be ready to read until time out occurs. If it occurs, then this function stops reading and returns all the data read successfully. If no data is read successfully, then the \texttt{transfer\_timeout} exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, then this function stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, then the \texttt{partial\_multibyte\_char} exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the \texttt{READ\_RAW} function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of the character have not arrived and read time out occurs, then the \texttt{transfer\_timeout} exception is raised instead. The exception can be handled and the read operation can be retried later.

278.7.12 READ_RAW Function

This function receives binary data from a service on an open connection.

Syntax

\begin{verbatim}
UTL_TCP.READ_RAW (c    IN OUT NOCOPY connection,
data  IN OUT NOCOPY RAW,
    len IN            PLS_INTEGER DEFAULT 1,
    peek IN            BOOLEAN     DEFAULT FALSE)
RETURN PLS_INTEGER;
\end{verbatim}
### Parameters

#### Table 278-14 READ_RAW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>data (IN OUT COPY)</td>
<td>Data received</td>
</tr>
<tr>
<td>len</td>
<td>Number of bytes of data to receive</td>
</tr>
<tr>
<td>peek</td>
<td>Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to <code>TRUE</code> and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
</tbody>
</table>

### Return Values

The number of bytes of data received

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`. This function does not return until the specified number of bytes have been read, or the end of input has been reached.

If transfer time out is set when the connection is opened, then this function waits for each data packet to be ready to read until time out occurs. If it occurs, then this function stops reading and returns all the data read successfully. If no data is read successfully, then the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

### 278.7.13 READ_TEXT Function

This function receives text data from a service on an open connection.

#### Syntax

```sql
UTL_TCP.READ_TEXT (  
  c IN OUT NOCOPY connection,  
  data IN OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,  
  len IN            PLS_INTEGER DEFAULT 1,  
  peek IN            BOOLEAN     DEFAULT FALSE)  
RETURN PLS_INTEGER;
```
Parameters

**Table 278-15  READ_TEXT Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
<tr>
<td>data</td>
<td>Data received</td>
</tr>
<tr>
<td>len</td>
<td>Number of characters of data to receive</td>
</tr>
<tr>
<td>peek</td>
<td>Normally, users want to read the data and remove it from the input queue, that is, consume it. In some situations, users may just want to look ahead at the data without removing it from the input queue so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and an input buffer must be set up when the connection is opened. The amount of data that you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.</td>
</tr>
</tbody>
</table>

Return Values

The number of characters of data received

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. This function does not return until the specified number of characters has been read, or the end of input has been reached. Text messages is converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

Unless explicitly overridden, the size of a VARCHAR2 buffer is specified in terms of bytes, while the parameter `len` refers to the maximum number of characters to be read. When the database character set is multibyte, where a single character may consist of more than 1 byte, you should ensure that the buffer can hold the maximum of characters. In general, the size of the VARCHAR2 buffer should equal the number of characters to be read, multiplied by the maximum number of bytes of a character of the database character set.

If transfer time out is set when the connection is opened, then this function waits for each data packet to be ready to read until time out occurs. If it occurs, then this function stops reading and returns all the data read successfully. If no data is read successfully, then the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, then this function stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, then the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `READ_RAW` function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of the character have not arrived and read time out occurs, then the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.
278.7.14 SECURE_CONNECTION Procedure

This procedure secures a TCP/IP connection using SSL/TLS.

SSL/TLS requires an Oracle wallet which must be specified when the connection was opened by the OPEN_CONNECTION Function.

Syntax

```
UTL_TCP.SECURE_CONNECTION (
    c    IN OUT NOCOPY connection);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection from which to receive data</td>
</tr>
</tbody>
</table>

278.7.15 WRITE_LINE Function

This function transmits a text line to a service on an open connection. The newline character sequence is appended to the message before it is transmitted.

Syntax

```
UTL_TCP.WRITE_LINE (
    c    IN OUT NOCOPY connection,
    data IN            VARCHAR2 DEFAULT NULL CHARACTER SET ANY_CS)
RETURN PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection to which to send data</td>
</tr>
<tr>
<td>data</td>
<td>Buffer containing the data to be sent</td>
</tr>
</tbody>
</table>

Return Values

The actual number of characters of data transmitted

Usage Notes

The connection must have already been opened through a call to OPEN_CONNECTION. Text messages are converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.
278.7.16 WRITE_RAW Function

This function transmits a binary message to a service on an open connection. The function does not return until the specified number of bytes have been written.

Syntax

```sql
UTL_TCP.WRITE_RAW (
    c    IN OUT NOCOPY connection,
    data IN             RAW,
    len  IN             PLS_INTEGER DEFAULT NULL)
RETURN PLS_INTEGER;
```

Parameters

Table 278-18  WRITE_RAW Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection to which to send data</td>
</tr>
<tr>
<td>data</td>
<td>Buffer containing the data to be sent</td>
</tr>
<tr>
<td>len</td>
<td>The number of bytes of data to transmit. When <code>len</code> is <code>NULL</code>, the whole length of data is written.</td>
</tr>
</tbody>
</table>

Return Values

The number of bytes of data transmitted

Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`.

278.7.17 WRITE_TEXT Function

This function transmits a text message to a service on an open connection.

Syntax

```sql
UTL_TCP.WRITE_TEXT (
    c    IN OUT NOCOPY connection,
    data IN             VARCHAR2 CHARACTER SET ANY_CS,
    len  IN             PLS_INTEGER DEFAULT NULL)
RETURN num_chars PLS_INTEGER;
```

Parameters

Table 278-19  WRITE_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>TCP connection to which to send data</td>
</tr>
<tr>
<td>data</td>
<td>Buffer containing the data to be sent</td>
</tr>
</tbody>
</table>
Table 278-19  (Cont.) WRITE_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>The number of characters of data to transmit. When len is NULL, the whole length of data is written. The actual amount of data written may be less because of network condition.</td>
</tr>
</tbody>
</table>

**Return Values**

The actual number of characters of data transmitted

**Usage Notes**

The connection must have already been opened through a call to OPEN_CONNECTION. Text messages are converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.
The **UTL_URL** package has two functions: **ESCAPE** and **UNESCAPE**.

This chapter contains the following topics:

- **Overview**
- **Exceptions**
- **Examples**
- **Summary of UTL_URL Subprograms**

See Also:

**UTL_HTTP**

### 279.1 UTL_URL Overview

A Uniform Resource Locator (URL) is a string that identifies a Web resource, such as a page or a picture. Use a URL to access such resources by way of the HyperText Transfer Protocol (HTTP).

For example, the URL for Oracle's Web site is:

```plaintext
http://www.oracle.com
```

Normally, a URL contains English alphabetic characters, digits, and punctuation symbols. These characters are known as the *unreserved characters*. Any other characters in URLs, including multibyte characters or binary octet codes, must be escaped to be accurately processed by Web browsers or Web servers. Some punctuation characters, such as dollar sign ($), question mark (?), colon (:), and equals sign (=), are reserved as delimiters in a URL. They are known as the *reserved characters*. To literally process these characters, instead of treating them as delimiters, they must be escaped.

The unreserved characters are:

- A through Z, a through z, and 0 through 9
- Hyphen (-), underscore (_), period (.), exclamation point (!), tilde (~), asterisk (*), accent (’), left parenthesis (‘), right parenthesis (’)

The reserved characters are:

- Semi-colon (;) slash (/), question mark (?), colon (:), at sign (@), ampersand (&), equals sign (=), plus sign (+), dollar sign ($), percentage sign (%), and comma (,)

The **UTL_URL** package has two functions that provide escape and unescape mechanisms for URL characters. Use the escape function to escape a URL before the URL is used fetch a
Web page by way of the UTL_HTTP package. Use the unescape function to unescape an escaped URL before information is extracted from the URL.

For more information, refer to the Request For Comments (RFC) document RFC2396. Note that this URL escape and unescape mechanism is different from the x-www-form-urlencoded encoding mechanism described in the HTML specification:

http://www.w3.org/TR/html

279.2 UTL_URL Exceptions

UTL_URL raises an exception when it encounter a processing issue.

The following table lists the exceptions that can be raised when the UTL_URL package API is invoked.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Error Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_URL</td>
<td>29262</td>
<td>The URL contains badly formed escape code sequences</td>
</tr>
<tr>
<td>BAD_FIXED_WIDTH_H_CHARSET</td>
<td>29274</td>
<td>Fixed-width multibyte character set is not allowed as a URL character set.</td>
</tr>
</tbody>
</table>

279.3 UTL_URL Examples

UTL_URL can be used for encoding and decoding.

You can implement the x-www-form-urlencoded encoding using the UTL_URL.ESCAPE function as follows:

```sql
CREATE OR REPLACE FUNCTION form_url_encode (data IN VARCHAR2, charset IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  RETURN utl_url.escape(data, TRUE, charset); -- note use of TRUE
END;
```

For decoding data encoded with the form-URL-encode scheme, the following function implements the decoding scheme:

```sql
CREATE OR REPLACE FUNCTION form_url_decode(data IN VARCHAR2, charset IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  RETURN utl_url.unescape(replace(data, '+', ' '), charset);
END;
```

279.4 Summary of UTL_URL Subprograms

This table lists and briefly describes the UTL_URL subprograms.
Table 279-2  UTL_URL Package Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESCAPE Function</td>
<td>Returns a URL with illegal characters (and optionally reserved characters) escaped using the %2-digit-hex-code format</td>
</tr>
<tr>
<td>UNESCAPE Function</td>
<td>Unescapes the escape character sequences to their original forms in a URL. Convert the %XX escape character sequences to the original characters</td>
</tr>
</tbody>
</table>

279.4.1 ESCAPE Function

This function returns a URL with illegal characters (and optionally reserved characters) escaped using the \%2-digit-hex-code format.

Syntax

```sql
UTL_URL.ESCAPE (url IN VARCHAR2 CHARACTER SET ANY_CS,
                 escape_reserved_chars IN BOOLEAN DEFAULT FALSE,
                 url_charset IN VARCHAR2 DEFAULT utl_http.body_charset)
RETURN VARCHAR2;
```

Parameters

Table 279-3  ESCAPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The original URL</td>
</tr>
<tr>
<td>escape_reserved_chars</td>
<td>Indicates whether the URL reserved characters should be escaped. If set to TRUE, both the reserved and illegal URL characters are escaped. Otherwise, only the illegal URL characters are escaped. The default value is FALSE.</td>
</tr>
<tr>
<td>url_charset</td>
<td>When escaping a character (single-byte or multibyte), determine the target character set that character should be converted to before the character is escaped in %hex-code format. If url_charset is NULL, the database charset is assumed and no character set conversion will occur. The default value is the current default body character set of the UTL_HTTP package, whose default value is ISO-8859-1. The character set can be named in Internet Assigned Numbers Authority (IANA) or in the Oracle naming convention.</td>
</tr>
</tbody>
</table>

Usage Notes

Use this function to escape URLs that contain illegal characters as defined in the URL specification RFC 2396. The legal characters in URLs are:

- A through Z, a through z, and 0 through 9
- Hyphen (-), underscore (_), period (.), exclamation point (!), tilde (~), asterisk (*), accent ('), left parenthesis ( ( ), right parenthesis ( ) )

The reserved characters consist of:
• Semi-colon (;), slash (/), question mark (?), colon (:), at sign (@), ampersand (&), equals sign (=), plus sign (+), dollar sign ($), and comma (,)

Many of the reserved characters are used as delimiters in the URL. You should escape characters beyond those listed here by using escape_url. Also, to use the reserved characters in the name-value pairs of the query string of a URL, those characters must be escaped separately. An escape_url cannot recognize the need to escape those characters because once inside a URL, those characters become indistinguishable from the actual delimiters. For example, to pass a name-value pair $logon=HR/<password> into the query string of a URL, escape the $ and / separately as %24logon=HR%2F<password> and use it in the URL.

Normally, you will escape the entire URL, which contains the reserved characters (delimiters) that should not be escaped. For example:

```sql
utl_url.escape('http://www.acme.com/a url with space.html')
```

Returns:

http://www.acme.com/a%20url%20with%20space.html

In other situations, you may want to send a query string with a value that contains reserved characters. In that case, escape only the value fully (with escape_reserved_chars set to TRUE) and then concatenate it with the rest of the URL. For example:

```sql
url := 'http://www.acme.com/search?check=' || utl_url.escape
    ('Is the use of the "" sign okay?', TRUE);
```

This expression escapes the question mark (?), dollar sign ($), and space characters in 'Is the use of the "$" sign okay?' but not the ? after search in the URL that denotes the use of a query string.

The Web server that you intend to fetch Web pages from may use a character set that is different from that of your database. In that case, specify the url_charset as the Web server character set so that the characters that need to be escaped are escaped in the target character set. For example, a user of an EBCDIC database who wants to access an ASCII Web server should escape the URL using US7ASCII so that a space is escaped as %20 (hex code of a space in ASCII) instead of %40 (hex code of a space in EBCDIC).

This function does not validate a URL for the proper URL format.

### 279.4.2 UNESCAPE Function

This function unescapes the escape character sequences to its original form in a URL, to convert the %XX escape character sequences to the original characters.

### Syntax

```sql
UTL_URL.UNESCAPE ( url IN VARCHAR2 CHARACTER SET ANY_CS,
                   url_charset IN VARCHAR2 DEFAULT utl_http.body_charset)
RETURN VARCHAR2;
```

---

**Chapter 279**

**Summary of UTL_URL Subprograms**

---

**Oracle**

279-4
Parameters

Table 279-4  UNESCAPE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL to unescape</td>
</tr>
<tr>
<td>url_charset</td>
<td>After a character is unescaped, the character is assumed to be in the source_charset character set and it will be converted from the source_charset to the database character set before the URL is returned. If source_charset is NULL, the database charset is assumed and no character set conversion occurred. The default value is the current default body character set of the UTL_HTTP package, whose default value is &quot;ISO-8859-1&quot;. The character set can be named in Internet Assigned Numbers Authority (IANA) or Oracle naming convention.</td>
</tr>
</tbody>
</table>

Usage Notes

The Web server that you receive the URL from may use a character set that is different from that of your database. In that case, specify the url_charset as the Web server character set so that the characters that need to be unescaped are unescaped in the source character set. For example, a user of an EBCDIC database who receives a URL from an ASCII Web server should unescape the URL using US7ASCII so that %20 is unescaped as a space (0x20 is the hex code of a space in ASCII) instead of a ? (because 0x20 is not a valid character in EBCDIC).

This function does not validate a URL for the proper URL format.
The WPG_DOCLOAD package provides an interface to download files, BLOBs and BFILES.

See Also:
For more information about implementation of this package:

- Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server
- Oracle Fusion Middleware User's Guide for mod_plsql

The chapter contains the following topics:

- Constants
- Summary of WPG_DOCLOAD Subprograms

280.1 WPG_DOCLOAD Constants

WPG_DOCLOAD defines several constants to use when specifying parameter values.

The WPG_DOCLOAD constants are listed below:

- NAME_COL_LEN
- MIMET_COL_LEN
- MAX_DOCTABLE_NAME_LEN

NAME_COL_LEN
The NAME column in your document table must be the same as the value of name_col_len.

name_col_len CONSTANT pls_integer := 64;

MIMET_COL_LEN
The MIME_TYPE column in your document table must be the same as the value of mimet_col_len.

mimet_col_len CONSTANT pls_integer := 48;

MAX_DOCTABLE_NAME_LEN
The name length of your document table must be less than max_doctable_name_len.

max_doctable_name_len CONSTANT pls_integer := 256;
280.2 Summary of `WPG_DOCLOAD` Subprograms

The `WPG_DOCLOAD` package uses one subprogram, the `DOWNLOAD_FILE` procedure.

**Table 280-1 WPG_DOCLOAD Package Subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DOWNLOAD_FILE Procedures</code></td>
<td>Downloads files, BLOBS and BFILES</td>
</tr>
</tbody>
</table>

**280.2.1 `DOWNLOAD_FILE` Procedures**

There are three versions of this download file procedure.

- The first version downloads files and is invoked from within a document download procedure to signal the PL/SQL Gateway that `p_filename` is to be downloaded from the document table to the client's browser.
- The second version can be called from within any procedure to signal the PL/SQL Gateway that `p_blob` is to be downloaded to the client's browser.
- The third version can be called from within any procedure to signal the PL/SQL Gateway that `p_bfile` is to be downloaded to the client's browser.

**Syntax**

```plaintext
WPG_DOCLOAD.DOWNLOAD_FILE(
    p_filename      IN             VARCHAR2,
    p_bcaching      IN             BOOLEAN DEFAULT TRUE);

WPG_DOCLOAD.DOWNLOAD_FILE(
    p_blob          IN OUT NOCOPY  BLOB);

WPG_DOCLOAD.DOWNLOAD_FILE(
    p_bfile         IN OUT         BFILE);
```

**Parameters**

**Table 280-2 DOWNLOAD_FILE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_filename</td>
<td>The file to download from the document table.</td>
</tr>
<tr>
<td>p_blob</td>
<td>The BLOB to download.</td>
</tr>
<tr>
<td>p_bfile</td>
<td>The BFILE to download (see Usage Notes).</td>
</tr>
<tr>
<td>p_bcaching</td>
<td>Whether browser caching is enabled (see Usage Notes).</td>
</tr>
</tbody>
</table>

**Usage Notes**

- Normally, a document will be downloaded to the browser unless the browser sends an 'If-Modified-Since' header to the gateway indicating that it has the requested document in its cache. In that case, the gateway will determine if the browser's cached copy is up to date, and if it is, it will send an HTTP 304 status
message to the browser indicating that the browser should display the cached copy. However, because a document URL and a document do not necessarily have a one-to-one relationship in the PL/SQL Web Gateway, in some cases it may be undesirable to have the cached copy of a document displayed. In those cases, the `p_bcaching` parameter should be set to FALSE to indicate to the gateway to ignore the 'If-Modified-Since' header, and download the document.

- `p_bfile` and `p_blob` are declared as IN OUT because the locator is initially opened to check for file accessibility and existence. The open operation can only be performed if the locator is writable and readable.
ANYDATA TYPE

An ANYDATA TYPE contains an instance of a given type, plus a description of the type. In this sense, an ANYDATA is self-describing. An ANYDATA can be persistently stored in the database.

This chapter contains the following topics:

- Restrictions
- Operational Notes
- Summary of ANYDATA Subprograms

281.1 ANYDATA TYPE Restrictions

Persistent storage of ANYDATA instances whose type contains embedded LOBs other than BFILES is not currently supported.

281.2 ANYDATA TYPE Operational Notes

This section contains notes related to ANYDATA TYPE construction and access.

Construction

There are 2 ways to construct an ANYDATA. The CONVERT* calls enable construction of the ANYDATA in its entirety with a single call. They serve as explicit CAST functions from any type in the Oracle ORDBMS to ANYDATA.

```sql
STATIC FUNCTION ConvertBDouble(dbl IN BINARY_DOUBLE) return ANYDATA,
STATIC FUNCTION ConvertBfile(b IN BFILE) RETURN ANYDATA,
STATIC FUNCTION ConvertBFloat(f1 IN BINARY_FLOAT) return ANYDATA,
STATIC FUNCTION ConvertBlob(b IN BLOB) RETURN ANYDATA,
STATIC FUNCTION ConvertChar(c IN CHAR) RETURN ANYDATA,
STATIC FUNCTION ConvertClob(c IN CLOB) RETURN ANYDATA,
STATIC FUNCTION ConvertCollection(col IN "collection_type") RETURN ANYDATA,
STATIC FUNCTION ConvertDate(dat IN DATE) RETURN ANYDATA,
STATIC FUNCTION ConvertIntervalDS(inv IN INTERVAL DAY TO SECOND) return ANYDATA,
STATIC FUNCTION ConvertIntervalYM(inv IN INTERVAL YEAR TO MONTH) return ANYDATA,
STATIC FUNCTION ConvertNchar(nc IN NCHAR) return ANYDATA,
STATIC FUNCTION ConvertNClob(nc IN NCLOB) return ANYDATA,
STATIC FUNCTION ConvertNumber(num IN NUMBER) RETURN ANYDATA,
STATIC FUNCTION ConvertNVarchar2(nc IN NVARCHAR2) return ANYDATA,
STATIC FUNCTION ConvertObject(obj IN "<object_type>") RETURN ANYDATA,
STATIC FUNCTION ConvertRaw(r IN RAW) RETURN ANYDATA,
STATIC FUNCTION ConvertRef(rf IN REF "<object_type>") RETURN ANYDATA,
STATIC FUNCTION ConvertTimestamp(ts IN TIMESTAMP) return ANYDATA,
STATIC FUNCTION ConvertTimestampTZ(ts IN TIMESTAMP WITH TIMEZONE) return ANYDATA,
STATIC FUNCTION ConvertTimestampLTZ(ts IN TIMESTAMP WITH LOCAL TIMEZONE) return ANYDATA,
STATIC FUNCTION ConvertUrowid(rid IN UROWID) return ANYDATA,
STATIC FUNCTION ConvertVarchar(c IN VARCHAR) RETURN ANYDATA,
STATIC FUNCTION ConvertVarchar2(c IN VARCHAR2) RETURN ANYDATA,
```
The second way to construct an ANYDATA is a piece by piece approach. The \texttt{BEGINCREATE} Static Procedure call begins the construction process and \texttt{ENDCREATE} Member Procedure call finishes the construction process. In between these two calls, the individual attributes of an object type or the elements of a collection can be set using \texttt{SET*} calls. For piece by piece access of the attributes of objects and elements of collections, the \texttt{PIECEWISE} Member Procedure should be invoked prior to \texttt{GET*} calls.

Note: The ANYDATA has to be constructed or accessed sequentially starting from its first attribute (or collection element). The \texttt{BEGINCREATE} call automatically begins the construction in a piece-wise mode. There is no need to call \texttt{PIECEWISE} immediately after \texttt{BEGINCREATE}. \texttt{ENDCREATE} should be called to finish the construction process (before which any access calls can be made).

Access

Access functions are available based on SQL. These functions do not throw exceptions on type-mismatch. Instead, they return \texttt{NULL} if the type of the ANYDATA does not correspond to the type of access. If you wish to use only ANYDATA functions of the appropriate types returned in a query, you should use a \texttt{WHERE} clause which uses \texttt{GETTYPENAME} and choose the type you are interested in (say "SYS.NUMBER"). Each of these functions returns the value of a specified datatype inside a \texttt{SYS.ANYDATA} wrapper.

\begin{verbatim}
MEMBER FUNCTION AccessBDouble(self IN ANYDATA) return BINARY_DOUBLE
  DETERMINISTIC,
MEMBER FUNCTION AccessBfile(self IN ANYDATA) return BFILE,
MEMBER FUNCTION AccessBFloat(self IN ANYDATA) return BINARY_FLOAT
  DETERMINISTIC,
MEMBER FUNCTION AccessBlob(self IN ANYDATA) return BLOB,
MEMBER FUNCTION AccessChar(self IN ANYDATA) return CHAR,
MEMBER FUNCTION AccessClob(self IN ANYDATA) return CLOB,
MEMBER FUNCTION AccessDate(self IN ANYDATA) return DATE,
MEMBER FUNCTION AccessIntervalYM(self IN ANYDATA) return INTERVAL YEAR TO MONTH,
MEMBER FUNCTION AccessIntervalDS(self IN ANYDATA) return INTERVAL DAY TO SECOND,
MEMBER FUNCTION AccessNchar(self IN ANYDATA) return NCHAR,
MEMBER FUNCTION AccessNClob(self IN ANYDATA) return NCLOB
MEMBER FUNCTION AccessNumber(self IN ANYDATA) return NUMBER,
MEMBER FUNCTION AccessNVarchar2(self IN ANYDATA) return NVARCHAR2,
MEMBER FUNCTION AccessRaw(self IN ANYDATA) return RAW,
MEMBER FUNCTION AccessTimestamp(self IN ANYDATA) return TIMESTAMP,
MEMBER FUNCTION AccessTimestampLTZ(self IN ANYDATA) return TIMESTAMP WITH LOCAL
  TIMEZONE,
MEMBER FUNCTION AccessTimestampTZ(self IN ANYDATA) return TIMESTAMP WITH
  TIMEZONE,
MEMBER FUNCTION AccessURowid(self IN ANYDATA) return UROWID DETERMINISTIC
MEMBER FUNCTION AccessVarchar(self IN ANYDATA) return VARCHAR,
MEMBER FUNCTION AccessVarchar2(self IN ANYDATA) return VARCHAR2,
\end{verbatim}

281.3 Summary of ANYDATA Subprograms

This table lists the ANYDATA subprograms in alphabetical order and briefly describes them.
Table 281-1  ANYDATA Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGINCREATE Static Procedure</td>
<td>Begins creation process on a new ANYDATA</td>
</tr>
<tr>
<td>ENDCREATE Member Procedure</td>
<td>Ends creation of an ANYDATA</td>
</tr>
<tr>
<td>GET* Member Functions</td>
<td>Gets the current data value (which should be of appropriate type)</td>
</tr>
<tr>
<td>GETTYPE Member Function</td>
<td>Gets the Type of the ANYDATA</td>
</tr>
<tr>
<td>GETTYPENAME Member Function</td>
<td>Get the fully qualified type name for the ANYDATA</td>
</tr>
<tr>
<td>PIECEWISE Member Procedure</td>
<td>Sets the MODE of access of the current data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT)</td>
</tr>
<tr>
<td>SET* Member Procedures</td>
<td>Sets the current data value.</td>
</tr>
</tbody>
</table>

281.3.1 BEGINCREATE Static Procedure

This procedure begins the creation process on a new ANYDATA.

Syntax

```
STATIC PROCEDURE BeginCreate(
    dtype          IN OUT NOCOPY AnyType,
    adata          OUT NOCOPY ANYDATA);
```

Parameters

Table 281-2  BEGINCREATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtype</td>
<td>The type of the ANYDATA. (Should correspond to OCI_TYPECODE_OBJECT or a Collection typecode.)</td>
</tr>
<tr>
<td>adata</td>
<td>ANYDATA being constructed.</td>
</tr>
</tbody>
</table>

Exception

DBMS_TYPES.INVALID_PARAMETERS: dtype is invalid (not fully constructed, and similar deficits.)

Usage Notes

There is no need to call PIECEWISE immediately after this call. The construction process begins in a piece-wise manner automatically.
281.3.2 ENDCREATE Member Procedure

This procedure ends creation of an ANYDATA. Other creation functions cannot be called after this call.

Syntax

```
MEMBER PROCEDURE EndCreate(
    self         IN OUT NOCOPY ANYDATA);
```

Parameters

**Table 281-3  ENDCREATE Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An ANYDATA.</td>
</tr>
</tbody>
</table>

281.3.3 GET* Member Functions

These functions get the current data value (which should be of appropriate type).

The type of the current data value depends on the MODE by which it is accessed (depending on whether the PIECEWISE call is invoked).

If PIECEWISE has NOT been called, the ANYDATA is accessed in its entirety and the type of the data value should match the type of the ANYDATA.

If PIECEWISE has been called, the ANYDATA is accessed piece-wise. The type of the data value should match the type of the attribute (or collection element) at the current position.

Syntax

```
MEMBER FUNCTION GetBDouble(
    self         IN ANYDATA,
    dbl          OUT NOCOPY BINARY_DOUBLE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetBfile(
    self         IN ANYDATA,
    b            OUT NOCOPY BFILE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetBFloat(
    self         IN ANYDATA,
    fl           OUT NOCOPY BINARY_FLOAT)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetBlob(
    self         IN ANYDATA,
    b            OUT NOCOPY BLOB)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetChar(
    self         IN ANYDATA,
    c            OUT NOCOPY CHAR)
MEMBER FUNCTION GetClob(
    self IN ANYDATA,
    c OUT NOCOPY CLOB)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetCollection(
    self IN ANYDATA,
    col OUT NOCOPY "<collection_type>")
RETURN PLS_INTEGER;

MEMBER FUNCTION GetDate(
    self IN ANYDATA,
    dat OUT NOCOPY DATE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetIntervalDS(
    self IN ANYDATA,
    inv OUT NOCOPY INTERVAL DAY TO SECOND)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetIntervalYM(
    self IN ANYDATA,
    inv OUT NOCOPY INTERVAL YEAR TO MONTH)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetNchar(
    self IN ANYDATA,
    nc OUT NOCOPY NCHAR)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetNClob(
    self IN ANYDATA,
    nc OUT NOCOPY NCLOB)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetNumber(
    self IN ANYDATA,
    num OUT NOCOPY NUMBER)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetNVarchar2(
    self IN ANYDATA,
    nc OUT NOCOPY NVARCHAR2)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetObject(
    self IN ANYDATA,
    obj OUT NOCOPY "<object_type>")
RETURN PLS_INTEGER;

MEMBER FUNCTION GetRaw(
    self IN ANYDATA,
    r OUT NOCOPY RAW)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetRef(
    self IN ANYDATA,
    rf OUT NOCOPY REF "<object_type>")
RETURN PLS_INTEGER;
MEMBER FUNCTION GetTimestamp(
    self  IN ANYDATA,
    ts   OUT NOCOPY TIMESTAMP)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetTimestampTZ(
    self  IN ANYDATA,
    ts   OUT NOCOPY TIMESTAMP WITH TIME ZONE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetTimestampLTZ(
    self  IN ANYDATA,
    ts   OUT NOCOPY TIMESTAMP WITH LOCAL TIME ZONE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetVarchar(
    self  IN ANYDATA,
    c    OUT NOCOPY VARCHAR)
RETURN       PLS_INTEGER;

MEMBER FUNCTION GetVarchar2(
    self  IN ANYDATA,
    c    OUT NOCOPY VARCHAR2)
RETURN       PLS_INTEGER;

Parameters

Table 281-4   GET* Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An ANYDATA.</td>
</tr>
<tr>
<td>num</td>
<td>The number to be obtained.</td>
</tr>
</tbody>
</table>

Return Values

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

The return value is relevant only if PIECEWISE has been already called (for a collection). In such a case, DBMS_TYPES.NO_DATA signifies the end of the collection when all elements have been accessed.

Exceptions

DBMS_TYPES.TYPE_MISMATCH: When the expected type is different from the passed in type.

DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

DBMS_TYPES.INCORRECT_USAGE: Incorrect usage.
281.3.4 GETTYPE Member Function

This function gets the typecode of the ANYDATA.

Syntax

```
MEMBER FUNCTION GETTYPE(
    self          IN ANYDATA,
    typ           OUT NOCOPY AnyType)
RETURN        PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An ANYDATA.</td>
</tr>
<tr>
<td>typ</td>
<td>The AnyType corresponding to the ANYDATA. May be NULL if it does not represent a user-defined type.</td>
</tr>
</tbody>
</table>

Return Values

The typecode corresponding to the type of the ANYDATA.

281.3.5 GETTYPENAME Member Function

This function gets the fully qualified type name for the ANYDATA.

If the ANYDATA is based on a built-in type, this function will return NUMBER and other relevant information.

If it is based on a user defined type, this function will return `schema_name.type_name`, for example, SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

Syntax

```
MEMBER FUNCTION GETTYPENAME(
    self         IN ANYDATA)
RETURN       VARCHAR2;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An ANYDATA.</td>
</tr>
</tbody>
</table>

Return Values

Type name of the ANYDATA.
281.3.6 PIECEWISE Member Procedure

This procedure sets the MODE of access of the current data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).

It sets the MODE of access of the data value to be a collection element at a time (if the data value is of collection type). Once this call has been made, subsequent calls to SET* and GET* will sequentially obtain individual attributes or collection elements.

Syntax

MEMBER PROCEDURE PIECEWISE(
    self         IN OUT NOCOPY ANYDATA);

Parameters

Table 281-7 PIECEWISE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The current data value.</td>
</tr>
</tbody>
</table>

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS
- DBMS_TYPES.INCORRECT_USAGE: On incorrect usage.

Usage Notes

The current data value must be of an OBJECT or COLLECTION type before this call can be made.

Piece-wise construction and access of nested attributes that are of object or collection types is not supported.

281.3.7 SET* Member Procedures

This procedure sets the current data value.

This is a list of procedures that should be called depending on the type of the current data value. The type of the data value should be the type of the attribute at the current position during the piece-wise construction process.

Syntax

MEMBER PROCEDURE SETBDOUBLE(
    self         IN OUT NOCOPY ANYDATA,
    dbl          IN BINARY_DOUBLE,
    last_elem    IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBFILE(
    self         IN OUT NOCOPY ANYDATA,
    b            IN BFILE,
    last_elem    IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBFLOAT(
    self         IN OUT NOCOPY ANYDATA,
self        IN OUT NOCOPY ANYDATA,
f1 IN         BINARY_FLOAT,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBLOB(
self        IN OUT NOCOPY ANYDATA,
b IN BLOB,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCHAR(
self        IN OUT NOCOPY ANYDATA,
c IN CHAR,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCLOB(
self        IN OUT NOCOPY ANYDATA,
c IN CLOB,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCOLLECTION(
self        IN OUT NOCOPY ANYDATA,
col IN "<collection_type>",
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETDATE(
self        IN OUT NOCOPY ANYDATA,
dat IN DATE,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALDS(
self        IN OUT NOCOPY ANYDATA,
inv IN INTERVAL DAY TO SECOND,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALYM(
self        IN OUT NOCOPY ANYDATA,
inv IN INTERVAL YEAR TO MONTH,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETNCHAR(
self        IN OUT NOCOPY ANYDATA,
nch IN NCHAR,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETNCLOB(
self        IN OUT NOCOPY ANYDATA,
nch IN NClob,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETNUMBER(
self        IN OUT NOCOPY ANYDATA,
um IN NUMBER,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETNVARCHAR2(
self        IN OUT NOCOPY ANYDATA,
nch IN NVarchar2,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETOBJECT(
self        IN OUT NOCOPY ANYDATA,
obj IN "<object_type>",
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETRAW(
self IN OUT NOCOPY ANYDATA,
r IN RAW,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETREF(
self IN OUT NOCOPY ANYDATA,
rf IN REF "<object_type>",
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMP(
self IN OUT NOCOPY ANYDATA,
ts IN TIMESTAMP,
last_elem IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPZ(
sel IN OUT NOCOPY ANYDATA,
ts IN TIMESTAMP WITH TIME ZONE,
last_elem IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPTZ(
sel IN OUT NOCOPY ANYDATA,
ts IN TIMESTAMP WITH LOCAL TIME ZONE,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETVARCHAR(
self IN OUT NOCOPY ANYDATA,
c IN VARCHAR,
last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETVARCHAR2(
self IN OUT NOCOPY ANYDATA,
c IN VARCHAR2,
last_elem IN boolean DEFAULT FALSE);

Parameters

Table 281-8  SET* Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>An ANYDATA.</td>
</tr>
<tr>
<td>num</td>
<td>The number, and associated information, that is to be set.</td>
</tr>
<tr>
<td>last_elem</td>
<td>Relevant only if ANYDATA represents a collection. Set to TRUE if it is the last element of the collection, FALSE otherwise.</td>
</tr>
</tbody>
</table>

Exceptions

- **DBMS_TYPES.INVALID_PARAMETERS**: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).
- **DBMS_TYPES.INCORRECT_USAGE**: Incorrect usage.
- **DBMS_TYPES.TYPE_MISMATCH**: When the expected type is different from the passed in type.
Usage Notes

When `BEGINCREATE` is called, construction has already begun in a piece-wise fashion. Subsequent calls to `SET*` will set the successive attribute values.

If the `ANYDATA` is a standalone collection, the `SET*` call will set the successive collection elements.
282

ANYDATASET TYPE

An ANYDATASET TYPE contains a description of a given type plus a set of data instances of that type. An ANYDATASET can be persistently stored in the database if desired, or it can be used as interface parameters to communicate self-descriptive sets of data, all of which belong to a certain type.

This chapter contains the following topics:

- Construction
- Summary of ANYDATASET TYPE Subprograms

282.1 ANYDATASET TYPE Construction

The ANYDATASET needs to be constructed value by value, sequentially.

For each data instance (of the type of the ANYDATASET), the ADDINSTANCE function must be invoked. This adds a new data instance to the ANYDATASET. Subsequently, SET* can be called to set each value in its entirety.

The MODE of construction/access can be changed to attribute/collection element wise by making calls to PIECEWISE.

- If the type of the ANYDATASET is TYPECODE_OBJECT, individual attributes will be set with subsequent SET* calls. Likewise on access.
- If the type of the current data value is a collection type individual collection elements will be set with subsequent SET* calls. Likewise on access. This call is very similar to ANYDATA.PIECEWISE call defined for the type ANYDATA.

Note that there is no support for piece-wise construction and access of nested (not top level) attributes that are of object types or collection types.

ENDCREATE should be called to finish the construction process (before which no access calls can be made).

282.2 Summary of ANYDATASET TYPE Subprograms

This table lists the ANYDATASET Type subprograms and briefly describes them.

<table>
<thead>
<tr>
<th>Table 282-1</th>
<th>ANYDATASET Type Subprograms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subprogram</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>ADDINSTANCE Member Procedure</td>
<td>Adds a new data instance to an ANYDATASET.</td>
</tr>
<tr>
<td>BEGINCREATE Static Procedure</td>
<td>Creates a new ANYDATASET which can be used to create a set of data values of the given ANYTYPE.</td>
</tr>
</tbody>
</table>
Table 282-1  (Cont.) ANYDATASET Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENDCREATE Member Procedure</td>
<td>Ends Creation of a ANYDATASET. Other creation functions cannot be called after this call.</td>
</tr>
<tr>
<td>GET* Member Functions</td>
<td>Gets the current data value (which should be of appropriate type).</td>
</tr>
<tr>
<td>GETCOUNT Member Function</td>
<td>Gets the number of data instances in an ANYDATASET.</td>
</tr>
<tr>
<td>GETINSTANCE Member Function</td>
<td>Gets the next instance in an ANYDATASET.</td>
</tr>
<tr>
<td>GETTYPE Member Function</td>
<td>Gets the ANYTYPE describing the type of the data instances in an ANYDATASET.</td>
</tr>
<tr>
<td>GETTYPENAME Member Function</td>
<td>Gets the AnyType describing the type of the data instances in an ANYDATASET.</td>
</tr>
<tr>
<td>PIECEWISE Member Procedure</td>
<td>Sets the MODE of construction, access of the data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).</td>
</tr>
<tr>
<td>SET* Member Procedures</td>
<td>Sets the current data value.</td>
</tr>
</tbody>
</table>

282.2.1 ADDINSTANCE Member Procedure

This procedure adds a new data instance to an ANYDATASET.

Syntax

MEMBER PROCEDURE AddInstance(
    self          IN OUT NOCOPY ANYDATASET);

Parameters

Table 282-2  ADDINSTANCE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being constructed.</td>
</tr>
</tbody>
</table>

Exceptions

DBMS_TYPES.invalid_parameters: Invalid parameters.
DBMS_TYPES.incorrect_usage: On incorrect usage.

Usage Notes

The data instances have to be added sequentially. The previous data instance must be fully constructed (or set to NULL) before a new one can be added.

This call DOES NOT automatically set the mode of construction to be piece-wise. The user has to explicitly call PIECEWISE if a piece-wise construction of the instance is intended.
282.2.2 BEGINCREATE Static Procedure

This procedure creates a new ANYDATASET which can be used to create a set of data values of the given ANYTYPE.

Syntax

```plsql
STATIC PROCEDURE BeginCreate(
    typecode     IN PLS_INTEGER,
    rtype        IN OUT NOCOPY AnyType,
    aset         OUT NOCOPY ANYDATASET);
```

Parameters

Table 282-3  BEGINCREATE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typecode</td>
<td>The typecode for the type of the ANYDATASET.</td>
</tr>
<tr>
<td>dtype</td>
<td>The type of the data values. This parameter is a must for user-defined types like TYPECODE_OBJECT, Collection typecodes, and similar others.</td>
</tr>
<tr>
<td>aset</td>
<td>The ANYDATASET being constructed.</td>
</tr>
</tbody>
</table>

Exceptions

DBMS_TYPES.invalid_parameters: dtype is invalid (not fully constructed, and like errors.)

282.2.3 ENDCREATE Member Procedure

This procedure ends creation of a ANYDATASET. Other creation functions cannot be called after this call.

Syntax

```plsql
MEMBER PROCEDURE ENDCREATE(
    self              IN OUT NOCOPY ANYDATASET);
```

Parameters

Table 282-4  ENDCREATE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being constructed.</td>
</tr>
</tbody>
</table>

282.2.4 GET* Member Functions

These functions get the current data value (which should be of the appropriate type).

The type of the current data value depends on the MODE used for accessing it (depending on how the PIECEWISE call is invoked). If PIECEWISE has not been called, the instance is
accessed in its entirety, and the type of the data value should match the type of the
ANYDATASET.

If PIECEWISE has been called, the instance is accessed piece-wise. The type of the
data value should match the type of the attribute (or collection element) at the current
position.

Syntax

MEMBER FUNCTION GETBDOUBLE(
    self IN ANYDATASET,
    dbl OUT NOCOPY BINARY_DOUBLE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETBFLOAT(
    self IN ANYDATASET,
    fl OUT NOCOPY BINARY_FLOAT)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETBFILE(
    self IN ANYDATASET,
    b OUT NOCOPY BFILE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETBLOB(
    self IN ANYDATASET,
    b OUT NOCOPY BLOB)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETCHAR(
    self IN ANYDATASET,
    c OUT NOCOPY CHAR)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETCLOB(
    self IN ANYDATASET,
    c OUT NOCOPY CLOB)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETCOLLECTION(
    self IN ANYDATASET,
    col OUT NOCOPY "<collection_type>")
RETURN PLS_INTEGER;

MEMBER FUNCTION GETDATE(
    self IN ANYDATASET,
    dat OUT NOCOPY DATE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETINTERVALDS(
    self IN ANYDATASET,
    inv IN OUT NOCOPY INTERVAL DAY TO SECOND)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETINTERVALYM(
    self IN ANYDATASET,
    inv IN OUT NOCOPY INTERVAL YEAR TO MONTH)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETNCHAR(
    self IN ANYDATASET,
MEMBER FUNCTION GETNCLOB(
    self IN ANYDATASET,
    nc OUT NOCOPY NCLOB)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETNUMBER(
    self IN ANYDATASET,
    num OUT NOCOPY NUMBER)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETNVARCHAR2(
    self IN ANYDATASET,
    nc OUT NOCOPY NVARCHAR2)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETOBJECT(
    self IN ANYDATASET,
    obj OUT NOCOPY "<object_type>")
RETURN PLS_INTEGER;

MEMBER FUNCTION GETRAW(
    self IN ANYDATASET,
    r OUT NOCOPY RAW)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETREF(
    self IN ANYDATASET,
    rf OUT NOCOPY REF "<object_type>")
RETURN PLS_INTEGER;

MEMBER FUNCTION GETTIMESTAMP(
    self IN ANYDATASET,
RETURN PLS_INTEGER;

MEMBER FUNCTION GETTIMESTAMPLTZ(
    self IN ANYDATASET,
    ts OUT NOCOPY TIMESTAMP WITH LOCAL TIME ZONE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETTIMESTAMPZ(
    self IN ANYDATASET,
    ts OUT NOCOPY TIMESTAMP WITH TIME ZONE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETUROWID(
    self IN ANYDATASET,
    rid OUT NOCOPY UROWID)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETVARCHAR(
    self IN ANYDATASET,
    c OUT NOCOPY VARCHAR)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETVARCHAR2(
    self IN ANYDATASET,
    c OUT NOCOPY VARCHAR2)
RETURN PLS_INTEGER;
Parameters

Table 282-5  GET* Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being accessed.</td>
</tr>
<tr>
<td>num</td>
<td>The number, and associated information, that is to be obtained.</td>
</tr>
</tbody>
</table>

Return Values

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

The return value is relevant only if PIECEWISE has been already called (for a collection). In such a case, DBMS_TYPES.NO_DATA signifies the end of the collection when all elements have been accessed.

Exceptions

DBMS_TYPES.INVALID_PARAMETER: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

DBMS_TYPES.INCORRECT_USAGE: Incorrect usage

DBMS_TYPES.TYPE_MISMATCH: When the expected type is different from the passed in type.

282.2.5 GETCOUNT Member Function

This function gets the number of data instances in an ANYDATASET.

Syntax

MEMBER FUNCTION GetCount (  
    self IN ANYDATASET)  
RETURN PLS_INTEGER;

Parameter

Table 282-6  GETCOUNT Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being accessed.</td>
</tr>
</tbody>
</table>

Return Values

The number of data instances.
282.2.6 GETINSTANCE Member Function

This function gets the next instance in an ANYDATASET. Only sequential access to the instances in an ANYDATASET is allowed.

After this function has been called, the GET* functions can be invoked on the ANYDATASET to access the current instance. If PIECEWISE is called before doing the GET* calls, the individual attributes (or collection elements) can be accessed.

It is an error to invoke this function before the ANYDATASET is fully created.

Syntax

MEMBER FUNCTION GETINSTANCE(
    self           IN OUT NOCOPY ANYDATASET)
RETURN         PLS_INTEGER;

Parameters

Table 282-7 GETINSTANCE Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being accessed.</td>
</tr>
</tbody>
</table>

Return Values

DBMS_TYPES.SUCCESS or DBMS_TYPES.NO_DATA

DBMS_TYPES.NO_DATA signifies the end of the ANYDATASET (all instances have been accessed).

Usage Notes

This function should be called even before accessing the first instance.

282.2.7 GETTYPE Member Function

This function gets the AnyType describing the type of the data instances in an ANYDATASET.

Syntax

MEMBER FUNCTION GETTYPE(
    self           IN ANYDATASET,
    typ            OUT NOCOPY AnyType)
RETURN         PLS_INTEGER;

Parameters

Table 282-8 GETTYPE Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td></td>
<td>The ANYDATASET.</td>
</tr>
<tr>
<td>typ</td>
<td></td>
<td>The ANYTYPE corresponding to the AnyData. May be NULL if it does not represent a user-defined function.</td>
</tr>
</tbody>
</table>
Return Values
The typecode corresponding to the type of the ANYDATASET.

282.2.8 GETTYPENAME Member Function

This procedure gets the fully qualified type name for the ANYDATASET.

If the ANYDATASET is based on a built-in, this function will return NUMBER and associated information.

If it is based on a user defined type, this function will return schema_name.type_name. For example, SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

Syntax

MEMBER FUNCTION GETTYPENAME(
    self IN ANYDATASET)
RETURN VARCHAR2;

Parameter

Table 282-9   GETTYPENAME Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being constructed.</td>
</tr>
</tbody>
</table>

Return Values
Type name of the ANYDATASET.

282.2.9 PIECEWISE Member Procedure

This procedure sets the MODE of construction, access of the data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).

It sets the MODE of construction, access of the data value to be a collection element at a time (if the data value is of a collection TYPE). Once this call has been made, subsequent SET* and GET* calls will sequentially obtain individual attributes or collection elements.

Syntax

MEMBER PROCEDURE PIECEWISE(
    self IN OUT NOCOPY ANYDATASET);
Parameters

Table 282-10  PIECEWISE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being constructed.</td>
</tr>
</tbody>
</table>

Exceptions

DBMS_TYPES.INVALID_PARAMETERS: Invalid parameters.

DBMS_TYPES.INCORRECT_USAGE: On incorrect usage.

Usage Notes

The current data value must be of an object or collection type before this call can be made. There is no support for piece-wise construction or access of embedded object type attributes or nested collections.

282.2.10 SET* Member Procedures

This procedure sets the current data value.

The type of the current data value depends on the mode with which we are constructing (depending on how we have invoked the PIECEWISE call). The type of the current data should be the type of the ANYDATASET if PIECEWISE has NOT been called. The type should be the type of the attribute at the current position if PIECEWISE has been called.

Syntax

MEMBER PROCEDURE SETBDOUBLE(
    self          IN OUT NOCOPY ANYDATASET,
    dbl           IN BINARY_DOUBLE,
    last_elem     IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETBFLOAT(
    self          IN OUT NOCOPY ANYDATASET,
    fl            IN BINARY_FLOAT,
    last_elem     IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETBFILE(
    self          IN OUT NOCOPY ANYDATASET,
    b             IN BFILE,
    last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETBLOB(
    self          IN OUT NOCOPY ANYDATASET,
    b             IN BLOB,
    last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETCHAR(
    self          IN OUT NOCOPY ANYDATASET,
    c             IN CHAR,
    last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETCLOB(
member procedure setcollection(
    self  in out nocopy anydataset,
    col   in "collection_type",
    last_elem boolean default false);

member procedure setdate(
    self  in out nocopy anydataset,
    dat   in date,
    last_elem boolean default false);

member procedure setintervalds(
    self  in out nocopy anydataset,
    inv   in interval day to second,
    last_elem boolean default false);

member procedure setintervalym(
    self  in out nocopy anydataset,
    inv   in interval year to month,
    last_elem boolean default false);

member procedure setnchar(
    self  in out nocopy anydataset,
    nc    in nchar,
    last_elem boolean default false);

member procedure setnclob(
    self  in out nocopy anydataset,
    nc    in nclob,
    last_elem boolean default false);

member procedure setnumber(
    self  in out nocopy anydataset,
    num   in number,
    last_elem boolean default false);

member procedure setnvarchar2(
    self  in out nocopy anydataset,
    nc    in nvarchar2,
    last_elem boolean default false);

member procedure setobject(
    self  in out nocopy anydataset,
    obj   in "object_type",
    last_elem boolean default false);

member procedure setraw(
    self  in out nocopy anydataset,
    r     in raw,
    last_elem boolean default false);

member procedure setref(
    self  in out nocopy anydataset,
    rf    in ref "object_type",
    last_elem boolean default false);

member procedure settimestamp(
    self  in out nocopy anydataset,
MEMBER PROCEDURE SETTIMESTAMPALTZ(
    self IN OUT NOCOPY ANYDATASET,
    ts IN TIMESTAMP WITH LOCAL TIME ZONE,
    last_elem IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPALTZ(
    self IN OUT NOCOPY ANYDATASET,
    ts IN TIMESTAMP WITH TIME ZONE,
    last_elem IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETUROWID(
    self IN OUT NOCOPY ANYDATASET,
    rid IN UROWID,
    last_elem IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETVARCHAR(
    self IN OUT NOCOPY ANYDATASET,
    c IN VARCHAR,
    last_elem BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETVARCHAR2(
    self IN OUT NOCOPY ANYDATASET,
    c IN VARCHAR2,
    last_elem BOOLEAN DEFAULT FALSE);

Parameters

Table 282-11 SET* Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYDATASET being accessed.</td>
</tr>
<tr>
<td>num</td>
<td>The number, and associated information, that is to be set.</td>
</tr>
<tr>
<td>last_elem</td>
<td>Relevant only if PIECEWISE has been already called (for a collection). Set to TRUE if it is the last element of the collection, FALSE otherwise.</td>
</tr>
</tbody>
</table>

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS: Invalid parameters (if it is not appropriate to add a number at this point in the creation process).
- DBMS_TYPES.INCORRECT_USAGE: Incorrect usage.
- DBMS_TYPES.TYPE_MISMATCH: When the expected type is different from the passed in type.
An ANYTYPE can contain a type description of any persistent SQL type, named or unnamed, including object types and collection types. It can also be used to construct new transient type descriptions.

New persistent types can only be created using the CREATE TYPE statement. Only new transient types can be constructed using the ANYTYPE interfaces.

This chapter discusses the following:

- Summary of ANYTYPE Subprograms

### 283.1 Summary of ANYTYPE Subprograms

This table lists the ANYTYPE subprograms in alphabetical order and briefly describes them.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGINCREATE Static Procedure</td>
<td>Creates a new instance of ANYTYPE which can be used to create a transient type description.</td>
</tr>
<tr>
<td>SETINFO Member Procedure</td>
<td>Sets any additional information required for constructing a COLLECTION or builtin type.</td>
</tr>
<tr>
<td>ADDATTR Member Procedure</td>
<td>Adds an attribute to an ANYTYPE (of typecode DBMS_TYPES.TYPENCODE_OBJECT).</td>
</tr>
<tr>
<td>ENDCREATE Member Procedure</td>
<td>Ends creation of a transient ANYTYPE. Other creation functions cannot be called after this call.</td>
</tr>
<tr>
<td>GETPERSISTENT Static Function</td>
<td>Returns an ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.</td>
</tr>
<tr>
<td>GETINFO Member Function</td>
<td>Gets the type information for the ANYTYPE.</td>
</tr>
<tr>
<td>GETATTRELEMINFOMember Function</td>
<td>Gets the type information for an attribute of the type (if it is of TYPENCODE_OBJECT). Gets the type information for a collection's element type if the self parameter is of a collection type.</td>
</tr>
</tbody>
</table>

#### 283.1.1 BEGINCREATE Static Procedure

This procedure creates a new instance of ANYTYPE which can be used to create a transient type description.

**Syntax**

```plaintext
STATIC PROCEDURE BEGINCREATE(
    typeid IN PLS_INTEGER,
    atype OUT NOCOPY ANYTYPE);
```
Parameters

Table 283-2  BEGIN CREATE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typecode</td>
<td>Use a constant from DBMS_TYPES package. Typecodes for user-defined type: DBMS_TYPES.TYPECODE_OBJECT, DBMS_TYPES.TYPECODE_VARRAY or DBMS_TYPES.TYPECODE_TABLE. Typecodes for builtin types: DBMS_TYPES.TYPECODE_NUMBER, and similar types.</td>
</tr>
<tr>
<td>atype</td>
<td>ANYTYPE for a transient type</td>
</tr>
</tbody>
</table>

283.1.2 SETINFO Member Procedure

This procedure sets any additional information required for constructing a COLLECTION or builtin type.

Syntax

MEMBER Procedure SETINFO(
    self       IN OUT NOCOPY ANYTYPE,
    prec       IN PLS_INTEGER,
    scale      IN PLS_INTEGER,
    len        IN PLS_INTEGER,
    csid       IN PLS_INTEGER,
    csfrm      IN PLS_INTEGER,
    atype      IN ANYTYPE DEFAULT NULL,
    elem_tc    IN PLS_INTEGER DEFAULT NULL,
    elem_count IN PLS_INTEGER DEFAULT 0);

Parameters

Table 283-3  SETINFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The transient ANYTYPE that is being constructed.</td>
</tr>
<tr>
<td>prec</td>
<td>Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>len</td>
<td>Optional. Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.</td>
</tr>
<tr>
<td>csid</td>
<td>Required if typecode represents types requiring character information such as CHAR, VARCHAR, or VARCHAR2.</td>
</tr>
<tr>
<td>csfrm</td>
<td>Required if typecode represents types requiring character information such as CHAR, VARCHAR, or VARCHAR2.</td>
</tr>
</tbody>
</table>
Table 283-3  (Cont.) SETINFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atype</td>
<td>Optional. Required if collection element typecode is a user-defined type such as TYPECODE_OBJECT, and similar others. It is also required for a built-in type that needs user-defined type information such as TYPECODE_REF. This parameter is not needed otherwise.</td>
</tr>
</tbody>
</table>

The Following Parameters Are Required For Collection Types

Table 283-4  SETINFO Procedure Parameters - Collection Types

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem_tc</td>
<td>Must be of the collection element's typecode (from DBMS_TYPES package).</td>
</tr>
<tr>
<td>elem_count</td>
<td>Pass 0 for elem_count if the self represents a nested table (TYPECODE_TABLE). Otherwise pass the collection count if self represents a VARRAY.</td>
</tr>
</tbody>
</table>

Exceptions

- DBMS_TYPES.INVALID_PARAMETER: Invalid Parameters (typecode, typeinfo)
- DBMS_TYPES.INCORRECT_USAGE: Incorrect usage (cannot call after calling ENDCREATE, and similar actions.)

Usage Notes

It is an error to call this function on an ANYTYPE that represents a persistent user defined type.

### 283.1.3 ADDATTR Member Procedure

This procedure adds an attribute to an ANYTYPE (of typecode DBMS_TYPES.TYPECODE_OBJECT).

Syntax

```sql
MEMBER PROCEDURE ADDATTR(
    self          IN OUT NOCOPY ANYTYPE,
    aname         IN VARCHAR2,
    typecode      IN PLS_INTEGER,
    prec          IN PLS_INTEGER,
    scale         IN PLS_INTEGER,
    len           IN PLS_INTEGER,
    csid          IN PLS_INTEGER,
    csfrm         IN PLS_INTEGER,
    attr_type     IN ANYTYPE DEFAULT NULL);
```
Parameters

Table 283-5  ADDATTR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The transient ANYTYPE that is being constructed. Must be of type DBMS_TYPES.TYPECODE_OBJECT.</td>
</tr>
<tr>
<td>aname</td>
<td>Optional. Attribute's name. Could be NULL.</td>
</tr>
<tr>
<td>typecode</td>
<td>Attribute's typecode. Can be built-in or user-defined typecode (from DBMS_TYPES package).</td>
</tr>
<tr>
<td>prec</td>
<td>Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>len</td>
<td>Optional. Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Give length.</td>
</tr>
<tr>
<td>csid</td>
<td>Optional. Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, or VARCHAR2.</td>
</tr>
<tr>
<td>csfrm</td>
<td>Optional. Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, or VARCHAR2.</td>
</tr>
<tr>
<td>attr_type</td>
<td>Optional. ANYTYPE corresponding to a user-defined type. This parameter is required if the attribute is a user defined type.</td>
</tr>
</tbody>
</table>

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (typecode, typeinfo)
- DBMS_TYPES.INCORRECT_USAGE: Incorrect usage (cannot call after calling EndCreate, and similar actions.)

283.1.4 ENDCREATE Member Procedure

This procedure ends creation of a transient ANYTYPE. Other creation functions cannot be called after this call.

Syntax

MEMBER PROCEDURE ENDCREATE(
    self           IN OUT NOCOPY ANYTYPE);

Parameter

Table 283-6  ENDCREATE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The transient ANYTYPE that is being constructed.</td>
</tr>
</tbody>
</table>
283.1.5 GETPERSISTENT Static Function

This procedure returns an ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

Syntax

```
STATIC FUNCTION GETPERSISTENT(
    schema_name      IN VARCHAR2,
    type_name        IN VARCHAR2,
    version          IN VARCHAR2 DEFAULT NULL)
RETURN           ANYTYPE;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>Schema name of the type.</td>
</tr>
<tr>
<td>type_name</td>
<td>Type name.</td>
</tr>
<tr>
<td>version</td>
<td>Type version.</td>
</tr>
</tbody>
</table>

Return Values

An ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

283.1.6 GETINFO Member Function

This function gets the type information for the ANYTYPE.

Syntax

```
MEMBER FUNCTION GETINFO ( 
    self        IN ANYTYPE,
    prec        OUT PLS_INTEGER,
    scale       OUT PLS_INTEGER,
    len         OUT PLS_INTEGER,
    csid        OUT PLS_INTEGER,
    csfrm       OUT PLS_INTEGER,
    schema_name OUT VARCHAR2,
    type_name   OUT VARCHAR2,
    version     OUT VARCHAR2,
    numelems    OUT PLS_INTEGER)
RETURN      PLS_INTEGER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYTYPE.</td>
</tr>
</tbody>
</table>
Table 283-8    (Cont.) GETINFO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prec</td>
<td>If typecode represents a number. Gives precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>scale</td>
<td>If typecode represents a number. Gives precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>len</td>
<td>If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.</td>
</tr>
<tr>
<td>csid</td>
<td>If typecode represents a type requiring character information such as: CHAR, VARCHAR, or VARCHAR2.</td>
</tr>
<tr>
<td>schema_name</td>
<td>Type's schema (if persistent).</td>
</tr>
<tr>
<td>type_name</td>
<td>Type's typename.</td>
</tr>
<tr>
<td>version</td>
<td>Type's version.</td>
</tr>
<tr>
<td>numelems</td>
<td>If self is a TYPECODE_VARRAY, this gives the VARRAY count. If self is of TYPECODE_OBJECT, this gives the number of attributes.</td>
</tr>
</tbody>
</table>

Return Values

The typecode of self.

Exceptions

- DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (position is beyond bounds or the ANYTYPE is not properly Constructed).

283.1.7 GETATTRELEMINFO Member Function

This function gets the type information for an attribute of the type (if it is of TYPECODE_OBJECT). Gets the type information for a collection’s element type if the self parameter is of a collection type.

Syntax

MEMBER FUNCTION GETATTRELEMINFO (  
  self            IN ANYTYPE,  
  pos             IN PLS_INTEGER,  
  prec            OUT PLS_INTEGER,  
  scale           OUT PLS_INTEGER,  
  len             OUT PLS_INTEGER,  
  csid            OUT PLS_INTEGER,  
  csfrm           OUT PLS_INTEGER,  
  attr_elt_type   OUT ANYTYPE  
  aname           OUT VARCHAR2)  
RETURN         PLS_INTEGER;
Parameters

Table 283-9  GETTRELEMINFO Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>The ANYTYPE.</td>
</tr>
<tr>
<td>pos</td>
<td>If self is of TYPECODE_OBJECT, this gives the attribute position (starting at 1). It is ignored otherwise.</td>
</tr>
<tr>
<td>prec</td>
<td>If attribute/collection element typecode represents a NUMBER. Gives precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>scale</td>
<td>If attribute/collection element typecode represents a NUMBER. Gives precision and scale. Ignored otherwise.</td>
</tr>
<tr>
<td>len</td>
<td>If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.</td>
</tr>
<tr>
<td>csid, csfrm</td>
<td>If typecode represents a type requiring character information such as: CHAR, VARCHAR, or VARCHAR2. Gives character set ID, character set form.</td>
</tr>
<tr>
<td>attr_elt_type</td>
<td>If attribute/collection element typecode represents a user-defined type, this returns the ANYTYPE corresponding to it. User can subsequently describe the attr_elt_type.</td>
</tr>
<tr>
<td>aname</td>
<td>Attribute name (if it is an attribute of an object type, NULL otherwise).</td>
</tr>
</tbody>
</table>

Return Values

The typecode of the attribute or collection element.

Exceptions

DBMS_TYPES.INVALID_PARAMETERS: Invalid Parameters (position is beyond bounds or the ANYTYPE is not properly constructed).
This chapter describes the types used with Oracle Database Advanced Queuing (AQ) packages for PL/SQL, DBMS_AQ, and DBMS_AQADM.

This chapter contains the following topics:

- Security Model
- Summary of Types

See Also:


284.1 Advanced Queuing (AQ) Types — Security Model

PUBLIC is granted EXECUTE privilege on the types described in this chapter.

284.2 Oracle Database Advanced Queuing (AQ) Types — Summary of Types

Oracle Database Advanced Queuing (AQ) uses types to specify certain types of information.

- AQ$_AGENT Type
- AQ$_AGENT_LIST_T Type
- AQ$_DESCRIPTOR Type
- AQ$_NTFN_DESCRIPTOR Type
- AQ$_NTFN_MSGID_ARRAY Type
- AQ$_POST_INFO Type
- AQ$_POST_INFO_LIST Type
- AQ$_PURGE_OPTIONS_T Type
- AQ$_RECIPIENT_LIST_T Type
- AQ$_REG_INFO Type
- AQ$_REG_INFO_LIST Type
- AQ$_SUBSCRIBER_LIST_T Type
- DEQUEUE_OPTIONS_T Type
284.2.1 AQ$_AGENT Type

This type identifies a producer or a consumer of a message.

Syntax

```
TYPE SYS.AQ$_AGENT IS OBJECT (
    name       VARCHAR2(512),
    address    VARCHAR2(1024),
    protocol   NUMBER);
```

Attributes

**Table 284-1  AQ$_AGENT Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of a producer or consumer of a message. The name must follow object name guidelines in the <em>Oracle Database SQL Language Reference</em> with regard to reserved characters.</td>
</tr>
<tr>
<td>address</td>
<td>Protocol-specific address of the recipient. If the protocol is 0, then the address is of the form <code>[schema.]queue[@dblink]</code>. For example, a queue named <em>emp_messages</em> in the <em>HR</em> queue at the site <em>dbs1.net</em> has the address: <em><a href="mailto:hr.emp_messages@dbs1.net">hr.emp_messages@dbs1.net</a></em></td>
</tr>
<tr>
<td>protocol</td>
<td>Protocol to interpret the address and propagate the message. Protocols 1-127 are reserved for internal use. If the protocol number is in the range 128 - 255, the address of the recipient is not interpreted by Oracle Database Advanced Queuing.</td>
</tr>
</tbody>
</table>

284.2.2 AQ$_AGENT_LIST_T Type

This type identifies the list of agents for which `DBMS_AQ.LISTEN` listens.

```
TYPE SYS.AQ$_AGENT_LIST_T IS TABLE OF SYS.AQ$_AGENT
    INDEX BY BINARY INTEGER;
```

See Also:

"AQ$_AGENT Type"
284.2.3 AQ$_.DESCRIPTOR Type

This type specifies the Oracle Database Advanced Queuing descriptor received by the AQ PL/SQL callbacks upon notification.

See Also:

"MESSAGE_PROPERTIES_T Type"

Syntax

```
TYPE SYS.AQ$_DESCRIPTION IS OBJECT (  
  queue_name      VARCHAR2(261),  
  consumer_name   VARCHAR2(512),  
  msg_id          RAW(16),  
  msg_prop        MSG_PROP_T,  
  gen_desc        AQ$_NTFN_DESCRIPTOR,  
  msgid_array     SYS.AQ$_NTFN_MSGID_ARRAY,  
  ntfnsRecdInGrp  NUMBER);
```

Attributes

Table 284-2  AQ$_DESCRIPTION Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Name of the queue in which the message was enqueued which resulted in the notification</td>
</tr>
<tr>
<td>consumer_name</td>
<td>Name of the consumer for the multiconsumer queue</td>
</tr>
<tr>
<td>msg_id</td>
<td>Identification number of the message</td>
</tr>
<tr>
<td>msg_prop</td>
<td>Message properties specified by the MSG_PROP_T type</td>
</tr>
<tr>
<td>gen_desc</td>
<td>Indicates the timeout specifications</td>
</tr>
<tr>
<td>msgid_array</td>
<td>Group notification message ID list</td>
</tr>
<tr>
<td>ntfnsRecdInGrp</td>
<td>Notifications received in group</td>
</tr>
</tbody>
</table>

284.2.4 AQ$_NTFN_DESCRIPTOR Type

This type is for storing a generic notification descriptor regarding PL/SQL notification flags.

Syntax

```
TYPE SYS.AQ$_NTFN_DESCRIPTOR IS OBJECT(   ntfn_flags   NUMBER);
```
Attributes

Table 284-3  AQ$_NTFN_DESCRIPTOR Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntfn_flags</td>
<td>Set to 1 if the notifications are already removed after a stipulated timeout. Set 2 to denote grouping. Default is 0.</td>
</tr>
</tbody>
</table>

284.2.5 AQ$_NTFN_MSGID_ARRAY Type

This type is for storing grouping notification data for AQ namespace, value $2^{30}$ which is the max varray size.

Syntax

```
TYPE SYS.AQ$_NTFN_MSGID_ARRAY AS VARRAY(1073741824)OF RAW(16);
```

284.2.6 AQ$_POST_INFO Type

This type specifies anonymous subscriptions to which you want to post messages.

Syntax

```
TYPE SYS.AQ$_POST_INFO IS OBJECT (  
  name       VARCHAR2(512),  
  namespace  NUMBER,  
  payload    RAW(32767));
```

Attributes

Table 284-4  AQ$_POST_INFO Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the anonymous subscription to which you want to post</td>
</tr>
<tr>
<td>namespace</td>
<td>To receive notifications from other applications through DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS</td>
</tr>
<tr>
<td>payload</td>
<td>The payload to be posted to the anonymous subscription</td>
</tr>
</tbody>
</table>
284.2.7 AQ$_POST_INFO_LIST Type

This type identifies the list of anonymous subscriptions to which you want to post messages.

See Also:
AQ$_POST_INFO Type

Syntax

TYPE SYS.AQ$_POST_INFO_LIST AS VARRAY(1024) OF SYS.AQ$_POST_INFO;

284.2.8 AQ$_PURGE_OPTIONS_T Type

This type specifies the options available for purging a queue table.

See Also:
PURGE_QUEUE_TABLE Procedure.

Syntax

TYPE AQ$_PURGE_OPTIONS_T is RECORD (  
  block           BOOLEAN       DEFAULT FALSE  
  delivery_mode   PLS_INTEGER   DEFAULT PERSISTENT);

Table 284-5 AQ$_PURGE_OPTIONS_T Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>TRUE/FALSE. If block is TRUE, then an exclusive lock on all the queues in the queue table is held while purging the queue table. This will cause concurrent enqueuers and dequeuers to block while the queue table is purged. The purge call always succeeds if block is TRUE. The default for block is FALSE. This will not block enqueuers and dequeuers, but it can cause the purge to fail with an error during high concurrency times.</td>
</tr>
<tr>
<td>delivery_mode</td>
<td>Kind of messages to purge, either DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT</td>
</tr>
</tbody>
</table>
284.2.9 AQ$_RECIPIENT_LIST_T Type

Identifies the list of agents that receive the message. This type can be used only when the queue is enabled for multiple dequeses.

See Also:

"AQ$_AGENT Type"

Syntax

TYPE SYS.AQ$_RECIPIENT_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY_INTEGER;

284.2.10 AQ$_REG_INFO Type

This type identifies a producer or a consumer of a message.

Syntax

TYPE SYS.AQ$_REG_INFO IS OBJECT {
   name                              VARCHAR2(512),
   namespace                         NUMBER,
   callback                          VARCHAR2(4000),
   context                           RAW(2000)  DEFAULT NULL,
   anyctx                            ANYDATA,
   ctxtype                           NUMBER,
   payloadcbk                        VARCHAR2(4000),
   timeout                           NUMBER,
   ntfn_grouping_class               NUMBER,
   ntfn_grouping_value               NUMBER    DEFAULT 600,
   ntfn_grouping_type                NUMBER,
   ntfn_grouping_start_time          TIMESTAMP(6) WITH TIME ZONE,
   ntfn_grouping_repeat_count        NUMBER);

Attributes

Table 284-6   AQ$_REG_INFO Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the name of the subscription. The subscription name is of the form schema.queue if the registration is for a single consumer queue or schema.queue:consumer_name if the registration is for a multiconsumer queues.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>namespace</td>
<td>Specifies the namespace of the subscription. To receive notification from Oracle Database Advanced Queuing queues, the namespace must be DBMS_AQ_NAMESPACE_AQ. To receive notifications from other applications through DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ_NAMESPACE_ANONYMOUS.</td>
</tr>
<tr>
<td>callback</td>
<td>Specifies the action to be performed on message notification. For HTTP notifications, use <a href="http://www.company.com:8080">http://www.company.com:8080</a>. For e-mail notifications, use mailto://xyz@company.com. For raw message payload for the PLSQLCALLBACK procedure, use plsql://schema.procedure?PR=0. For user-defined type message payload converted to XML for the PLSQLCALLBACK procedure, use plsql://schema.procedure?PR=1.</td>
</tr>
<tr>
<td>context</td>
<td>Specifies the context that is to be passed to the callback function.</td>
</tr>
<tr>
<td>anyctx</td>
<td>Specifies the ANYDATA type context that is passed to the callback function.</td>
</tr>
<tr>
<td>ctxtype</td>
<td>Specifies the context type. Valid values are 0 (RAW) or 1 (ANYDATA).</td>
</tr>
<tr>
<td>payloadcbk</td>
<td>Specifies payload for the callback function.</td>
</tr>
<tr>
<td>timeout</td>
<td>Specifies the time in seconds.</td>
</tr>
<tr>
<td>ntnf_grouping_class</td>
<td>Currently, only the following flag can be set to specify criterion for grouping. The default value will be 0. If ntnf_grouping_class is 0, all other notification grouping attributes must be 0.</td>
</tr>
<tr>
<td></td>
<td>• NTFN_GROUPING_CLASS_TIME - Notifications grouped by time, that is, the user specifies a time value and a single notification gets published at the end of that time.</td>
</tr>
<tr>
<td>ntnf_grouping_value</td>
<td>Time-period of grouping notifications specified in seconds, meaning the time after which grouping notification would be sent periodically until ntnf_grouping_repeat_count is exhausted.</td>
</tr>
<tr>
<td>ntnf_grouping_type</td>
<td>• NTFN_GROUPING_TYPE_SUMMARY - Summary of all notifications that occurred in the time interval. (Default)</td>
</tr>
<tr>
<td></td>
<td>• NTFN_GROUPING_TYPE_LAST - Last notification that occurred in the interval.</td>
</tr>
</tbody>
</table>
Table 284-6  (Cont.) AQ$_REG_INFO Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntfn_grouping_start_time</td>
<td>Notification grouping start time. Notification grouping can start from a user-specified time that should a valid timestamp with time zone. If ntfn_grouping_start_time is not specified when using grouping, the default is to current timestamp with time zone</td>
</tr>
<tr>
<td>ntfn_grouping_repeat_count</td>
<td>Grouping notifications will be sent as many times as specified by the notification grouping repeat count and after that revert to regular notifications. The ntfn_grouping_repeat_count, if not specified, will default to <code>NTFN_GROUPING_FOREVER</code> - Keep sending grouping notifications forever.</td>
</tr>
</tbody>
</table>

Usage Notes
You can use the following notification mechanisms:

- OCI callback
- e-mail callback
- PL/SQL callback

Table 284-7 shows the actions performed for nonpersistent queues for different notification mechanisms when RAW presentation is specified. Table 284-8 shows the actions performed when XML presentation is specified.

Table 284-7  Actions Performed for Nonpersistent Queues When RAW Presentation Specified

<table>
<thead>
<tr>
<th>Queue Payload Type</th>
<th>oci Callback</th>
<th>E-mail</th>
<th>PL/SQL Callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>OCI callback receives the RAW data in the payload.</td>
<td>Not supported</td>
<td>PL/SQL callback receives the RAW data in the payload.</td>
</tr>
<tr>
<td>Oracle object type</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Table 284-8  Actions Performed for Nonpersistent Queues When XML Presentation Specified

<table>
<thead>
<tr>
<th>Queue Payload Type</th>
<th>OCI Callback</th>
<th>E-mail</th>
<th>PL/SQL Callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>OCI callback receives the XML data in the payload.</td>
<td>XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.</td>
<td>PL/SQL callback receives the XML data in the payload.</td>
</tr>
<tr>
<td>Oracle object type</td>
<td>OCI callback receives the XML data in the payload.</td>
<td>XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.</td>
<td>PL/SQL callback receives the XML data in the payload.</td>
</tr>
</tbody>
</table>
284.2.11 AQ$_REG_INFO_LIST Type

Identifies the list of registrations to a queue.

See Also:

"AQ$_REG_INFO Type"

Syntax

TYPE SYS.AQ$_REG_INFO_LIST AS VARRAY(1024) OF SYS.AQ$_REG_INFO;

284.2.12 AQ$_SUBSCRIBER_LIST_T Type

This type identifies the list of subscribers that subscribe to a queue.

See Also:

"AQ$_AGENT Type"

Syntax

TYPE SYS.AQ$_SUBSCRIBER_LIST_T IS TABLE OF SYS.AQ$_AGENT INDEX BY BINARY_INTEGER;

284.2.13 DEQUEUE_OPTIONS_T Type

DEQUEUE_OPTIONS_T specifies the options available for the dequeue operation.

Syntax

TYPE DEQUEUE_OPTIONS_T IS RECORD (  
consumer_name VARCHAR2(30) DEFAULT NULL,  
dequeue_mode BINARY_INTEGER DEFAULT REMOVE,  
navigation BINARY_INTEGER DEFAULT NEXT_MESSAGE,  
visibility BINARY_INTEGER DEFAULT ON_COMMIT,  
wait BINARY_INTEGER DEFAULT FOREVER,  
msgid RAW(16) DEFAULT NULL,  
correlation VARCHAR2(128) DEFAULT NULL,  
deq_condition VARCHAR2(4000) DEFAULT NULL,  
signature aq$_sig_prop DEFAULT NULL,  
thansformation VARCHAR2(61) DEFAULT NULL,  
delivery_mode PLS_INTEGER DEFAULT PERSISTENT);
### Attributes

#### Table 284-9 DEQUEUE_OPTIONS_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer_name</td>
<td>Name of the consumer. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers, then this field should be set to NULL. [For secure queues, consumer_name must be a valid AQ agent name, mapped to the database user performing the dequeue operation, through dbms_aqadm.enable_db_access procedure call.]</td>
</tr>
</tbody>
</table>
Table 284-9  (Cont.) DEQUEUE_OPTIONS_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visibility</td>
<td>Specifies whether the new message is dequeued as part of the current transaction. The visibility parameter is ignored when using the BROWSE dequeue mode. Possible settings are: ON_COMMIT: The dequeue will be part of the current transaction. This setting is the default. IMMEDIATE: The dequeue operation is not part of the current transaction, but an autonomous transaction which commits at the end of the operation.</td>
</tr>
<tr>
<td>wait</td>
<td>Specifies the wait time if there is currently no message available which matches the search criteria. Possible settings are: FOREVER: Wait forever. This setting is the default. NO_WAIT: Do not wait. NUMBER: Wait time in seconds.</td>
</tr>
<tr>
<td>msgid</td>
<td>Specifies the message identifier of the message to be dequeued.</td>
</tr>
<tr>
<td>correlation</td>
<td>Specifies the correlation identifier of the message to be dequeued. Special pattern matching characters, such as the percent sign (%) and the underscore (_) can be used. If more than one message satisfies the pattern, then the order of dequeuing is undetermined.</td>
</tr>
<tr>
<td>deq_condition</td>
<td>A conditional expression based on the message properties, the message data properties, and PL/SQL functions. A deq_condition is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the WHERE clause of a SQL query). Message properties include priority, corrid and other columns in the queue table. To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload. The deq_condition parameter cannot exceed 4000 characters. If more than one message satisfies the deq_condition, then the order of dequeuing is undetermined.</td>
</tr>
<tr>
<td>signature</td>
<td>Currently not implemented</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies a transformation that will be applied after dequeuing the message. The source type of the transformation must match the type of the queue.</td>
</tr>
<tr>
<td>delivery_mode</td>
<td>The deqer specifies the delivery mode of the messages it wishes to dequeue in the dequeue options. It can be BUFFERED or PERSISTENT or PERSISTENT_OR_BUFFERED. The message properties of the dequeued message indicate the delivery mode of the dequeued message. Array dequeue is only supported for buffered messages with an array size of ‘1’.</td>
</tr>
</tbody>
</table>
284.2.14 ENQUEUE_OPTIONS_T Type

ENQUEUE_OPTIONS_T Type specifies the options available for the enqueue operation.

Syntax

TYPE SYS.ENQUEUE_OPTIONS_T IS RECORD (
    visibility BINARY_INTEGER DEFAULT ON_COMMIT,
    relative_msgid RAW(16) DEFAULT NULL,
    sequence_deviation BINARY_INTEGER DEFAULT NULL,
    transformation VARCHAR2(61) DEFAULT NULL,
    delivery_mode PLS_INTEGER NOT NULL DEFAULT PERSISTENT);

Attributes

Table 284-10 ENQUEUE_OPTIONS_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visibility</td>
<td>Specifies the transactional behavior of the enqueue request. Possible settings are:</td>
</tr>
<tr>
<td></td>
<td>ON_COMMIT: The enqueue is part of the current transaction. The operation is complete when the transaction commits. This setting is the default.</td>
</tr>
<tr>
<td></td>
<td>IMMEDIATE: The enqueue operation is not part of the current transaction, but an autonomous transaction which commits at the end of the operation. This is the only value allowed when enqueuing to a non-persistent queue.</td>
</tr>
<tr>
<td>relative_msgid</td>
<td>Specifies the message identifier of the message which is referenced in the sequence deviation operation. This field is valid only if BEFORE is specified in sequence_deviation. This parameter is ignored if sequence deviation is not specified.</td>
</tr>
<tr>
<td>sequence_deviation</td>
<td>Specifies whether the message being enqueued should be dequeued before other messages already in the queue. Possible settings are:</td>
</tr>
<tr>
<td></td>
<td>BEFORE: The message is enqueued ahead of the message specified by relative_msgid.</td>
</tr>
<tr>
<td></td>
<td>TOP: The message is enqueued ahead of any other messages.</td>
</tr>
<tr>
<td>transformation</td>
<td>Specifies a transformation that will be applied before enqueuing the message. The return type of the transformation function must match the type of the queue.</td>
</tr>
<tr>
<td>delivery_mode</td>
<td>The enqueuer specifies the delivery mode of the messages it wishes to enqueue in the enqueue options. It can be BUFFERED or PERSISTENT. The message properties of the enqueued message indicate the delivery mode of the enqueued message. Array enqueue is only supported for buffered messages with an array size of '1'.</td>
</tr>
</tbody>
</table>
284.2.15 QUEUE_PROPS_T Type

This type specifies the Oracle Database Advanced Queuing descriptor received by the AQ PL/SQL callbacks upon notification.

Syntax

```
TYPE QUEUE_PROPS_T IS RECORD {
  retry_delay     NUMBER         DEFAULT 0,
  retention_time  NUMBER         DEFAULT 0,
  sort_list       VARCHAR2(30)   DEFAULT NULL,
  cache_hint      BINARY_INTEGER DEFAULT AUTO
};
```

Attributes

**Table 284-11  QUEUE_PROPS_T Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retry_delay</td>
<td>Specifies the number of seconds after which this message is scheduled for</td>
</tr>
<tr>
<td></td>
<td>processing again after an application rollback</td>
</tr>
<tr>
<td>retention_time</td>
<td>Specifies the number of seconds a message is retained in the queue table after</td>
</tr>
<tr>
<td></td>
<td>being dequeued from the queue</td>
</tr>
<tr>
<td>sort_list</td>
<td>Sharded queues can be sorted by ENQ_TIME or PRIORITY, ENQ_TIME.</td>
</tr>
<tr>
<td>cache_hint</td>
<td>Specifies a hint to sharded queue whether to cache messages or not. User</td>
</tr>
<tr>
<td></td>
<td>can specify following:</td>
</tr>
<tr>
<td></td>
<td>• UNCACHED: Message cache will not cache messages</td>
</tr>
<tr>
<td></td>
<td>• CACHED: Message cache will cache messages and give preference to those</td>
</tr>
<tr>
<td></td>
<td>queues with CACHED hint.</td>
</tr>
<tr>
<td></td>
<td>• AUTO: Message cache does best effort based on available memory</td>
</tr>
</tbody>
</table>

284.2.16 SYS.MSG_PROP_T Type

This type is used in PL/SQL notification, as one field in aq$ descriptor, to pass message properties of an AQ message to the PL/SQL notification client callback.

Syntax

```
CREATE or replace TYPE sys.msg_prop_t AS OBJECT {
  priority         NUMBER,
  delay            NUMBER,
  expiration       NUMBER,
  correlation      VARCHAR2(128),
  attempts         NUMBER,
  exception_queue  VARCHAR2(51),
  enqueue_time     DATE,
  state            NUMBER,
  sender_id        aq$_agent,
  original_msgid   RAW(16),
  delivery_mode    NUMBER
};
```
Parameters

Table 284-12  SYS.MSG_PROP_T Type Attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.</td>
</tr>
<tr>
<td>delay</td>
<td>Specifies the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. Dequeuing by msgid overrides the delay specification. A message enqueued with delay set is in the WAITING state, and when the delay expires, the message goes to the READY state. DELAY processing requires the queue monitor to be started. However the queue monitor is started automatically by the system if needed. Delay is set by the producer who enqueues the message. The possible settings follow:</td>
</tr>
<tr>
<td></td>
<td>NO_DELAY: The message is available for immediate dequeuing</td>
</tr>
<tr>
<td></td>
<td>number: The number of seconds to delay the message</td>
</tr>
<tr>
<td>expiration</td>
<td>Specifies the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the time the message is ready for dequeue. Expiration processing requires the queue monitor to be running. However the queue monitor is started automatically by the system if needed. The possible settings follow:</td>
</tr>
<tr>
<td></td>
<td>NEVER: The message does not expire</td>
</tr>
<tr>
<td></td>
<td>number: The number of seconds message remains in READY state. If the message is not dequeued before it expires, then it is moved to the exception queue in the EXPIRED state.</td>
</tr>
<tr>
<td>correlation</td>
<td>Returns the identifier supplied by the producer of the message at enqueue time.</td>
</tr>
<tr>
<td>attempts</td>
<td>Returns the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.</td>
</tr>
</tbody>
</table>
Table 284-12 (Cont.) SYS.MSG_PROP_T Type Attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exception_queue</td>
<td>Specifies the name of the queue into which the message is moved if it cannot be processed successfully. Messages are moved automatically into the exception queue. Messages are moved into the exception queue in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• RETRY_COUNT, the number of unsuccessful dequeue attempts, has exceeded the specification for the MAX_RETRIES parameter in the DBMS_AQADM.CREATE_QUEUE procedure during queue creation. For multiconsumer queues, the message becomes eligible to be moved to the exception queue even if failed dequeue attempts exceed the MAX_RETRIES parameter for only one of the consumers. But the message will not be moved until either all other consumers have successfully consumed the message or failed more than MAX_RETRIES. You can view MAX_RETRIES for a queue in the ALL_QUEUES data dictionary view. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then RETRY_COUNT is not incremented. • A message was not dequeued before the expiration time elapsed. • Message propagation to the specified destination queue failed with one of the following errors:</td>
</tr>
<tr>
<td></td>
<td>* There were no recipients for the multiconsumer destination queue. * Recipients were specified for a single-consumer destination queue. * Destination queue was an exception queue. * There was an error when applying transformation. The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log. If the default exception queue is specified, then the parameter returns a NULL value at dequeue time.</td>
</tr>
<tr>
<td>enqueue_time</td>
<td>Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user at enqueue time.</td>
</tr>
</tbody>
</table>
Table 284-12  (Cont.) SYS.MSG_PROP_T Type Attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| state     | Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. The possible states follow:  
  - DBMS_AQ.READY: The message is ready to be processed.  
  - DBMS_AQ.WAITING: The message delay has not yet been reached.  
  - DBMS_AQ.PROCESSED: The message has been processed and is retained.  
  - DBMS.AQ.EXPIRED: The message has been moved to the exception queue. |
| sender_id | The application-sender identification specified at enqueue time by the message producer. Sender id is of type aq$_agent. Sender name is required for secure queues at enqueue time. This must be a valid AQ agent name, mapped to the database user performing the enqueue operation, through dbms_aqadm.enable_db_access procedure call. Sender address and protocol should not be specified. The Sender id in the message properties returned at dequeue time may have a sender address if the message was propagated from another queue. The value of the address is the source_queue, source database name if it was a remote database [format source_queue@source_database_name] |
| original_msgid | This parameter is used by Oracle Database Advanced Queuing for propagating messages. |
| delivery_mode | DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT. |

284.2.17 MESSAGE_PROPERTIES_T Type

This type is defined inside the DBMS_AQ package, and describes the information that AQ uses to convey the state of individual messages. These are set at enqueue time, and their values are returned at dequeue time.

See Also:

AQ$_RECIPIENT_LIST_T Type

Syntax

```
TYPE message_properties_t IS RECORD (  
  priority        BINARY_INTEGER  NOT NULL DEFAULT 1,  
  delay           BINARY_INTEGER  NOT NULL DEFAULT NO_DELAY,  
  expiration      BINARY_INTEGER  NOT NULL DEFAULT NEVER,  
  correlation     VARCHAR2(128)   DEFAULT NULL,  
  attempts        BINARY_INTEGER,  
  recipient_list  AQ$_RECIPIENT_LIST_T,  
  exception_queue VARCHAR2(61)   DEFAULT NULL,  
  enqueue_time    VARCHAR2(61)   DEFAULT NULL,  
  state           BINARY_INTEGER,  
)  
```

Chapter 284

Oracle Database Advanced Queuing (AQ) Types —Summary of Types

284-16
Attributes

Table 284-13  MESSAGE_PROPERTIES_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.</td>
</tr>
<tr>
<td>delay</td>
<td>Specifies the delay of the enqueued message. The delay represents the number of seconds after which a message is available for de dequeuing. De queuing by msgid overrides the delay specification. A message enqueued with delay set is in the WAITING state, and when the delay expires, the message goes to the READY state. Del ay processing requires the queue monitor to be started. However the queue monitor is started automatically by the system if needed. Delay is set by the producer who enqueues the message. The possible settings follow:</td>
</tr>
<tr>
<td>expiration</td>
<td>Specifies the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the time the message is ready for dequeue. Expiration processing requires the queue monitor to be running. However the queue monitor is started automatically by the system if needed. The possible settings follow:</td>
</tr>
<tr>
<td>correlation</td>
<td>Returns the identifier supplied by the producer of the message at enqueue time.</td>
</tr>
<tr>
<td>attempts</td>
<td>Returns the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.</td>
</tr>
<tr>
<td>recipient_list</td>
<td>This parameter is only valid for queues that allow multiple consumers. The default recipients are the queue subscribers. This parameter is not returned to a consumer at dequeue time. For type definition, see the &quot;AQ$AGENT Type&quot;.</td>
</tr>
</tbody>
</table>
Table 284-13  (Cont.) MESSAGE_PROPERTIES_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exception_queue</td>
<td>Specifies the name of the queue into which the message is moved if it cannot be processed successfully. Messages are moved automatically into the exception queue. Messages are moved into the exception queue in the following cases:</td>
</tr>
<tr>
<td></td>
<td>· RETRY_COUNT, the number of unsuccessful dequeue attempts, has exceeded the specification for the MAX_RETRIES parameter in the DBMS_AQADM.CREATE_QUEUE procedure during queue creation. For multiconsumer queues, the message becomes eligible to be moved to the exception queue even if failed dequeue attempts exceed the MAX_RETRIES parameter for only one of the consumers. But the message will not be moved until either all other consumers have successfully consumed the message or failed more than MAX_RETRIES. You can view MAX_RETRIES for a queue in the ALL_QUEUES data dictionary view.</td>
</tr>
<tr>
<td></td>
<td>· A message was not dequeued before the expiration time elapsed.</td>
</tr>
<tr>
<td></td>
<td>· Message propagation to the specified destination queue failed with one of the following errors:</td>
</tr>
<tr>
<td></td>
<td>· There were no recipients for the multiconsumer destination queue.</td>
</tr>
<tr>
<td></td>
<td>· Recipients were specified for a single-consumer destination queue.</td>
</tr>
<tr>
<td></td>
<td>· Destination queue was an exception queue</td>
</tr>
<tr>
<td></td>
<td>· There was an error when applying transformation. The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log. If the default exception queue is specified, then the parameter returns a NULL value at dequeue time.</td>
</tr>
<tr>
<td>enqueue_time</td>
<td>Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user at enqueue time.</td>
</tr>
<tr>
<td>state</td>
<td>Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. The possible states follow:</td>
</tr>
<tr>
<td></td>
<td>· DBMS_AQ.READY: The message is ready to be processed.</td>
</tr>
<tr>
<td></td>
<td>· DBMS_AQ.WAITING: The message delay has not yet been reached.</td>
</tr>
<tr>
<td></td>
<td>· DBMS_AQ.PROCESSED: The message has been processed and is retained.</td>
</tr>
<tr>
<td></td>
<td>· DBMSAQ.EXPIRED: The message has been moved to the exception queue.</td>
</tr>
</tbody>
</table>
## Table 284-13  (Cont.) MESSAGE_PROPERTIES_T Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender_id</td>
<td>The application-sender identification specified at enqueue time by the message producer. Sender id is of type aq$_agent. Sender name is required for secure queues at enqueue time. This must be a valid AQ agent name, mapped to the database user performing the enqueue operation, through dbms_aqadm.enable_db_access procedure call. Sender address and protocol should not be specified. The Sender id in the message properties returned at dequeue time may have a sender address if the message was propagated from another queue. The value of the address is the source_queue, source database name if it was a remote database [format source_queue@source_database_name]</td>
</tr>
<tr>
<td>original_msgid</td>
<td>This parameter is used by Oracle Database Advanced Queuing for propagating messages.</td>
</tr>
<tr>
<td>signature</td>
<td>Currently not implemented</td>
</tr>
<tr>
<td>transaction_group</td>
<td>Specifies the transaction_group for the dequeued message. Messages belonging to the same transaction group will have the same value for this attribute. This attribute is only set by the DBMS_AQ.DEQUEUE_ARRAY. This attribute cannot be used to set the transaction group of a message through DBMS_AQ.ENQUEUE or DBMS_AQ.ENQUEUE_ARRAY calls.</td>
</tr>
<tr>
<td>user_property</td>
<td>This optional attribute is used to store additional information about the payload.</td>
</tr>
<tr>
<td>delivery_mode</td>
<td>The message publisher specifies the delivery mode in the message_properties. This can be DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT. Array enqueue is only supported for buffered messages with an array size of ‘1’.</td>
</tr>
</tbody>
</table>

### 284.2.18 MESSAGE_PROPERTIES_ARRAY_T Type

This type is used by `dbms_aq.enqueue_array` and `dbms_aq.dequeue_array` calls to hold the set of message properties.

Each element in the `payload_array` should have a corresponding element in the `MESSAGE_PROPERTIES_ARRAY_T` VARRAY.

*See Also:*

"MESSAGE_PROPERTIES_T Type"

**Syntax**

```sql
TYPE MESSAGE_PROPERTIES_ARRAY_T IS VARRAY (2147483647)
    OF MESSAGE_PROPERTIES_T;
```
284.2.19 MSGID_ARRAY_T Type

The `msgid_array_t` type is used in `dbms_aq.enqueue_array` and `dbms_aq.dequeue_array` calls to hold the set of message IDs that correspond to the enqueued or dequeued messages.

Syntax

```
TYPE MSGID_ARRAY_T IS TABLE OF RAW(16) INDEX BY BINARY_INTEGER
```
DBFS Content Interface Types

DBFS Content Interface Types are public types that support the DBMS_DBFS_CONTENT interface.

This chapter contains the following topics:

• Overview
• Security Model
• Data Structures

Related Topics

• DBMS_DBFS_CONTENT
  The DBMS_DBFS_CONTENT package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.

285.1 DBFS Content Interface Types Overview

The type definitions described in this chapter support the DBMS_DBFS_CONTENT interface in implementing metadata tables, packages, views, dependent application-side entities, and service-provider entities.

285.2 DBFS Content Interface Types Security Model

The user can access the content operational and administrative interfaces (packages, types, tables, and so on) with the DBFS_ROLE. This role can be granted to users as needed.

285.3 DBFS Content Interface Types —Data Structures

Types that support the DBMS_DBFS_CONTENT interface include both Object and Table types.

Object Types

• DBMS_DBFS_CONTENT_CONTEXT_T Object Type
• DBMS_DBFS_CONTENT_LIST_ITEM_T Object Type
• DBMS_DBFS_CONTENT_PROPERTY_T Object Type

Table Types

• DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type
• DBMS_DBFS_CONTENT_PROPERTIES_T Table Type
• DBMS_DBFS_CONTENT_RAW_T Table Type
285.3.1 DBMS_DBFS_CONTENT_CONTEXT_T Object Type

This type describes the execution context for the providers. It provides the user performing the operation with the Access Control List, the owner of the item(s), a timestamp for doing asof queries, and whether or not the item(s) are read_only. This type can be used both as input, in the case of path item creation functions, and output, in the case of path item query, or both.

Syntax

CREATE OR REPLACE TYPE dbms_dbfs_content_context_t
  AUTHID DEFINER
AS OBJECT {
  principal     VARCHAR2(32),
  acl           VARCHAR2(1024),
  owner         VARCHAR2(32),
  asof          TIMESTAMP,
  read_only     INTEGER);

Fields

Table 285-1  DBMS_DBFS_CONTENT_CONTEXT_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>File system user</td>
</tr>
<tr>
<td>acl</td>
<td>Access control list</td>
</tr>
<tr>
<td>owner</td>
<td>Path item owner</td>
</tr>
<tr>
<td>asof</td>
<td>Timestamp</td>
</tr>
<tr>
<td>read_only</td>
<td>Nonzero if the path item is read-only</td>
</tr>
</tbody>
</table>

285.3.2 DBMS_DBFS_CONTENT_LIST_ITEM_T Object Type

This type describes a type to assist in listing the contents of a directory.

Syntax

CREATE OR REPLACE TYPE dbms_dbfs_content_list_item_t
  AUTHID DEFINER
AS OBJECT {
  path          VARCHAR2(1024),
  item_name     VARCHAR2(256),
  item_type     INTEGER);

Fields

Table 285-2  DBMS_DBFS_CONTENT_LIST_ITEM_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path to the path item</td>
</tr>
<tr>
<td>item_name</td>
<td>Name of the path item</td>
</tr>
</tbody>
</table>
Table 285-2  (Cont.) DBMS_DBFS_CONTENT_LIST_ITEM_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_type</td>
<td>Type of path item. (See Table 48-4</td>
</tr>
</tbody>
</table>

285.3.3 DBMS_DBFS_CONTENTPROPERTY_T Object Type

This type describes a single \((\text{name}, \text{value}, \text{typecode})\) property tuple. All properties (standard, optional, and user-defined) are described using such tuples.

The type is used by both the client-facing interfaces and by store providers for the DBMS_DBFS_CONTENT interface.

Syntax

```sql
CREATE OR REPLACE TYPE dbms_dbfs_content_property_t
  AUTHID DEFINER
AS OBJECT (
  propname     VARCHAR2(32),
  propvalue    VARCHAR2(1024),
  typecode     INTEGER);
```

Fields

Table 285-3  DBMS_DBFS_CONTENTPROPERTY_T Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prop_name</td>
<td>Name of property</td>
</tr>
<tr>
<td>prop_value</td>
<td>Value of property</td>
</tr>
<tr>
<td>typecode</td>
<td>Property type (See Constants in DBMS_TYPES)</td>
</tr>
</tbody>
</table>

285.3.4 DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type

This type is a variable-sized array of DBMS_DBFS_CONTENT_LIST_ITEM_T Object Type. It is used by both the client-facing interfaces and by store providers for the DBMS_DBFS_CONTENT interface.

Syntax

```sql
CREATE OR REPLACE TYPE dbms_dbfs_content_list_items_t AS
  TABLE OF dbms_dbfs_content_list_item_t;
```

Related Topics

- **DBMS_DBFS_CONTENT_LIST_ITEM_T Object Type**
  This type describes a type to assist in listing the contents of a directory.
- **DBMS_DBFS_CONTENT**
  The DBMS_DBFS_CONTENT package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.
285.3.5 DBMS_DBFS_CONTENT_PROPERTIES_T Table Type

This type is a variable-sized array of property tuples of DBMS_DBFS_CONTENT_PROPERTY_T Object Type. It is used by both the client-facing interfaces and by store providers for the DBMS_DBFS_CONTENT interface.

Syntax

CREATE OR REPLACE TYPE dbms_dbfs_content_properties_t AS TABLE OF dbms_dbfs_content_property_t;

Related Topics

• DBMS_DBFS_CONTENTPROPERTY_T Object Type
  This type describes a single (name, value, typecode) property tuple. All properties (standard, optional, and user-defined) are described using such tuples.

285.3.6 DBMS_DBFSCONTENT_RAW_T Table Type

This type is an array of RAW. It is to enable RAW data transport for batch interfaces in the DBMS_DBFSCONTENT interface.

Syntax

CREATE OR REPLACE TYPE dbms_dbfs_content_raw_t AS TABLE OF RAW(32767);

Related Topics

• DBMS_DBFSCONTENT
  The DBMS_DBFSCONTENT package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.
Oracle supports the UriType family of types that can be used to store and query Uri-refs inside the database. The UriType itself is an abstract object type and the HTTPURITYPE, XDBURITYPE and DBURITYPE are subtypes of it.

You can create a UriType column and store instances of the DBURITYPE, XDBURITYPE or the HTTPURITYPE inside of it. You can also define your own subtypes of the UriType to handle different URL protocols.

Oracle also provides a UriFactory package that can be used as a factory method to automatically generate various instances of these UriTypes by scanning the prefix, such as http:// or /oradb. You can also register your subtype and provide the prefix that you support. For instance, if you have written a subtype to handle the gopher protocol, you can register the prefix gopher:// to be handled by your subtype. The UriFactory will then generate your subtype instance for any URL starting with that prefix.

This chapter contains the following topics:

- Summary of URITYPE Supertype Subprograms
- Summary of HTTPURITYPE Subtype Subprograms
- Summary of DBURITYPE Subtype Subprograms
- Summary of XDBURITYPE Subtype Subprograms
- Summary of URIFACTORY Package Subprograms

### 286.1 Summary of URITYPE Supertype Subprograms

The UriType is the abstract super type. It provides a standard set of functions to get the value pointed to by the URI. The actual implementation of the protocol must be defined by the subtypes of this type.

Instances of this type cannot be created directly. However, you can create columns of this type and store subtype instances in it, and also select from columns without knowing the instance of the URL stored.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETBLOB</td>
<td>Returns the BLOB located at the address specified by the URL.</td>
</tr>
</tbody>
</table>
Table 286-1  (Cont.) URITYPE Type Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETCLOB</td>
<td>Returns the CLOB located at the address specified by the URL.</td>
</tr>
<tr>
<td>GETCONTENTTYPE</td>
<td>Returns the URL, in escaped format, stored inside the UriType instance.</td>
</tr>
<tr>
<td>GETEXTERNALURL</td>
<td>Returns the URL, in escaped format, stored inside the UriType instance.</td>
</tr>
<tr>
<td>GETURL</td>
<td>Returns the URL, in non-escaped format, stored inside the UriType instance.</td>
</tr>
<tr>
<td>GETXML</td>
<td>Returns the XMLType located at the address specified by the URL.</td>
</tr>
</tbody>
</table>

286.1.1 UriType Supertype GETBLOB

This function returns the BLOB located at the address specified by the URL.

This function can be overridden in the subtype instances. The options are described below.

This function returns the BLOB located at the address specified by the URL.

**Syntax**

```sql
MEMBER FUNCTION getBlob()
  RETURN BLOB;
```

This function returns the BLOB located at the address specified by the URL and the content type.

**Syntax**

```sql
MEMBER FUNCTION getBlob(content OUT VARCHAR2)
  RETURN BLOB;
```

This function returns the BLOB located at the address specified by the URL in the specified character set.

**Syntax**

```sql
FUNCTION getBlob(csid IN NUMBER)
  RETURN BLOB;
```

| Table 286-2  UriType Supertype GETBLOB Parameters |
|--------------|--------------------------------------------------|
| Parameter    | IN / OUT  | Description                                                                 |
| content      | (OUT)     | Content type of the document to which URI is pointing.                      |
| csid         | (IN)      | Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error |
286.1.2 UriType Supertype GETCLOB

This function returns the CLOB located at the address specified by the URL.

This function can be overridden in the subtype instances. This function returns either a permanent CLOB or a temporary CLOB. If a temporary CLOB is returned, it must be freed. The options are described below.

This function returns the CLOB located at the address specified by the URL.

**Syntax**

MEMBER FUNCTION getClob()
    RETURN CLOB;

This function returns the CLOB located at the address specified by the URL and the content type.

**Syntax**

MEMBER FUNCTION getClob(content OUT VARCHAR2)
    RETURN CLOB;

**Table 286-3** UriType Supertype GETCLOB Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
</tbody>
</table>

286.1.3 UriType Supertype GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI.

This function can be overridden in the subtype instances. This function returns the content type as VARCHAR2.

**Syntax**

MEMBER FUNCTION getContentType()
    RETURN VARCHAR2;

286.1.4 UriType Supertype GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the UriType instance.

The subtype instances override this member function to provide additional semantics. For instance, the HTTPURITYPE function does not store the prefix http:// in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the getExternalUrl function or the getUrl function to get to the URL value instead of using the attribute present in the UriType instance.

**Syntax**

MEMBER FUNCTION getExternalUrl()
    RETURN varchar2;
286.1.5 UriType Supertype GETURL

This function returns the URL, in non-escaped format, stored inside the UriType instance.

The subtype instances override this member function to provide additional semantics. For instance, the HTTPURITYPE function does not store the prefix http:// in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the getExternalUrl function or the getUrl function to get to the URL value instead of using the attribute present in the UriType instance.

Syntax

MEMBER FUNCTION getUrl()
RETURN varchar2;

286.1.6 UriType Supertype GETXML

This function returns the XMLType located at the address specified by the URL.

This function can be overridden in the subtype instances. The options are described below.

This function returns the XMLType located at the address specified by the URL.

Syntax

MEMBER FUNCTION getXML()
RETURN XMLType;

MEMBER FUNCTION getXML(content OUT VARCHAR2)
RETURN XMLType;

Table 286-4 UriType Supertype GETXML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
</tbody>
</table>

286.2 Summary of HTTPURITYPE Subtype Subprograms

The HTTPURITYPE is a subtype of the UriType that provides support for the HTTP protocol. This uses the UTL_HTTP package underneath to access the HTTP URLs. Proxy and secure wallets are not supported in this release.
Table 286-5 HTTPURITYPE Type Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEURI</td>
<td>Creates an instance of HTTPURITYPE from the given URI.</td>
</tr>
<tr>
<td>GETBLOB</td>
<td>Returns the BLOB located at the address specified by the URL.</td>
</tr>
<tr>
<td>GETCLOB</td>
<td>Returns the CLOB located at the address specified by the URL.</td>
</tr>
<tr>
<td>GETCONTENTTYPE</td>
<td>Returns the content type of the document pointed to by the URI.</td>
</tr>
<tr>
<td>GETEXTERNALURL</td>
<td>Returns the URL, in escaped format, stored inside the UriType instance.</td>
</tr>
<tr>
<td>GETURL</td>
<td>Returns the URL, in non-escaped format, stored inside the UriType instance.</td>
</tr>
<tr>
<td>GETXML</td>
<td>Returns the XMLType located at the address specified by the URL.</td>
</tr>
<tr>
<td>HTTPURITYPE</td>
<td>Creates an instance of HTTPURITYPE from the given URI.</td>
</tr>
</tbody>
</table>

286.2.1 HttpUriType Subtype CREATEURI

This static function constructs a HTTPURITYPE instance. The HTTPURITYPE instance does not contain the prefix http:// in the stored URL.

Syntax

STATIC FUNCTION createUri(
    url IN varchar2)
RETURN HTTPURITYPE;

Table 286-6 HttpUriType Subtype CREATEURI Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>(IN)</td>
<td>The URL string containing a valid HTTP URL; escaped format.</td>
</tr>
</tbody>
</table>

286.2.2 HttpUriType Subtype GETBLOB

This function returns the BLOB located at the address specified by the HTTP URL.

The subprograms of the URI Subtype ATTPURITYPE member subprogram GETBLOB are described below.

This function returns the BLOB located at the address specified by the HTTP URL.

Syntax

MEMBER FUNCTION getBlob()
RETURN BLOB;

This function returns the BLOB located at the address specified by the HTTP URL and the content type.
Syntax

MEMBER FUNCTION getBlob(content OUT VARCHAR2)
    RETURN BLOB;

This function returns the BLOB located at the address specified by the URL in the specified character set.

Syntax

FUNCTION getBlob(csid IN NUMBER)
    RETURN BLOB;

Table 286-7  HttpUriType Subtype GETBLOB Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
<tr>
<td>csid</td>
<td>(IN)</td>
<td>Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.</td>
</tr>
</tbody>
</table>

286.2.3 HttpUriType Subtype GETCLOB

This function returns the CLOB located by the HTTP URL address. If a temporary CLOB is returned, it must be freed.

The subprograms of the URI Subtype HTTPURITYPE member subprogram GETCLOB are described below.

Returns the CLOB located at the address specified by the HTTP URL.

Syntax

MEMBER FUNCTION getClob()
    RETURN CLOB;

Returns the CLOB located at the address specified by the HTTP URL and the content type.

Syntax

MEMBER FUNCTION getClob(content OUT VARCHAR2)
    RETURN CLOB;

Table 286-8  HttpUriType Subtype GETCLOB Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
</tbody>
</table>
286.2.4 HttpUriType Subtype GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI.

Syntax

MEMBER FUNCTION getContentType()
RETURN VARCHAR2;

286.2.5 HttpUriType Subtype GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the HTTPURITYPE instance. The subtype instances override this member function.

The HTTPURITYPE function does not store the prefix http://, but generates it for the external URL.

Syntax

MEMBER FUNCTION getExternalUrl()
RETURN varchar2;

286.2.6 HttpUriType Subtype GETURL

This function returns the URL, in non-escaped format, stored inside the HTTPURITYPE instance.

Syntax

MEMBER FUNCTION getUrl()
RETURN varchar2;

286.2.7 HttpUriType Subtype GETXML

This function returns the XMLType located at the address specified by the URL. An error is thrown if the address does not point to a valid XML document.

The subprograms of the URI Subtype HttpUriType member subprogram GETXML are described below.

This function returns the XMLType located at the address specified by the URL.

Syntax

MEMBER FUNCTION getXML()
RETURN XMLType;

This function returns the XMLType located at the address specified by the URL and the content type.

Syntax

MEMBER FUNCTION getXML(content OUT VARCHAR2)
RETURN XMLType;
### 286.2.8 HttpUriType Subtype HTTPURITYPE

This constructs a HTTPURITYPE instance. The HTTPURITYPE instance does not contain the prefix http:// in the stored URL.

**Syntax**

```sql
CONSTRUCTOR FUNCTION HTTPURITYPE(
    url IN VARCHAR2);
```

### 286.3 Summary of DBURITYPE Subtype Subprograms

The DBURITYPE is a subtype of the UriType that provides support for DBUri-refs. A DBUri-ref is an intra-database URL that can be used to reference any row or row-column data in the database.

The URL is specified as an XPath expression over a XML visualization of the database. The schemas become elements which contain tables and views. These tables and views further contain the rows and columns inside them.

### Table 286-11  DBURITYPE Type Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEURI</td>
<td>Constructs a DBURITYPE instance.</td>
</tr>
<tr>
<td>DBURITYPE</td>
<td>Creates an instance of DBURITYPE from the given URI.</td>
</tr>
<tr>
<td>GETBLOB</td>
<td>Returns the BLOB located at the address specified by the DBURITYPE instance.</td>
</tr>
<tr>
<td>GETCLOB</td>
<td>Returns the CLOB located at the address specified by the DBURITYPE instance.</td>
</tr>
<tr>
<td>GETCONTENTTYPE</td>
<td>Returns the content type of the document pointed to by the URI.</td>
</tr>
<tr>
<td>GETEXTERNALURL</td>
<td>Returns the URL, in escaped format, stored inside the DBURITYPE instance.</td>
</tr>
<tr>
<td>GETURL</td>
<td>Returns the URL, in non-escaped format, stored inside the DBURITYPE instance.</td>
</tr>
</tbody>
</table>
### Table 286-11  (Cont.) DBURITYPE Type Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETXML</td>
<td>Returns the XMLType located at the address specified by the URL</td>
</tr>
</tbody>
</table>

#### 286.3.1 DBUriType Subtype CREATEURI

This static function constructs a DBURITYPE instance. Parses the URL given and creates a DBURITYPE instance.

**Syntax**

```sql
STATIC FUNCTION createUri(
    url IN varchar2)
RETURN DBURITYPE;
```

**Table 286-12  DBUriType Subtype CREATEURI Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>(IN)</td>
<td>The URL string, in escaped format, containing a valid DBURITYPE.</td>
</tr>
</tbody>
</table>

#### 286.3.2 DBUriType Subtype DBURITYPE

This constructs a DBURITYPE instance.

**Syntax**

```sql
CONSTRUCTOR FUNCTION DBURITYPE(
    url IN varchar2);
```

**Table 286-13  DBUriType Subtype DBURITYPE Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>(IN)</td>
<td>The URL string containing a valid DBURITYPE. The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.</td>
</tr>
</tbody>
</table>

#### 286.3.3 DBUriType Subtype GETBLOB

This function returns the BLOB located at the address specified by the URL.

The subprograms of the URI Subtype DBURITYPE member subprogram GETBLOB are described below. The options are described in the following table.

This function returns the BLOB located at the address specified by the URL.
Syntax

MEMBER FUNCTION getBlob()
  RETURN BLOB;

This function returns the BLOB located at the address specified by the URL and the content type.

Syntax

MEMBER FUNCTION getBlob(content OUT VARCHAR2)
  RETURN BLOB;

This function returns the BLOB located at the address specified by the URL in the specified character set.

Syntax

FUNCTION getBlob(csid IN NUMBER)
  RETURN BLOB;

Table 286-14  DBUriType Subtype GETBLOB Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
<tr>
<td>csid</td>
<td>(IN)</td>
<td>Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.</td>
</tr>
</tbody>
</table>

286.3.4 DBUriType Subtype GETCLOB

This function returns the CLOB located at the address specified by the DBURIType instance.

If a temporary CLOB is returned, it must be freed. The document returned may be an XML document or a text document. When the DBUri-ref identifies an element in the XPath, the result is a well-formed XML document. On the other hand, if it identifies a text node, then what is returned is only the text content of the column or attribute. The options are described below.

The following function returns the CLOB located at the address specified by the DBURIType instance.

Syntax

MEMBER FUNCTION getClob()
  RETURN CLOB;

The following function returns the CLOB located at the address specified by the DBURIType instance and the content type.

Syntax

MEMBER FUNCTION getClob(content OUT VARCHAR2)
  RETURN CLOB;
Table 286-15  DBUriType Subtype GETCLOB Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
</tbody>
</table>

286.3.5 DBUriType Subtype GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI.

Syntax

```
MEMBER FUNCTION getContentType()
RETURN VARCHAR2;
```

286.3.6 DBUriType Subtype GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the DBURITYPE instance. The DBUri servlet URL that processes the DBURITYPE has to be appended before using the escaped URL in Web pages.

Syntax

```
MEMBER FUNCTION getExternalUrl()
RETURN varchar2;
```

286.3.7 DBUriType Subtype GETURL

This function returns the URL, in non-escaped format, stored inside the DBURITYPE instance.

Syntax

```
MEMBER FUNCTION getUrl()
RETURN varchar2;
```

286.3.8 DBUriType Subtype GETXML

This function returns the XMLType located at the address specified by the URL.

The subprograms of the URI Subtype DBPURITYPE member subprogram GETXML are described below. The options are described in the following table.

This function returns the XMLType located at the address specified by the URL.

Syntax

```
MEMBER FUNCTION getXML()
    RETURN XMLType;
```

This function returns the XMLType located at the address specified by the URL and the content type.
Syntax

MEMBER FUNCTION getXML(content OUT VARCHAR2)
    RETURN XMLType;

Table 286-16  DBUriType Subtype GETXML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
</tbody>
</table>

286.4 Summary of XDBURITYPE Subtype Subprograms

XDBURITYPE is a new subtype of URIType. It provides a way to expose documents in the Oracle XML DB hierarchy as URIs that can be embedded in any URIType column in a table.

The URL part of the URI is the hierarchical name of the XML document it refers to. The optional fragment part uses the XPath syntax, and is separated from the URL part by '#'. The more general XPointer syntax for specifying a fragment is not currently supported.

Table 286-17  XDBURITYPE Type Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEURI</td>
<td>Returns the UriType corresponding to the specified URL.</td>
</tr>
<tr>
<td>GETBLOB</td>
<td>Returns the BLOB corresponding to the contents of the document specified by the XDBURITYPE instance.</td>
</tr>
<tr>
<td>GETCLOB</td>
<td>Returns the CLOB corresponding to the contents of the document specified by the XDBURITYPE instance.</td>
</tr>
<tr>
<td>GETCONTENTTYPE</td>
<td>Returns the content type of the document pointed to by the URI.</td>
</tr>
<tr>
<td>GETEXTERNALURL</td>
<td>Returns the URL, in escaped format, stored inside the XDBURITYPE instance.</td>
</tr>
<tr>
<td>GETURL</td>
<td>Returns the URL, in non-escaped format, stored inside the XDBURITYPE instance.</td>
</tr>
<tr>
<td>GETXML</td>
<td>Returns the XMLType corresponding to the contents of the document specified by the URL.</td>
</tr>
<tr>
<td>XDBURITYPE</td>
<td>Creates an instance of XDBURITYPE from the given URI.</td>
</tr>
</tbody>
</table>

286.4.1 XDBUriType Subtype CREATEURI

This static function constructs a XDBURITYPE instance. It parses the given URL and creates a XDBURITYPE instance.

Syntax

STATIC FUNCTION createUri(
    url IN varchar2)
    RETURN XDBURITYPE
Parameter | IN / OUT | Description
--- | --- | ---
url | (IN) | The URL string, in escaped format, containing a valid XDBURITYPE.

286.4.2 XDBUriType Subtype GETBLOB

This function returns the **BLOB** located at the address specified by the **XDBURITYPE** instance.

The subprograms of the URI Subtype XDBURITYPE member subprogram GETBLOB are described below. The options are described in the following table.

This function returns the **BLOB** located at the address specified by the URL.

**Syntax**

```
MEMBER FUNCTION getBlob()
    RETURN BLOB;
```

This function returns the **BLOB** located at the address specified by the URL and the content type.

**Syntax**

```
MEMBER FUNCTION getBlob(content OUT VARCHAR2)
    RETURN BLOB;
```

This function returns the **BLOB** located at the address specified by the URL in the specified character set.

**Syntax**

```
FUNCTION getBlob(csid IN NUMBER)
    RETURN BLOB;
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
<tr>
<td>csid</td>
<td>(IN)</td>
<td>Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.</td>
</tr>
</tbody>
</table>

286.4.3 XDBUriType Subtype GETCLOB

This function returns the **CLOB** located at the address specified by the **XDBURITYPE** instance. If a temporary **CLOB** is returned, it must be freed.

The subprograms of the URI Subtype XDBURITYPE member subprogram GETCLOB are described below. The options are described in the following table.

Returns the **CLOB** located at the address specified by the **XDBUriType** instance.

**Syntax**

```
MEMBER FUNCTION getClob()
    RETURN CLOB;
```
Returns the CLOB located at the address specified by the XDBUriType instance and the content type.

Syntax

MEMBER FUNCTION getClob(content OUT VARCHAR2)
    RETURN CLOB;

Table 286-19 XDBUriType Subtype GETCLOB Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
</tbody>
</table>

286.4.4 XDBUriType Subtype GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI. This function returns the content type as VARCHAR2.

Syntax

MEMBER FUNCTION getContentType()
    RETURN VARCHAR2;

286.4.5 XDBUriType Subtype GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the XDBURITYPE instance.

Syntax

MEMBER FUNCTION getExternalUrl()
    RETURN varchar2;

286.4.6 XDBUriType Subtype GETURL

This function returns the URL, in non-escaped format, stored inside the XDBURITYPE instance.

Syntax

MEMBER FUNCTION getUrl()
    RETURN varchar2;

286.4.7 XDBUriType Subtype GETXML

This function returns the XMLType located at the address specified by the URL.

The subprograms of the URI Subtype XDBURITYPE member subprogram GETXML are described below. The options are described in the following table.

This function returns the XMLType located at the address specified by the URL.
Syntax

MEMBER FUNCTION getXML()
   RETURN XMLType;

This function returns the XMLType located at the address specified by the URL and the content type.

Syntax

MEMBER FUNCTION getXML(content OUT VARCHAR2)
   RETURN XMLType;

Table 286-20  XDBUriType Subtype GETXML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>(OUT)</td>
<td>Content type of the document to which URI is pointing.</td>
</tr>
</tbody>
</table>

286.4.8 XDBUriType subtype XDBURITYPE

This constructs a XDBURITYPE instance.

Syntax

CONSTRUCTOR FUNCTION XDBURITYPE(
   url     IN   VARCHAR2,
   flags   IN   RAW := NULL)
   RETURN self AS RESULT;

Table 286-21  XDBUriType subtype XDBURITYPE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>(IN)</td>
<td>The URL string containing a valid XDBUriType. The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.</td>
</tr>
<tr>
<td>flags</td>
<td>(IN)</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - Expand all XInclude elements before returning the result contents. If any XInclude element cannot be successfully resolved according to the XInclude fallback semantics, then an error is raised.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 - Indicates that any errors during document retrieval should be suppressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 - Both flag bits (1, 2) are enabled</td>
</tr>
</tbody>
</table>
286.5 Summary of URIFACTORY Package Subprograms

The UriFactory package contains factory methods that can be used to generate the appropriate instance of the URI types without having to hard code the implementation in the program.

The UriFactory package also provides the ability to register new subtypes of the UriType to handle various other protocols. For example, you can invent a new protocol ecom:// and define a subtype of the UriType to handle that protocol and register it with UriFactory. After that any factory method would generate the new subtype instance if it sees the ecom:// prefix.

<table>
<thead>
<tr>
<th>Table 286-22</th>
<th>URIFACTORY Type Subprograms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>GETURI</td>
<td>Returns the correct URL handler for the given URL string.</td>
</tr>
<tr>
<td>ESCAPEURI</td>
<td>Returns a URL in escaped format.</td>
</tr>
<tr>
<td>UNESCAPEURI</td>
<td>Returns a URL in unescaped format.</td>
</tr>
<tr>
<td>REGISTERURLHANDLER</td>
<td>Registers a particular type name for handling a particular URL.</td>
</tr>
<tr>
<td>UNREGISTERURLHANDLER</td>
<td>Unregisters a URL handler.</td>
</tr>
</tbody>
</table>

286.5.1 UriFactory Package GETURI

This factory method returns the correct URI handler for the given URI string.

It returns a subtype instance of the UriType that can handle the protocol. By default, it always creates an XDBURITYPE instance, if it cannot resolve the URL. A URL handler can be registered for a particular prefix using the REGISTERURLHANDLER function. If the prefix matches, GETURI would then use that subtype.

Syntax

FUNCTION getUri(
    url IN Varchar2)
RETURN UriType;

<table>
<thead>
<tr>
<th>Table 286-23</th>
<th>UriFactory Package GETURI Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>IN / OUT</td>
</tr>
<tr>
<td>uri</td>
<td>(IN)</td>
</tr>
</tbody>
</table>

286.5.2 UriFactory Package ESCAPEURI

This function returns a URL in escaped format.

The subtype instances override this member function to provide additional semantics. For instance, the HTTPURITYPE does not store the prefix http:// in the URL itself. When generating the external URL, it appends the prefix and generates it. For this
reason, use the `GETEXTERNALURL` function or the `GETURI` function to get to the URL value instead of using the attribute present in the `UriType`.

**Syntax**

```sql
MEMBER FUNCTION escapeUri()
RETURN varchar2;
```

<table>
<thead>
<tr>
<th>Table 286-24 UriFactory Package ESCAPEURI Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>url</td>
</tr>
</tbody>
</table>

### 286.5.3 UriFactory Package UNESCAPEURI

This function returns a URL in unescaped format.

This function is the reverse of the `ESCAPEURI` function. This function scans the string and converts any non-URL hexadecimal characters into the equivalent UTF-8 characters. Since the return type is a `VARCHAR2`, the characters would be converted into the equivalent characters as defined by the database character set.

**Syntax**

```sql
FUNCTION unescapeUri()
RETURN varchar2;
```

<table>
<thead>
<tr>
<th>Table 286-25 UriFactory Package UNESCAPEURI Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>url</td>
</tr>
</tbody>
</table>

### 286.5.4 UriFactory Package REGISTERURLHANDLER

This package registers a particular type name for handling a particular URL.

The type specified must be valid and must be a subtype of the `UriType` or one of its subtypes. It must also implement the `createUri` static member function. This function is called by the `GETURI` function to generate an instance of the type. The `stripprefix` parameter indicates that the prefix must be stripped off before calling this function.

**Syntax**

```sql
PROCEDURE registerUrlHandler(
    prefix IN varchar2,
    schemaName IN varchar2,
    typename IN varchar2,
    ignoreCase IN boolean := true,
    stripprefix IN boolean := true);
```
Table 286-26 UriFactory Package REGISTERURLHANDLER Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>(IN)</td>
<td>The prefix to handle; for example, http://.</td>
</tr>
<tr>
<td>schemaName</td>
<td>(IN)</td>
<td>Name of the schema where the type resides; case sensitive.</td>
</tr>
<tr>
<td>typename</td>
<td>(IN)</td>
<td>The name of the type to handle the URL; case sensitive.</td>
</tr>
<tr>
<td>ignoreCase</td>
<td>(IN)</td>
<td>Ignore case when matching prefixes.</td>
</tr>
<tr>
<td>stripprefix</td>
<td>(IN)</td>
<td>Strip prefix before generating the instance of the type.</td>
</tr>
</tbody>
</table>

286.5.5 UriFactory Package UNREGISTERURLHANDLER

This procedure unregisters a URL handler. This only unregisters user registered handler prefixes and not predefined system prefixes such as http://.

Syntax

PROCEDURE unregisterUrlHandler(
    prefix IN VARCHAR2);

Table 286-27 UriFactory Package UNREGISTERURLHANDLER Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>(IN)</td>
<td>The prefix to be unregistered.</td>
</tr>
</tbody>
</table>
**287**

**JMS Types**

PL/SQL users can use the `DBMS_AQ` package to enqueue and dequeue messages from JMS queues.

The JMS types member and static functions and procedures in this chapter are needed to populate JMS messages for enqueuing or to interpret a dequeued JMS message.

This chapter contains these topics:

- Overview
- Security Model
- Java Versus PL/SQL Datatypes
- More on Bytes_Stream and Map Messages
- Upcasting and Downcasting Between General and Specific Messages
- JMS Types Error Reporting
- Oracle JMS Type Constants
- JMS Types Error Reporting
- Oracle JMS Type Constants
- `CONVERT_JMS_SELECTOR`
- Summary of JMS Types

### 287.1 JMS Types Overview

Java Message Service (JMS) is a well known public standard interface for accessing messaging systems. Oracle JMS (OJMS) implements JMS based on Oracle Streams Advanced Queuing (AQ) and a relational database system (RDBMS). Messages are stored in queues as OJMS specific ADTs. Java clients use OJMS packages to enqueue, dequeue, and manipulate these messages.

PL/SQL users, on the other hand, use the `DBMS_AQ` package to enqueue and dequeue JMS messages and the member functions in this chapter to populate and interpret them. Oracle Database Advanced Queuing offers such member functions for the following JMS ADTs:

- `aq$_jms_header`
- `aq$_jms_message`
- `aq$_jms_text_message`
- `aq$_jms_bytes_message`
- `aq$_jms_map_message`
- `aq$_jms_stream_message`

In addition to these populating and interpreting member functions, Oracle Database Advanced Queuing offers:
287.2 JMS Types Security Model

PUBLIC is granted EXECUTE privilege in these JMS types.

- SYS.AQ$_JMS_MESSAGE Type
- SYS.AQ$_JMS_TEXT_MESSAGE Type
- SYS.AQ$_JMS_BYTES_MESSAGE Type
- SYS.AQ$_JMS_MAP_MESSAGE Type
- SYS.AQ$_JMS_STREAM_MESSAGE Type
- SYS.AQ$_JMS_OBJECT_MESSAGE Type
- SYS.AQ$_JMS_NAMEARRAY Type
- SYS.AQ$_JMS_VALUE Type
- SYS.AQ$_JMS_EXCEPTION Type

287.3 Java Versus PL/SQL Datatypes

Datatypes do not map one-to-one between PL/SQL and Java.

Some Java types, such as BYTE and SHORT, are not present in PL/SQL. PL/SQL type INT was chosen to represent these types. If a PL/SQL INT value intended to hold a Java BYTE or SHORT value exceeds the corresponding range Java enforces, an out-of-range error is thrown.

Other Java types have more than one counterpart in PL/SQL with different capabilities. A Java String can be represented by both VARCHAR2 and CLOB, but VARCHAR2 has a maximum limit of 4000 bytes. When retrieving TEXT data from map, stream, and bytes message types, a CLOB is always returned. When updating the map, stream and bytes message types, users can submit either a VARCHAR2 or CLOB.

Similarly, a Java BYTE ARRAY can be represented by both RAW and BLOB, with RAW having a maximum size of 32767. When retrieving BYTE ARRAY data from map, stream, and bytes message types, a BLOB is always returned. When updating the map, stream and bytes message types, users can submit either a RAW or BLOB.

See Also:
JMS specification 3.11.3, Conversion Provided by StreamMessage and MapMessage

New JMS Support in Oracle Database 10g

In Oracle Database 10g, a new AQ$_JMS_VALUE ADT has been added in the SYS schema for OJMS PL/SQL users. It is specifically used to implement the read_object
procedure of aq$ _jms_stream_message and get_object procedure of aq$ _jms_map_message, to mimic the Java general object class Object. AＱ$ ＪＭＳ ＶＡＬＵＥ ADT can represent any datatype that JMS StreamMessage and MapMessage can hold.

The collection ADT AＱ$ ＪＭＳ ＮＡＭＥＡＲＲＡＹ was added for the getNames method of MapMessage. It holds an array of names.

In this release the ADT AＱ$ ＪＭＳ ＥＸＣＥPTION was added to represent a Java exception thrown in an OJMS JAVA stored procedure on the PL/SQL side. Now you can retrieve a Java exception thrown by an OJMS stored procedure and analyze it on the PL/SQL side.

287.4 More on Bytes, Stream and Map Messages

Oracle uses Java stored procedure to implement some of the procedures of AＱ$ ＭＡＰ ＭＥＳＳＡＧＥ, AＱ$ ＪＭＳ ＳＴＲＥＡＭ ＭＥＳＳＡＧＥ, and AＱ$ ＪＭＳ ＢＹＴＥＳ ＭＥＳＳＡＧＥ types. These types have some common functionality that are different from AＱ$ ＪＭＳ ＴＥＸＴ ＭＥＳＳＡＧＥ type. This section discusses this common functionality.

This section contains these topics:

• Using Java Stored Procedures to Encode and Decode Oracle Database Advanced Queuing Messages
• Initialize the Jserv Static Variable
• Get the Payload Data Back to PL/SQL
• Garbage Collect the Static Variable
• Use a Message Store: A Static Variable Collection
• Typical Calling Sequences
• Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages
• Differences Between Bytes and Stream Messages
• Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes

Using Java Stored Procedures to Encode and Decode Oracle Database Advanced Queuing Messages

The major difference between map, stream, bytes, and other messages is that the message payload is encoded as a byte stream by JAVA. Retrieving and updating these payloads in PL/SQL therefore requires Oracle JAVA stored procedures.

A message payload is stored in two places during processing. On the PL/SQL side it is stored as the data members of a JMS message ADT, and on the Jserv side it is stored as a static variable. (Jserv is the JVM inside Oracle Database.) When the payload is processed, the payload data is first transformed to a static variable on the Jserv side. Once the static variable is initialized, all later updates on the message payload are performed on this static variable. At the end of processing, payload data is flushed back to the PL/SQL side.

Oracle provides member procedures that maintain the status of the Jserv static variable and enforce rules when calling these member procedures. These procedures are in the following ADTs:

• aq$ _jms_bytes_message
• aq$ _jms_map_message
Initialize the Jserv Static Variable

Before you make any other calls to manipulate the payload data, the Jserv static variable must be properly initialized. This is done by calling the `prepare` or `clear_body` procedure. The `prepare` procedure uses the payload data in PL/SQL ADTs to initialize the static variable, while `clear_body` initializes the static variable to an empty payload (empty hashtable or stream).

**Note:** It is important to call the `prepare` or `clear_body` procedure before any other calls to properly initialize the Jserv static variables. Usually these two methods are called once at the beginning. But they can be called multiple times for one message. Any call of these two methods without first calling the `flush` procedure wipes out all updates made to the messages.

Get the Payload Data Back to PL/SQL

Calling the `flush` procedure synchronizes changes made to the Jserv static variable back to the PL/SQL ADTs. The `flush` call is required when you want the changes made to be reflected in the ADT payload. It is important to synchronize the changes back to the ADT, because it is the ADT payload that matters.

Garbage Collect the Static Variable

The `clean` procedure forces garbage collection of the static variable. It is there to do cleanup and free JVM memory. You can avoid memory leaks by doing it immediately after finishing processing the message.

Use a Message Store: A Static Variable Collection

Instead of a single static variable, Oracle uses a collection of static variables to process the message payload on the Jserv side. This collection is called the message store. Each map, bytes, or stream message type has its own message store within one session.

Oracle uses the operation ID parameter to locate the correct static variable to work on within the message store. Initialization calls such as `prepare` and `clear_body` give users an operation ID, which is used in later message access.

After users complete message processing, they must call the `clean` procedure with the operation ID to clean up the message store. This avoids possible memory leaks. The `clean_all` static procedures of message ADTs `aq$jms_bytes_message`, `aq$jms_map_message`, and `aq$jms_stream_message` clean up all static variables of their corresponding message stores.

Typical Calling Sequences

This section describes typical procedures for retrieving and populating messages.

Here is a typical procedure for retrieving messages

1. Call `prepare` for a message.
This call also gives you an operation ID if you do not specify one.

2. Call multiple retrieving procedures with the provided operation ID.
3. Call the clean procedure with the provided operation ID.

Here is a typical procedure for populating messages:

1. Call clear_body for a message.
   
   For aq$jms_map_message, you can also call prepare to update the message based on
   the existing payload. This call also gives you an operation ID if you do not specify one.

2. Call multiple updating procedures with the provided operation ID.
3. Call the flush method with the provided operation ID.
4. Call the clean procedure with the provided operation ID.

**Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages**

According to the JMS specification, when a message is received, its body is read-only. Users
can call the clear_body method to make the body writable. This method erases the current
message body and sets the message body to be empty.

The OJMS JAVA API follows the rule set by JMS specification. In updating the JMS message
ADTs in PL/SQL, however, Oracle enforces the rule selectively:

- **Map messages**
  The restriction is relaxed, because adding more entries on top of a existing map payload
  is a convenient way for users to update the payload. Therefore there are no read-only or
  write-only modes for map messages.

- **Stream and bytes messages**
  The restriction is not relaxed, because these payloads use a stream when reading and
  writing data. It is difficult to update the payload while in the middle of a stream. Oracle
  enforces read-only and write-only modes in processing stream and bytes message
  payloads. Calling the prepare procedure initializes the message payload in read-only
  mode. Calling the clear_body procedure initializes the message payload in write-only
  mode.

  Calling the reset procedure resets the pointer to the beginning of the stream and
  switches the mode from write-only to read-only. The reset procedure keeps the updates
  made to the message payload in the Jserv static variable.

  The prepare procedure, on the other hand, overwrites the message payload in the Jserv
  static variable with the payload in the PL/SQL ADT.

  Oracle provides member function get_mode for users to query the mode.

**Differences Between Bytes and Stream Messages**

Member functions of bytes messages are not exactly the same as those of stream messages.
Stream messages are encoded using Java ObjectOutputStream and bytes messages are
encoded using Java DataOutputStream. In stream messages each primitive type is written
and read as a Java Object, but in a bytes message they are written and read as raw bytes
according to the encoding mechanism of DataOutputStream.

For stream messages, the read_bytes method works on a stream of bytes to the end of the
byte array field written by the corresponding write_bytes method. The read_bytes method
of bytes message works on a stream of bytes to the end of the whole byte stream. This is why the read_bytes member procedure of aq$_bytes_message also requires a length parameter to tell how long it is to read.

You will not see a type conversion error raised by bytes message, because bytes messages do not support type conversion.

Methods get_unsigned_byte and get_unsigned_short are available for bytes messages, but not for stream messages. This is because stream messages read Java objects, and there are no Java objects as unsigned bytes or unsigned shorts.

Methods read_string and write_string methods are not available for bytes messages. The bytes message ADT must enforce some character encoding. It has methods read_utf and write_utf which support utf-8 encoding.

**Note:**
All data written by bytes messages use DataOutputStream as the basis. See JDK API documentation JavaSoft.com for details on how the data is encoded into bytes.

## Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes

The payloads of bytes, map, and stream message types are stored as either RAW or BLOB in the database. In this release Oracle Database Advanced Queuing provides the following member functions to set and get these payloads as raw bytes without interpreting them:

set_bytes(payload IN BLOB)
set_bytes(payload IN RAW)
get_bytes(payload OUT BLOB)
get_bytes(payload OUT RAW)

These functions were provided for bytes messages in Oracle9i Release 2 (9.2).

### 287.5 Upcasting and Downcasting Between General and Specific Messages

OJMS ADT aq$_jms_message is used to represent a general message, so that different types of messages can reside on the same Oracle Database Advanced Queuing queue. Oracle Database Advanced Queuing supports retrieving and populating of aq$_jms_message by supporting upcasting and downcasting between this ADT and ADTs of specific message types.

To read an aq$_jms_message, you must first downcast it to a specific message type according to its message_type field.

To populate an aq$_jms_message, you must first populate a specific message and upcast it to aq$_jms_message. This avoids copying all member functions of other specific message ADTs to this ADT. It also guarantees that the manipulation of this ADT is consistent with other specific message ADTs.
# 287.6 JMS Types Error Reporting

This table lists Oracle JMS types related errors.

<table>
<thead>
<tr>
<th>ORA error number</th>
<th>dbms_jms_plsql package constants</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA-24190</td>
<td>ERROR_DATA_OVERFLOW</td>
<td>The payload data exceeds the size that an out parameter can hold. For example, the get_text procedure with a VARCHAR2 parameter of aq$_jms_text_message or get_bytes procedure with a RAW parameter of aq$_jms_bytes_message.</td>
</tr>
<tr>
<td>ORA-24191</td>
<td>ERROR_PROP_NAME_EXIST</td>
<td>Setting a property that is previous set</td>
</tr>
<tr>
<td>ORA-24192</td>
<td>ERROR_PROP_NAME_NULL</td>
<td>Occurs when setting a property with null property name.</td>
</tr>
<tr>
<td>ORA-24193</td>
<td>ERROR_EXCEED_RANGE</td>
<td>PL/SQL number type exceeds the valid range of the respective Java type. For example set_byte_property, set_short_property of aq$_jms_head ADT; set_byte and set_short of aq$_jms_map_message ADT; write_byte and write_short of aq$_jms_stream_message and aq$_jms_bytes_message ADT.</td>
</tr>
<tr>
<td>ORA-24194</td>
<td>ERROR_TYPE_MISMATCH</td>
<td>The type conversion between the Java type of the retrieving method and the Java type of a field of the payload is not valid.</td>
</tr>
<tr>
<td>ORA-24195</td>
<td>ERROR_MAP_TOO_LARGE</td>
<td>The size of the map exceeds the aq$_jms_namearray ADT capacity. The current size limit is 1024. You can use the get_names function with offset and length parameters to retrieve the name array in multiple small chunks.</td>
</tr>
<tr>
<td>ORA-24196</td>
<td>ERROR_WRONG_MODE</td>
<td>The message payload is being accessed with a wrong access mode. For example, trying to read a message payload with write-only mode or trying to write a message payload with the read-only mode.</td>
</tr>
<tr>
<td>ORA-24197</td>
<td>ERROR_JAVA_EXCEPTION</td>
<td>ORA-24197 error is raised when a Java exception is raised that does not fit in any of the other error categories. You can use the get_exception static procedure of aq$_jms_map_message, aq$_jms_bytes_message, and aq$_jms_stream_message to retrieve the exception information last thrown by the Java stored procedure. A single static variable is used to store the last exception and is overwritten if another exception is thrown before you retrieve it. A new ADT aq$_jms_exception is created to represent the exception information on the PL/SQL side.</td>
</tr>
<tr>
<td>ORA-24198</td>
<td>ERRORINVALID_ID</td>
<td>An invalid operation ID is being provided to access a message.</td>
</tr>
<tr>
<td>ORA-24199</td>
<td>ERROR_STORE_OVERFLOW</td>
<td>The number of messages (with the same type) that users are trying to manipulate exceeds the size of the message store on the Java stored procedure side. The current size of the store is 20. It unusual to need to manipulate more than 20 messages at the same time. A common mistake is to forget to call the clean procedure after using one message. The clean procedure frees the message slot for use by other messages attempting access.</td>
</tr>
</tbody>
</table>
287.7 Oracle JMS Type Constants

These constants can be useful when dealing with message type functions.

**DBMS_AQ Package Constants**

DBMS_AQ package constants specify different types of JMS messages. They are useful when dealing with general message types during upcasting and downcasting or constructing a general message with a specific message type:

- JMS_TEXT_MESSAGE CONSTANT BINARY_INTEGER;
- JMS_BYTES_MESSAGE CONSTANT BINARY_INTEGER;
- JMS_STREAM_MESSAGE CONSTANT BINARY_INTEGER;
- JMS_MAP_MESSAGE CONSTANT BINARY_INTEGER;
- JMS_OBJECT_MESSAGE CONSTANT BINARY_INTEGER;

**SYS.DBMS_JMS_PLSQL Package Constants**

SYS.DBMS_JMS_PLSQL package constants are new in Oracle Database 10g.

These constants specify the mode of message payload. They are useful when interpreting the mode of the message payload returned from the `get_mode` function:

- MESSAGE_ACCESS_READONLY CONSTANT PLS_INTEGER;
- MESSAGE_ACCESS_WRITEONLY CONSTANT PLS_INTEGER;

These constants specify the ADT type of an Oracle Database Advanced Queuing queue. They are useful during the conversion of JMS selectors to Oracle Database Advanced Queuing rules:

- DESTPLOAD_JMSTYPE CONSTANT PLS_INTEGER;
- DESTPLOAD_USERADT CONSTANT PLS_INTEGER;
- DESTPLOAD_ANYDATA CONSTANT PLS_INTEGER;

These constants specify the type of data that can be held by a `aq$_jms_value` type. They are useful when interpreting the `aq$_jms_value` returned by the `get_object` method of `AQ$_JMS_MAP_MESSAGE` or `read_object` method of `AQ$_JMS_STREAM_MESSAGE`:

- DATA_TYPE_BYTE CONSTANT PLS_INTEGER;
- DATA_TYPE_SHORT CONSTANT PLS_INTEGER;
- DATA_TYPE_INTEGER CONSTANT PLS_INTEGER;
- DATA_TYPE_LONG CONSTANT PLS_INTEGER;
- DATA_TYPE_FLOAT CONSTANT PLS_INTEGER;
- DATA_TYPE_DOUBLE CONSTANT PLS_INTEGER;
- DATA_TYPE_BOOLEAN CONSTANT PLS_INTEGER;
- DATA_TYPE_CHARACTER CONSTANT PLS_INTEGER;
- DATA_TYPE_STRING CONSTANT PLS_INTEGER;
- DATA_TYPE_BYTES CONSTANT PLS_INTEGER;
- DATA_TYPE_UNSIGNED_BYTE CONSTANT PLS_INTEGER;
- DATA_TYPE_UNSIGNED_SHORT CONSTANT PLS_INTEGER;

These constants specify the error number of the ORA errors that can be raised by the functions of message type ADTs. They are useful in user error handlers:

- ERROR_DATA_OVERFLOW CONSTANT PLS_INTEGER := -24190;
- ERROR_PROP_NAME_EXIST CONSTANT PLS_INTEGER := -24191;
- ERROR_PROP_NAME_NULL CONSTANT PLS_INTEGER := -24192;
- ERROR_EXCEED_RANGE CONSTANT PLS_INTEGER := -24193;
- ERROR_TYPE_MISMATCH CONSTANT PLS_INTEGER := -24194;
Oracle Database includes three stored procedures to help users convert JMS selectors into Oracle Database Advanced Queuing rules. These rules can be used in `ADD_SUBSCRIBER` operations as subscriber rules or in `DEQUEUE` operations as dequeue conditions. These procedures are in the `SYS.dbms_jms_plsql` package.

### Convert with Minimal Specification

The first procedure assumes the destination payload type is one of the JMS ADTs whose corresponding constant is `dbms_jms_plsql.DESTPLOAD_JMSTYPE` and also assumes that the J2EE compliant mode is true.

**Syntax**

Function `convert_jms_selector(selector IN VARCHAR2) RETURN VARCHAR2`

**Returns**

The converted Oracle Database Advanced Queuing rule or null if there is any conversion error.

**Exceptions**

ORA-24197 if the Java stored procedure throws an exception during execution.

### Convert with Destination Payload Type Specified

The second procedure takes one more parameter: `dest_pload_type`. The conversion of a JMS selector to an Oracle Database Advanced Queuing rule happens only if this parameter is `SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE` or `SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA`. The function returns exactly the same `VARCHAR2` value as the selector parameter if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_USERADT`. The function returns null if `dest_pload_type` parameter is none of these three constants.

This function assumes that the J2EE compliant mode is true.

**Syntax**

Function `convert_jms_selector(selector IN VARCHAR2, dest_pload_type IN PLS_INTEGER) RETURN VARCHAR2`

**Returns**

The converted Oracle Database Advanced Queuing rule or null if there is any conversion error.

**Exceptions**

ORA-24197 if the Java stored procedure throws an exception during execution.
Convert with Destination Payload Type and Compliant Mode Specified

The third procedure takes a \texttt{dest\_pload\_type} parameter and a \texttt{compliant} parameter. The conversion of a JMS selector to an Oracle Database Advanced Queuing rule happens only if the \texttt{dest\_pload\_type} parameter is \texttt{SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE} or \texttt{SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA}. The function returns exactly the same \texttt{VARCHAR2} value as the selector parameter if the \texttt{dest\_pload\_type} parameter is \texttt{SYS.dbms_jms_plsql.DESTPLOAD_USERADT}. The function returns null if the \texttt{dest\_pload\_type} parameter is none of these three constants.

The \texttt{compliant} parameter controls if the conversion is in J2EE compliant mode or not. The noncompliant conversion of a JMS selector is for backward compatibility.

Syntax

\begin{verbatim}
Function convert_jms_selector(
    selector         IN  VARCHAR2,
    dest_pload_type  IN  PLS_INTEGER,
    compliant        IN  BOOLEAN )
Returns
The converted Oracle Database Advanced Queuing rule or null if there is any conversion error.

Exceptions
ORA-24197 if the Java stored procedure throws an exception during execution.
\end{verbatim}

287.9 Summary of JMS Types

This lists shows the JMS types.

- \texttt{SYS.AQ$\_JMS\_MESSAGE Type}
- \texttt{SYS.AQ$\_JMS\_TEXT\_MESSAGE Type}
- \texttt{SYS.AQ$\_JMS\_BYTES\_MESSAGE Type}
- \texttt{SYS.AQ$\_JMS\_MAP\_MESSAGE Type}
- \texttt{SYS.AQ$\_JMS\_STREAM\_MESSAGE Type}
- \texttt{SYS.AQ$\_JMS\_OBJECT\_MESSAGE Type}
- \texttt{SYS.AQ$\_JMS\_NAMEARRAY Type}
- \texttt{SYS.AQ$\_JMS\_VALUE Type}
- \texttt{SYS.AQ$\_JMS\_EXCEPTION Type}

287.9.1 SYS.AQ$\_JMS\_MESSAGE Type

This ADT type can represent any of five different JMS message types: text message, bytes message, stream message, map message, or object message. Queues created using this ADT can therefore store all five types of JMS messages.

This section contains these topics:
• CONSTRUCT Static Functions
• Cast Methods
• JMS Header Methods
• System Properties Methods
• User Properties Methods
• Payload Methods

Syntax

TYPE AQ$ _JMS_MESSAGE AS OBJECT{
  header        aq$ _jms_header,
  senderid      varchar2(100),
  message_type  INT,
  text_len      INT,
  bytes_len     INT,
  text_vc       varchar2(4000),
  bytes_raw     raw(2000),
  text_lob      clob,
  bytes_lob     blob,
  STATIC FUNCTION  construct (mtype      IN  INT)
    RETURN aq$ _jms_message,
  STATIC FUNCTION  construct (text_msg   IN  aq$ _jms_text_message)
    RETURN aq$ _jms_message,
  STATIC FUNCTION  construct (bytes_msg  IN  aq$ _jms_bytes_message)
    RETURN aq$ _jms_message,
  STATIC FUNCTION  construct (stream_msg IN  aq$ _jms_stream_message)
    RETURN aq$ _jms_message,
  STATIC FUNCTION  construct (map_msg    IN  aq$ _jms_map_message)
    RETURN aq$ _jms_message,
  STATIC FUNCTION  construct (object_msg IN  aq$ _jms_object_message)
    RETURN aq$ _jms_message,
  MEMBER FUNCTION  cast_to_bytes_msg  RETURN aq$ _jms_bytes_message,
  MEMBER FUNCTION  cast_to_map_msg    RETURN aq$ _jms_map_message,
  MEMBER FUNCTION  cast_to_object_msg RETURN aq$ _jms_object_message,
  MEMBER FUNCTION  cast_to_stream_msg RETURN aq$ _jms_stream_message,
  MEMBER FUNCTION  cast_to_text_msg   RETURN aq$ _jms_text_message,
  MEMBER PROCEDURE  set_replyto  (replyto IN sys.aq$ _agent),
  MEMBER PROCEDURE  set_type     (type     IN  VARCHAR),
  MEMBER PROCEDURE  set_userid   (userid   IN  VARCHAR),
  MEMBER PROCEDURE  set_appid    (appid    IN  VARCHAR),
  MEMBER PROCEDURE  set_groupid  (groupid  IN  VARCHAR),
  MEMBER PROCEDURE  set_groupseq (groupseq IN  INT),
  MEMBER FUNCTION  get_replyto  RETURN sys.aq$ _agent,
  MEMBER FUNCTION  get_type     RETURN VARCHAR,
  MEMBER FUNCTION  get_userid   RETURN VARCHAR,
  MEMBER FUNCTION  get_appid    RETURN VARCHAR,
  MEMBER FUNCTION  get_groupid  RETURN VARCHAR,
  MEMBER FUNCTION  get_groupseq RETURN INT,
  MEMBER PROCEDURE  clear_properties,
  MEMBER PROCEDURE  set_boolean_property (property_name IN VARCHAR,
                                          property_value IN BOOLEAN),
  MEMBER PROCEDURE  set_byte_property   (property_name IN VARCHAR,
                                          property_value IN INT),
  MEMBER PROCEDURE  set_double_property (property_name IN VARCHAR,
                                          property_value IN DOUBLE PRECISION),
  MEMBER PROCEDURE  set_float_property  (property_name IN VARCHAR,
                                          property_value IN FLOAT),
  MEMBER PROCEDURE  set_int_property   (property_name IN VARCHAR,}
property_value IN INT),
MEMBER PROCEDURE set_long_property (property_name IN VARCHAR,
property_value IN NUMBER),
MEMBER PROCEDURE set_short_property (property_name IN VARCHAR,
property_value IN INT),
MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_text (payload IN VARCHAR2),
MEMBER PROCEDURE set_bytes (payload IN RAW),
MEMBER PROCEDURE get_text (payload OUT VARCHAR2),
MEMBER PROCEDURE get_bytes (payload OUT RAW),

CONSTRUCT Static Functions

There are six CONSTRUCT static functions in this type.

STATIC FUNCTION construct (mtype IN INT) RETURN aq$_jms_message

Creates an instance of aq$_jms_message, which can hold a specific type of JMS message (TextMessage, BytesMessage, MapMessage, StreamMessage or ObjectMessage). The message type of the created aq$_jms_message instance depends on the mtype parameter passed to the construct method. Once a message has been constructed, it can be used to store JMS messages of the type it has been constructed to hold.

The mtype parameter must be one of the following constants described in "Oracle JMS Type Constants":

DBMS_AQ.JMS_TEXT_MESSAGE
DBMS_AQ.JMS_BYTES_MESSAGE
DBMS_AQ.JMS_STREAM_MESSAGE
DBMS_AQ.JMS_MAP_MESSAGE
DBMS_AQ.JMS_OBJECT_MESSAGE

STATIC FUNCTION construct (text_msg IN aq$_jms_text_message) RETURN aq$_jms_message

Creates an aq$_jms_message from an aq$_jms_text_message.

STATIC FUNCTION construct (bytes_msg IN aq$_jms_bytes_message) RETURN aq$_jms_message;

Creates an aq$_jms_message from an aq$_jms_bytes_message.

STATIC FUNCTION construct (stream_msg IN aq$_jms_stream_message) RETURN aq$_jms_message;

Creates an aq$_jms_message from an aq$_jms_stream_message.
STATIC FUNCTION construct (map_msg IN aq$_jms_map_message) RETURN aq$_jms_message;
Creates an aq$_jms_message from an aq$_jms_map_message.

STATIC FUNCTION construct (object_msg IN aq$_jms_object_message) RETURN aq$_jms_message;
Creates an aq$_jms_message from an aq$_jms_object_message.

Cast Methods

cast_to_bytes_msg RETURN aq$_jms_bytes_message
Casts an aq$_jms_message to an aq$_jms_bytes_message. Returns an aq$_jms_bytes_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_BYTES_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_BYTES_MESSAGE.

cast_to_map_msg RETURN aq$_jms_map_message
Casts an aq$_jms_message to an aq$_jms_map_message. Returns an aq$_jms_map_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_MAP_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_MAP_MESSAGE.

cast_to_object_msg RETURN aq$_jms_object_message
Casts an aq$_jms_message to an aq$_jms_object_message. Returns an aq$_jms_object_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_OBJECT_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_OBJECT_MESSAGE.

cast_to_stream_msg RETURN aq$_jms_stream_message
Casts an aq$_jms_message to an aq$_jms_stream_message. Returns an aq$_jms_stream_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_STREAM_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_STREAM_MESSAGE.

cast_to_text_msg RETURN aq$_jms_text_message
Casts an aq$_jms_message to an aq$_jms_text_message. Returns an aq$_jms_text_message or null if the message_type attribute of the aq$_jms_message is not DBMS_AQ.JMS_TEXT_MESSAGE. This function raises ORA-24198 if the message_type field of the aq$_jms_message is not DBMS_AQJMS.JMS_TEXT_MESSAGE.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the replyto parameter, which corresponds to JMSReplyTo.

get_replyto RETURN sys.aq$_agent
Returns replyto, which corresponds to JMSReplyTo.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text and corresponds to JMSType.

get_type RETURN VARCHAR
Returns type, which corresponds to JMSType.
System Properties Methods

set_userid (userid IN VARCHAR)
Sets userid, which corresponds to JMSXUserID.

set_appid (appid IN VARCHAR)
Sets appid, which corresponds to JMSXAppID.

set_groupid (groupid IN VARCHAR)
Sets groupid, which corresponds to JMSXGroupID.

set_groupseq (groupseq IN INT)
Sets groupseq, which corresponds to JMSXGroupSeq.

get_userid RETURN VARCHAR
Returns userid, which corresponds to JMSXUserID.

get_appid RETURN VARCHAR
Returns appid, which corresponds to JMSXAppID.

get_groupid RETURN VARCHAR
Returns groupid, which corresponds to JMSXGroupID.

get_groupseq RETURN VARCHAR
Returns groupseq, which corresponds to JMSXGroupSeq.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether property_name is null or exists. If it is not null, the procedure stores property_value in an internal representation (a NUMBER type). Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If it is not null, the procedure checks whether property_value is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the byte datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether property_name is null or exists. If it is not null, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether property_name is null or exists. If it is not null, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.
set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If it is not null, the procedure checks whether property_value is within -2147483648 to 2147483647 (32-bits). This check is necessary because the INT datatype is 38 bits in PL/SQL and Oracle Database. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether property_name is null or exists. If it is not null, the procedure stores property_value. In PL/SQL and Oracle Database, the NUMBER datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If it is not null, the procedure checks whether property_value is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the short datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether property_name is null or exists. If it is not null, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a BOOLEAN property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a BYTE property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a DOUBLE property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a FLOAT property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a Integer property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a long property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a short property, then this function returns the value of the property. Otherwise it returns a null.
**get_string_property (property_name IN VARCHAR) RETURN VARCHAR**
If the property with the corresponding property name passed in exists, and if it is a STRING property, then this function returns the value of the property. Otherwise it returns a null.

**Payload Methods**

**set_text (payload IN VARCHAR2)**
Sets the payload, a VARCHAR value, to an internal representation.

**set_text (payload IN CLOB)**
Sets the payload, a CLOB value, to an internal representation.

**set_bytes (payload IN RAW)**
Sets the payload, a RAW value, to an internal representation.

**set_bytes (payload IN BLOB)**
Sets the payload, a BLOB value, to an internal representation.

**get_text (payload OUT VARCHAR2)**
Puts the internal representation of the payload into a VARCHAR2 variable payload.

**get_text (payload OUT CLOB)**
Puts the internal representation of the payload into a CLOB variable payload.

**get_bytes (payload OUT RAW)**
Puts the internal representation of the payload into a RAW variable payload.

**get_bytes (payload OUT BLOB)**
Puts the internal representation of the payload into a BLOB variable payload.

### 287.9.2 SYS.AQ$_JMS_TEXT_MESSAGE Type

This type is the ADT used to store a TextMessage in an Oracle Database Advanced Queuing queue.

This section contains these topics:

- **CONSTRUCT aq$_jms_text_message Function**
- **JMS Header Methods**
- **System Properties Methods**
- **User Properties Methods**
- **Payload Methods**

**Syntax**

```sql
TYPE AQ$_JMS_TEXT_MESSAGE AS OBJECT(
    header   aq$_jms_header,
    text_len INT,
    text_vc  varchar2(4000),
    text_lob clob,
STATIC FUNCTION construct RETURN aq$_jms_text_message,
MEMBER PROCEDURE set_replyto (replyto IN sys.aq$_agent),
MEMBER PROCEDURE set_type (type IN VARCHAR),
MEMBER FUNCTION get_replyto RETURN sys.aq$_agent,
```
MEMBER FUNCTION  get_type    RETURN VARCHAR,
MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
MEMBER FUNCTION get_userid   RETURN VARCHAR,
MEMBER FUNCTION get_appid    RETURN VARCHAR,
MEMBER FUNCTION get_groupid  RETURN VARCHAR,
MEMBER FUNCTION get_groupseq RETURN INT,
MEMBER PROCEDURE clear_properties,
MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR, property_value IN BOOLEAN),
MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR, property_value IN INT),
MEMBER PROCEDURE set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION),
MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR, property_value IN FLOAT),
MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR, property_value IN INT),
MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR, property_value IN NUMBER),
MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR, property_value IN INT),
MEMBER PROCEDURE set_string_property (property_name IN VARCHAR, property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property  (property_name IN VARCHAR) RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_text            (payload  IN  VARCHAR2),
MEMBER PROCEDURE set_text            (payload  IN  CLOB),
MEMBER PROCEDURE get_text            (payload  OUT VARCHAR2),
MEMBER PROCEDURE get_text            (payload  OUT CLOB));

CONSTRUCT aq$_jms_text_message Function

STATIC FUNCTION construct RETURN aq$_jms_text_message
Creates an empty aq$_jms_text_message.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to JMSType in JMS.

get_replyto RETURN sys.aq$_agent
Returns replyto, which corresponds to JMSReplyTo.
get_type RETURN VARCHAR
Returns type, which corresponds to JMSType.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets userid, which corresponds to JMSXUserID in JMS.

set_appid (appid IN VARCHAR)
Sets appid, which corresponds to JMSXAppID in JMS.

set_groupid (groupid IN VARCHAR)
Sets groupid, which corresponds to JMSXGroupID in JMS.

set_groupseq (groupseq IN INT)
Sets groupseq, which corresponds to JMSXGroupSeq in JMS.

get_userid RETURN VARCHAR
Returns userid, which corresponds to JMSXUserID.

get_appid RETURN VARCHAR
Returns appid, which corresponds to JMSXAppID.

get_groupid RETURN VARCHAR
Returns groupid, which corresponds to JMSXGroupID.

get_groupseq RETURN INT
Returns groupseq, which corresponds to JMSXGroupSeq.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether property_name is null or exists. If not, the procedure stores
property_value in an internal representation. Raises exception ORA-24191 if the
property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether
property_value is within -128 to 127 (8-bits). This check is necessary because
neither PL/SQL nor RDBMS defines the BYTE datatype. Raises exception ORA-24191
if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if
the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE
PRECISION)
Checks whether property_name is null or exists. If not, the procedure stores
property_value. Raises exception ORA-24191 if the property name exists or
ORA-24192 if the property name is null.
set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the INT datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
 Checks whether property_name is null or exists. If not, the procedure stores property_value. In PL/SQL and Oracle Database, the NUMBER datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the short datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a BOOLEAN property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a BYTE property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a DOUBLE property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a FLOAT property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a Integer property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a long property, then this function returns the value of the property. Otherwise it returns a null.
get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a short property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a STRING property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_text (payload IN VARCHAR2)
Sets the payload, a VARCHAR2 value, to an internal representation.

set_text (payload IN CLOB)
Sets the payload, a CLOB value, to an internal representation.

get_text (payload OUT VARCHAR2)
Puts the internal representation of the payload into a VARCHAR2 variable payload.

get_text (payload OUT CLOB)
Puts the internal representation of the payload into a CLOB variable payload.

287.9.3 SYS.AQ$_JMS_BYTES_MESSAGE Type

The SYS.AQ$_JMS_BYTES_MESSAGE type is the ADT used to store a BytesMessage in an Oracle Database Advanced Queuing queue.

This section contains these topics:

• CONSTRUCT aq$_jms_bytes_message Function
• JMS Header Methods
• System Properties Methods
• User Properties Methods
• Payload Methods

Syntax

TYPE AQ$_JMS_BYTES_MESSAGE AS OBJECT(
   header     aq$_jms_header,
   bytes_len  INT,
   bytes_raw  raw(2000),
   bytes_lob  blob,
STATIC FUNCTION construct RETURN aq$_jms_bytes_message,
MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
MEMBER PROCEDURE set_type     (type    IN VARCHAR),
MEMBER FUNCTION get_replyto RETURN sys.aq$_agent,
MEMBER FUNCTION get_type    RETURN VARCHAR,
MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
MEMBER PROCEDURE set_groupseq (groupseq IN INT),
MEMBER FUNCTION get_userid   RETURN VARCHAR,
MEMBER FUNCTION get_appid    RETURN VARCHAR,
MEMBER FUNCTION get_groupid  RETURN VARCHAR,
MEMBER FUNCTION get_groupseq RETURN INT,
MEMBER PROCEDURE clear_properties,
MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
    property_value IN BOOLEAN),
MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
    property_value IN INT),
MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
    property_value IN DOUBLE PRECISION),
MEMBER PROCEDURE set_float_property   (property_name IN VARCHAR,
    property_value IN FLOAT),
MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
    property_value IN INT),
MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
    property_value IN NUMBER),
MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
    property_value IN INT),
MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
    property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property  (property_name IN VARCHAR) RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_bytes           (payload  IN RAW),
MEMBER PROCEDURE set_bytes           (payload  IN BLOB),
MEMBER PROCEDURE get_bytes           (payload  OUT RAW),
MEMBER PROCEDURE get_bytes           (payload  OUT BLOB),
MEMBER FUNCTION  prepare             (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE reset               (id IN PLS_INTEGER),
MEMBER PROCEDURE flush               (id IN PLS_INTEGER),
MEMBER PROCEDURE clear_body          (id IN PLS_INTEGER),
MEMBER PROCEDURE clean               (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER FUNCTION get_mode           (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_boolean       (id IN PLS_INTEGER) RETURN BOOLEAN,
MEMBER FUNCTION read_byte          (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_bytes         (id IN PLS_INTEGER,
    value OUT NOCOPY BLOB, length IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_char          (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION read_double        (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION read_float         (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION read_int           (id IN PLS_INTEGER) RETURN INT,
MEMBER FUNCTION read_long          (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION read_short         (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_unsigned_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_unsigned_short (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE read_utf          (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE write_boolean     (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte        (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN RAW),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN BLOB),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN RAW,
    offset IN INT, length IN INT),
MEMBER PROCEDURE write_char        (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double      (id IN PLS_INTEGER,
    property_value IN INT),

Chapter 287
Summary of JMS Types
287-21
CONSTRUCT aq$_jms_bytes_message Function

STATIC FUNCTION construct RETURN aq$_jms_bytes_message
Creates an empty aq$_jms_bytes_message.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

get_replyto RETURN sys.aq$_agent
Returns replyto, which corresponds to JMSReplyTo.

get_type RETURN VARCHAR
Returns type, which corresponds to JMSType.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets userid, which corresponds to JMSXUserID in JMS.

get_userid RETURN VARCHAR
Returns userid, which corresponds to JMSXUserID.

set_appid (appid IN VARCHAR)
Sets appid, which corresponds to JMSXAppID in JMS.

get_appid RETURN VARCHAR
Returns appid, which corresponds to JMSXAppID.

set_groupid (groupid IN VARCHAR)
Sets groupid, which corresponds to JMSXGroupID in JMS.

get_groupid RETURN VARCHAR
Returns groupid, which corresponds to JMSXGroupID.

set_groupseq (groupseq IN INT)
Sets groupseq, which corresponds to JMSXGroupSeq in JMS.

get_groupseq RETURN NUMBER
Returns groupseq, which corresponds to JMSXGroupSeq.
User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether property_name is null or exists. If not, the procedure stores property_value in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the BYTE datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the INT datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether property_name is null or exists. If not, the procedure stores property_value. In PL/SQL and Oracle Database, the NUMBER datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the short datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.
get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a BOOLEAN property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a BYTE property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a DOUBLE property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a FLOAT property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a Integer property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a long property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a short property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a STRING property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_bytes (payload in RAW)
Sets the payload, a RAW value, to an internal representation.

set_bytes (payload in BLOB)
Sets the payload, a BLOB value, to an internal representation.

get_bytes (payload out RAW)
Puts the internal representation of the payload into a RAW variable payload. Raises exception ORA-24190 if the length of the internal payload is more than 32767 (the maximum length of RAW in PL/SQL).
get_bytes (payload out BLOB)
Puts the internal representation of the payload into a BLOB variable payload.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in aq$_jms_bytes_message and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter id is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If id is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
This function also sets the message access mode to MESSAGE_ACCESS_READONLY. Subsequent calls of write_XXX procedure raise an ORA-24196 error. Users can call the clear_body procedure to set the message access mode to MESSAGE_ACCESS_READONLY. This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

reset (id IN PLS_INTEGER)
Resets the starting position of the stream to the beginning and puts the bytes message in read-only mode. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to the aq$_jms_bytes_message. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter id is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If id is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
It also sets the message access mode to MESSAGE_ACCESS_WRITEONLY. Later calls of read_XXX procedure raise ORA-24196 error. Users can call the reset or prepare procedures to set the message access mode to MESSAGE_ACCESS_READONLY. Write-only and read-only modes affect only the payload functions of AQ$_JMS_BYTES_MESSAGE. They do not affect the header functions.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)
Closes and cleans up the DataInputStream or DataOutputStream at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.
get_mode (id IN PLS_INTEGER) RETURN PLS_INTEGER
Returns the current mode of this message. The return value is either
SYS.dbms_jms.plsql.MESSAGE_ACCESS_READONLY or
SYS.dbms_jms.plsql.MESSAGE_ACCESS_WRITEONLY. Raises exception ORA-24197 if
the Java stored procedure throws an exception during execution or ORA-24198 if the
operation ID is invalid.

read_boolean (id IN PLS_INTEGER) RETURN BOOLEAN
Reads a Boolean value from the bytes message and returns the Boolean value read.
Null is returned if the end of the message stream has been reached. Parameter id is
the operation ID. Raises exception ORA-24196 if the bytes message is in write-only
mode, ORA-24197 if the Java stored procedure throws an exception during execution,
or ORA-24198 if the operation ID is invalid.

read_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads a BYTE value from the bytes message and returns the BYTE value read. Null is
returned if the end of the stream has been reached. Because there is no BYTE type in
PL/SQL, Oracle Database uses PLS_INTEGER to represent a BYTE. Although PL/SQL
users get a PLS_INTEGER, they are guaranteed that the value is in the Java BYTE value
range. If this value is issued with a write_byte function, then there will not be an out
of range error. Parameter id is the operation ID. Raises exception ORA-24196 if the
bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws
an exception during execution, or ORA-24198 if the operation ID is invalid.

read_bytes (id IN PLS_INTEGER, value OUT NO COPY BLOB, length IN
PLS_INTEGER) RETURN PLS_INTEGER
Reads length of the bytes from bytes message stream into value and returns the total
number of bytes read. If there is no more data (because the end of the stream has
been reached), then it returns -1. Raises exceptions ORA-24196 if the bytes message
is in write-only mode, ORA-24197 if the Java stored procedure throws an exception
during execution, or ORA-24198 if the operation ID is invalid.

read_char (id IN PLS_INTEGER) RETURN CHAR
Reads a character value from the bytes message and returns the character value
read. Null is returned if the end of the stream has been reached. Raises exception
ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored
procedure throws an exception during execution, or ORA-24198 if the operation ID is
invalid.

read_double (id IN PLS_INTEGER) RETURN DOUBLE PRECISION
Reads a double from the bytes message and returns the character value read. Null is
returned if the end of the stream has been reached. Raises exception ORA-24196 if the
bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an
exception during execution, or ORA-24198 if the operation ID is invalid.

read_float (id IN PLS_INTEGER) RETURN FLOAT
Reads a float from the bytes message and returns the float read. Null is returned if the
end of the stream has been reached. Raises exception ORA-24196 if the bytes
message is in write-only mode, ORA-24197 if the Java stored procedure throws an
exception during execution, or ORA-24198 if the operation ID is invalid.

read_int (id IN PLS_INTEGER) RETURN INT
Reads an INT from the bytes message and returns the INT read. Null is returned if the
end of the stream has been reached. Raises exception ORA-24196 if the bytes
message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_long (id IN PLS_INTEGER) RETURN NUMBER**
Reads a long from the bytes message and returns the long read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_short (id IN PLS_INTEGER) RETURN PLS_INTEGER**
Reads a short value from the bytes message and returns the short value read. Null is returned if the end of the stream has been reached. Because there is no short type in PL/SQL, PLS_INTEGER is used to represent a SHORT. Although PL/SQL users get a PLS_INTEGER, they are guaranteed that the value is in the Java short value range. If this value is issued with a write_short function, then there will not be an out of range error. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_unsigned_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER**
Reads an unsigned 8-bit number from the bytes message stream and returns the next byte from the bytes message stream, interpreted as an unsigned 8-bit number. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_unsigned_short (id IN PLS_INTEGER) RETURN PLS_INTEGER**
Reads an unsigned 16-bit number from the bytes message stream and returns the next two bytes from the bytes message stream, interpreted as an unsigned 16-bit integer. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_utf (id IN PLS_INTEGER, value OUT NOCOPY CLOB)**
Reads a string that has been encoded using a UTF-8 format from the bytes message. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_boolean (id IN PLS_INTEGER, value IN BOOLEAN)**
Writes a Boolean to the bytes message stream as a 1-byte value. The value true is written as the value (byte)1. The value false is written as the value (byte)0. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_byte (id IN PLS_INTEGER, value IN PLS_INTEGER)**
Writes a byte to the bytes message. Because there is no BYTE type in PL/SQL, PLS_INTEGER is used to represent a BYTE. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_bytes (id IN PLS_INTEGER, value IN RAW)**
Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.
write_bytes (id IN PLS_INTEGER, value IN BLOB)
Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW, offset IN PLS_INTEGER, length IN PLS_INTEGER)
Writes a portion of a byte array to the bytes message stream. Parameter offset is the initial offset within the byte array. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBoundsException exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB, offset IN INT, length IN INT)
Writes a portion of a byte array to the bytes message stream. Parameter offset is the initial offset within the byte array. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBoundsException exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_char (id IN PLS_INTEGER, value IN CHAR)
Writes a character value to the bytes message. If this value has multiple characters, it is the first character that is written. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_double (id IN PLS_INTEGER, value IN DOUBLE PRECISION)
Writes a double to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_float (id IN PLS_INTEGER, value IN FLOAT)
Writes a float to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_int (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes an INT to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_long (id IN PLS_INTEGER, value IN NUMBER)
Writes a long to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_short (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a short to the bytes message as two bytes, high byte first. Because there is no short type in PL/SQL, INT is used to represent a short. Raises exception ORA-24193 if the parameter value exceeds the valid range, ORA-24196 if the bytes message is in
read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_utf (id IN PLS_INTEGER, value IN VARCHAR2)**

Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_utf (id IN PLS_INTEGER, value IN CLOB)**

Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

287.9.4 SYS.AQ$-_JMS_MAP_MESSAGE Type

This type is the ADT used to store a MapMessage in an Oracle Database Advanced Queuing queue.

This section contains these topics:

- **CONSTRUCT aq$_jms_map_message Function**
- **JMS Header Methods**
- **System Properties Methods**
- **User Properties Methods**
- **Payload Methods**

**Syntax**

```sql
TYPE aq$-_jms_map_message AS object(
    header aq$-_jms_header,
    bytes_len int,
    bytes_raw raw(2000),
    bytes_lob blob,
    STATIC FUNCTION construct RETURN aq$-_jms_map_message,
    MEMBER PROCEDURE set_replyto (replyto IN sys.aq$_agent),
    MEMBER PROCEDURE set_type     (type    IN VARCHAR),
    MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
    MEMBER FUNCTION get_type     RETURN VARCHAR,
    MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
    MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
    MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
    MEMBER PROCEDURE set_groupseq (groupseq IN INT),
    MEMBER FUNCTION get_userid   RETURN VARCHAR,
    MEMBER FUNCTION get_appid    RETURN VARCHAR,
    MEMBER FUNCTION get_groupid  RETURN VARCHAR,
    MEMBER FUNCTION get_groupseq RETURN INT,
    MEMBER PROCEDURE clear_properties,
    MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR, property_value IN BOOLEAN),
    MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR, property_value IN INT),
    MEMBER PROCEDURE set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION),
    MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR, property_value IN FLOAT),
);```
MEMBER FUNCTION get_double (id IN PLS_INTEGER, name IN VARCHAR2) RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float (id IN PLS_INTEGER, name IN VARCHAR2) RETURN FLOAT,
MEMBER FUNCTION get_int (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER,
MEMBER FUNCTION get_long (id IN PLS_INTEGER, name IN VARCHAR2) RETURN NUMBER,
MEMBER FUNCTION get_short (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER,
MEMBER PROCEDURE get_string (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY CLOB),
MEMBER FUNCTION get_names (id IN PLS_INTEGER) RETURN aq$_jms_namearray,
MEMBER FUNCTION get_names (id IN PLS_INTEGER, names OUT aq$_jms_namearray, offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE get_object (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY AQ$_JMS_VALUE),
MEMBER FUNCTION get_size (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION item_exists (id IN PLS_INTEGER, name IN VARCHAR2) RETURN BOOLEAN);

CONSTRUCT aq$_jms_map_message Function

STATIC FUNCTION construct RETURN aq$_jms_map_message
Creates an empty aq$_jms_map_message object.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

get_replyto RETURN sys.aq$_agent
Returns replyto, which corresponds to JMSReplyTo.

get_type RETURN VARCHAR
Returns type, which corresponds to JMSType.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets userid, which corresponds to JMSXUserID in JMS.

set_appid (appid IN VARCHAR)
Sets appid, which corresponds to JMSXAppID in JMS.

set_groupid (groupid IN VARCHAR)
Sets groupid, which corresponds to JMSXGroupID in JMS.

set_groupseq (groupseq IN INT)
Sets groupseq, which corresponds to JMSXGroupSeq in JMS.

get_userid RETURN VARCHAR
Returns userid, which corresponds to JMSXUserID.
get_appid RETURN VARCHAR
Returns appid, which corresponds to JMSAppID.

get_groupid RETURN VARCHAR
Returns groupid, which corresponds to JMSGroupID.

get_groupseq RETURN NUMBER
Returns groupseq, which corresponds to JMSGroupSeq.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether property_name is null or exists. If not, the procedure stores property_value in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the BYTE datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the INT datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether property_name is null or exists. If not, the procedure stores property_value. In PL/SQL and Oracle Database, the NUMBER datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -32768 to 32767 (16-bits). This check is necessary because
neither PL/SQL nor RDBMS defines the short datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

(set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a BOOLEAN property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a BYTE property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a DOUBLE property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a FLOAT property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a Integer property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a long property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a short property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a STRING property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

(set_bytes (payload IN RAW)
Sets the internal payload as a RAW variable without any interpretation. The payload of aq$_jms_map_message is stored as either RAW or BLOB in the database. This member function sets a payload as a RAW variable without interpreting it.

(set_bytes (payload IN BLOB)
Sets the internal payload as a BLOB variable without any interpretation. The payload of aq$_jms_map_message is stored as either RAW or BLOB in the database. This member function sets a payload as a BLOB variable without interpreting it.
get_bytes (payload OUT RAW)
Puts the internal payload into a RAW variable without any interpretation. The payload of \texttt{aq\$\_jms\_map\_message} is stored as either RAW or BLOB in the database. This member function gets a payload as raw bytes without interpreting it. Raises exceptions ORA-24190 if the length of internal payload is more than 32767.

get_bytes (payload OUT BLOB)
Puts the internal payload into a BLOB variable without any interpretation. The payload of \texttt{aq\$\_jms\_map\_message} is stored as either RAW or BLOB in the database. This member function gets a payload as a BLOB without interpreting it.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in \texttt{aq\$\_jms\_map\_message} and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter \texttt{id} is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If \texttt{id} is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID. This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to \texttt{aq\$\_jms\_map\_message}. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter \texttt{id} is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If \texttt{id} is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID. This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)
Closes and cleans up the DataInputStream or DataOutputStream at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

set_boolean (id IN PLS_INTEGER, name IN VARCHAR2, value IN BOOLEAN)
Sets the Boolean value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.
set_byte (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the BYTE value with the specified name in the map. Because there is no BYTE type in PL/SQL, PLS_INTEGER is used to represent a byte. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN RAW)
Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN RAW, offset IN INT, length IN INT)
Sets a portion of the byte array value with the specified name in the map. Parameter offset is the initial offset within the byte array, and parameter length is the number of bytes to use. If the range [offset … offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBoundsException exception is thrown in the Java stored procedure, and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN BLOB)
Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN BLOB, offset IN INT, length IN INT)
Sets a portion of the byte array value with the specified name in the map. Parameter offset is the initial offset within the byte array, and parameter length is the number of bytes to use. If the range [offset … offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBoundsException exception is thrown in the Java stored procedure, and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_char (id IN PLS_INTEGER, name IN VARCHAR2, value IN CHAR)
Sets the character value with the specified name in the map. If this value has multiple characters, then it is the first character that is used. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_double (id IN PLS_INTEGER, name IN VARCHAR2, value IN DOUBLE PRECISION)
Sets the double value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_float (id IN PLS_INTEGER, name IN VARCHAR2, value IN FLOAT)
This procedure is to set the float value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.
set_int (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the int value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_long (id IN PLS_INTEGER, name IN VARCHAR2, value IN NUMBER)
Sets the long value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_short (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the short value with the specified name in the map. Because there is no short type in PL/SQL, PLS_INTEGER is used to represent a short. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_string (id IN PLS_INTEGER, name IN VARCHAR2, value IN VARCHAR2)
Sets the string value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_boolean (id IN PLS_INTEGER, name IN VARCHAR2) RETURN BOOLEAN
Retrieves the Boolean value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_byte (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the BYTE value with the specified name. If there is no item by this name, then null is returned. Because there is no BYTE type in PL/SQL, PLS_INTEGER is used to represent a byte. Although the PL/SQL users get a PLS_INTEGER, they are guaranteed that the value is in the Java BYTE value range. If this value is issued with a set_byte function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY BLOB)
Retrieves the byte array value with the specified name. If there is no item by this name, then null is returned. Because the size of the array might be larger than the limit of PL/SQL RAW type, a BLOB is always returned here. The BLOB returned is a copy, which means it can be modified without affecting the message payload. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.
get_char (id IN PLS_INTEGER, name IN VARCHAR2) RETURN CHAR
Retrieves and returns the character value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

get_double (id IN PLS_INTEGER, name IN VARCHAR2) RETURN DOUBLE PRECISION
Retrieves and returns the double value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

get_float (id IN PLS_INTEGER, name IN VARCHAR2) RETURN FLOAT
Retrieves the float value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_int (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the INT value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_long (id IN PLS_INTEGER, name IN VARCHAR2) RETURN NUMBER
Retrieves the long value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_short (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the short value with the specified name. If there is no item by this name, then null is returned. Because there is no short type in PL/SQL, INT is used to represent a short. Although the PL/SQL users get an PLS_INTEGER, they are guaranteed that the value is in the Java short value range. If this value is issued with a set_short function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_string (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY CLOB)
Retrieves the string value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_names (id IN PLS_INTEGER) RETURN aq$_jms_namearray
Retrieves all the names within the map message and returns them in a varray. Because aq$_jms_namearray has a size as 1024 and each element is a VARCHAR(200), this function will return an error if the size of the name array of the payload exceeds the limit. Raises exception ORA-24195 if the size of the name array or the size of a name exceeds the limit.

get_names (id IN PLS_INTEGER, names OUT aq$_jms_namearray, offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER
Retrieves a portion of the names within the map message. Because aq$_jms_namearray has a size as 1024 and each element is a VARCHAR(200), this function will return an error if either limits are exceeded during the retrieval. (This means there is no sense to put a length
parameter greater than 1024.) The index of the names of a map messages begins from 0. Parameter offset is the offset from which to start retrieving. The function returns the number of names that have been retrieved. The names retrieved is the intersection of the interval [offset, offset+length-1] and interval [0, size-1] where size is the size of this map message. If the intersection is an empty set, then names will be returned as null and the function returns 0 as the number of names retrieved. If users iterate the names by retrieving in small steps, then this can be used to test that there are no more names to read from map message. Raises exception ORA-24195 if the size of the name array or the size of a name exceed the limit, ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_object (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY AQ$_JMS_VALUE)
Returns a general value ADT AQ$_JMS_VALUE. If there is no item by this name, then null is returned. Users can use the type attribute of this ADT to interpret the data. See the map in the AQ$_JMS_VALUE ADT for the correspondence among dbms_jms_plsql package constants, Java datatype and AQ$_JMS_VALUE attribute. Note this member procedure might bring additional overhead compared to other get member procedures or functions. It is used only if the user does not know the datatype of the fields within a message before hand. Otherwise it is a good idea to use a specific get member procedure or function. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_size (id IN PLS_INTEGER) RETURN PLS_INTEGER
Retrieves the size of the map message. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

item_exists (id IN PLS_INTEGER, name IN VARCHAR2) RETURN BOOLEAN
Indicates that an item exists in this map message by returning TRUE. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

287.9.5 SYS.AQ$_JMS_STREAM_MESSAGE Type

This type is the ADT used to store a StreamMessage in an Oracle Database Advanced Queuing queue.

This section contains these topics:

- **CONSTRUCT aq$ _jms_stream_message Function**
- **JMS Header Methods**
- **System Properties Methods**
- **User Properties Methods**
- **Payload Methods**

**Syntax**

```sql
TYPE aq$_jms_stream_message AS object(
    header     aq$_jms_header,
    bytes_len  int,
    bytes_raw  raw(2000),
```

ORACLE
Chapter 287
Summary of JMS Types

bytes_lob blob,
STATIC FUNCTION construct RETURN aq$_jms_stream_message,
MEMBER PROCEDURE set_replyto (replyto IN sys.aq$_agent),
MEMBER PROCEDURE set_type (type IN VARCHAR),
MEMBER FUNCTION get_replyto RETURN sys.aq$_agent,
MEMBER FUNCTION get_type RETURN VARCHAR,
MEMBER PROCEDURE set_userid (userid IN VARCHAR),
MEMBER PROCEDURE set_appid (appid IN VARCHAR),
MEMBER PROCEDURE set_groupid (groupid IN VARCHAR),
MEMBER PROCEDURE set_groupseq (groupseq IN INT),
MEMBER FUNCTION get_userid RETURN VARCHAR,
MEMBER FUNCTION get_appid RETURN VARCHAR,
MEMBER FUNCTION get_groupid RETURN VARCHAR,
MEMBER FUNCTION get_groupseq RETURN INT,
MEMBER PROCEDURE clear_properties,
MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR, property_value IN BOOLEAN),
MEMBER PROCEDURE set_byte_property (property_name IN VARCHAR, property_value IN INT),
MEMBER PROCEDURE set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION),
MEMBER PROCEDURE set_float_property (property_name IN VARCHAR, property_value IN FLOAT),
MEMBER PROCEDURE set_int_property (property_name IN VARCHAR, property_value IN INT),
MEMBER PROCEDURE set_long_property (property_name IN VARCHAR, property_value IN NUMBER),
MEMBER PROCEDURE set_short_property (property_name IN VARCHAR, property_value IN INT),
MEMBER PROCEDURE set_string_property (property_name IN VARCHAR, property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_bytes (payload IN RAW),
MEMBER PROCEDURE set_bytes (payload IN BLOB),
MEMBER PROCEDURE get_bytes (payload OUT RAW),
MEMBER PROCEDURE get_bytes (payload OUT BLOB),
MEMBER FUNCTION prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE reset (id IN PLS_INTEGER),
MEMBER PROCEDURE flush (id IN PLS_INTEGER),
MEMBER PROCEDURE clean_body (id IN PLS_INTEGER),
MEMBER PROCEDURE clean (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER FUNCTION get_mode (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_boolean (id IN PLS_INTEGER) RETURN BOOLEAN,
MEMBER FUNCTION read_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_bytes (id IN PLS_INTEGER) RETURN BLOB,
MEMBER FUNCTION read_bytes (id IN PLS_INTEGER, value OUT NOCOPY BLOB),
MEMBER FUNCTION read_char (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION read_double (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION read_float (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION read_int (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_long (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION read_short (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_string RETURN CLOB,
MEMBER PROCEDURE read_string  (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE read_object    (id IN PLS_INTEGER,
                              value OUT NOCOPY AQ$_JMS_VALUE),
MEMBER PROCEDURE write_boolean  (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte     (id IN PLS_INTEGER, value IN INT),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN RAW,
                              offset IN INT, length IN INT),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN BLOB,
                              offset IN INT, length IN INT),
MEMBER PROCEDURE write_char     (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double   (id IN PLS_INTEGER, value IN DOUBLE PRECISION),
MEMBER PROCEDURE write_int      (id IN PLS_INTEGER, value IN NUMBER),
MEMBER PROCEDURE write_short    (id IN PLS_INTEGER, value IN INT),
MEMBER PROCEDURE write_string   (id IN PLS_INTEGER, value IN VARCHAR2),
MEMBER PROCEDURE write_string   (id IN PLS_INTEGER, value IN CLOB));

CONSTRUCT aq$_jms_stream_message Function

STATIC FUNCTION construct RETURN aq$_jms_stream_message
Creates an empty aq$_jms_stream_message object.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to JMSType in JMS.

get_replyto RETURN sys.aq$_agent
Returns replyto, which corresponds to JMSReplyTo.

get_type RETURN VARCHAR
Returns type, which corresponds to JMSType.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets userid, which corresponds to JMSXUserID in JMS.

set_appid (appid IN VARCHAR)
Sets appid, which corresponds to JMSXAppID in JMS.

set_groupid (groupid IN VARCHAR)
Sets groupid, which corresponds to JMSXGroupID in JMS.

set_groupseq (groupseq IN INT)
Sets groupseq, which corresponds to JMSXGroupSeq in JMS.

get_userid RETURN VARCHAR
Returns userid, which corresponds to JMSXUserID.
get_appid RETURN VARCHAR
Returns appid, which corresponds to JMSXAppID.

get_groupid RETURN VARCHAR
Returns groupid, which corresponds to JMSXGroupID.

get_groupseq RETURN NUMBER
Returns groupseq, which corresponds to JMSXGroupSeq.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether property_name is null or exists. If not, the procedure stores property_value in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the BYTE datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether property_name is null or exists. If not, the procedure stores property_value. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the INT datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether property_name is null or exists. If not, the procedure stores property_value. In PL/SQL and Oracle Database, the NUMBER datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If not, the procedure checks whether property_value is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the short datatype. Raises exception ORA-24191 if the
property name exists, ORA-24192 if the property name is null, or ORA-24193 if the
property value exceeds the valid range.

**set_string_property** (property_name IN VARCHAR, property_value IN
VARCHAR)
Checks whether property_name is null or exists. If not, the procedure stores
property_value. Raises exception ORA-24191 if the property name exists or
ORA-24192 if the property name is null.

**get_boolean_property** (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a
BOOLEAN property, then this function returns the value of the property. Otherwise it
returns a null.

**get_byte_property** (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a
BYTE property, then this function returns the value of the property. Otherwise it returns
a null.

**get_double_property** (property_name IN VARCHAR) RETURN DOUBLE
PRECISION
If the property with the corresponding property name passed in exists, and if it is a
DOUBLE property, then this function returns the value of the property. Otherwise it
returns a null.

**get_float_property** (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a
FLOAT property, then this function returns the value of the property. Otherwise it returns
a null.

**get_int_property** (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a
Integer property, then this function returns the value of the property. Otherwise it returns
a null.

**get_long_property** (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a
long property, then this function returns the value of the property. Otherwise it returns
a null.

**get_short_property** (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a
short property, then this function returns the value of the property. Otherwise it returns
a null.

**get_string_property** (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a
STRING property, then this function returns the value of the property. Otherwise it returns
a null.

**Payload Methods**

**get_bytes** (payload OUT RAW)
Puts the internal payload into a RAW variable without any interpretation. The payload
of type **aq$_jms_stream_message** is stored as either **RAW** or **BLOB** in the database. This
member function gets a payload as raw bytes without interpreting it. Raises exception ORA-24190 if the length of internal payload is more than 32767.

get_bytes (payload OUT BLOB)
Puts the internal payload into a BLOB variable without any interpretation. The payload of type $jms_stream_message is stored as either RAW or BLOB in the database. This member function gets a payload as a BLOB variable without interpreting it.

set_bytes (payload IN RAW)
Sets the internal payload as the RAW variable without any interpretation. The payload of type $jms_stream_message is stored as either RAW or BLOB in the database. This member function sets a payload as raw bytes without interpreting it.

set_bytes (payload IN BLOB)
Sets the internal payload as the BLOB variable without any interpretation. The payload of type $jms_stream_message is stored as either RAW or BLOB in the database. This member function sets a payload as a BLOB variable without interpreting it.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in $jms_stream_message and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter id is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If id is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
This function also sets the message access mode to MESSAGE_ACCESS_READONLY. Subsequent calls of write_XXX procedure raise an ORA-24196 error. Users can call the clear_body procedure to set the message access mode to MESSAGE_ACCESS_READONLY. This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

reset (id IN PLS_INTEGER)
Resets the starting position of the stream to the beginning and puts the stream message in MESSAGE_ACCESS_READONLY mode.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to $jms_stream_message. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter id is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If id is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
It also sets the message access mode to MESSAGE_ACCESS_WRITEONLY. Later calls of read_XXX procedure raise ORA-24196 error. Users can call the reset or prepare procedures to set the message access mode to MESSAGE_ACCESS_READONLY. Write-only and read-only modes affect only the payload functions of AQ$JMS_BYTES_MESSAGE. They do not affect the header functions.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

**clean (id IN PLS_INTEGER)**
Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**clean_all**
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

**get_mode (id IN PLS_INTEGER) RETURN PLS_INTEGER**
Returns the current mode of this message. The return value is either `SYS.dbms_aqjms.READ_ONLY` or `SYS.dbms_aqjms.WRITE_ONLY`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**read_boolean (id IN PLS_INTEGER) RETURN BOOLEAN**
Reads and returns a Boolean value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER**
Reads and returns a byte value from the stream message. If the end of the message stream has been reached, then null is returned. Because there is no `BYTE` type in PL/SQL, `INT` is used to represent a byte. Although PL/SQL users get an `INT`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `write_byte` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_bytes (id IN PLS_INTEGER) RETURN BLOB**
Reads and returns a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

**read_bytes (id IN PLS_INTEGER, value OUT NOCOPY BLOB)**
Reads a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.
read_char (id IN PLS_INTEGER) RETURN CHAR
Reads and returns a character value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_double (id IN PLS_INTEGER) RETURN DOUBLE PRECISION
Reads and returns a double from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_float (id IN PLS_INTEGER) RETURN FLOAT
Reads and returns a float from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_int (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns an INT from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_long (id IN PLS_INTEGER) RETURN NUMBER
Reads and returns a long from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_short (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns a short value from the stream message. Because there is no short type in PL/SQL, PLS_INTEGER is used to represent a SHORT. Although PL/SQL users get an INT, they are guaranteed that the value is in the Java short value range. If this value is issued with a write_short function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_string RETURN CLOB
Reads and returns a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

read_string (id IN PLS_INTEGER, value OUT NOCOPY CLOB)
Reads a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between
the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read_object (id IN PLS_INTEGER, value OUT NOCOPY AQ$_JMS_VALUE)**

Returns a general value ADT AQ$_JMS_VALUE. Users can use the type attribute of this ADT to interpret the data. See Table 287-2 for the correspondence among dbms_jms_plsql package constants, Java datatype and AQ$_JMS_VALUE attribute. This member procedure might bring additional overhead compared to other read member procedures or functions. It is used only if the user does not know the datatype of the fields within a message beforehand. Otherwise it is a good idea to use a specific read member procedure or function.

Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_boolean (id IN PLS_INTEGER, value IN BOOLEAN)**

Writes a Boolean to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_byte (id IN PLS_INTEGER, value IN INT)**

Writes a byte to the stream message. Because there is no BYTE type in PL/SQL, INT is used to represent a byte. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_bytes (id IN PLS_INTEGER, value IN RAW)**

Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_bytes (id IN PLS_INTEGER, value IN RAW, offset IN INT, length IN INT)**

Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter offset is the initial offset within the byte array, and parameter length is the number of bytes to use. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBoundsException exception is thrown in the Java stored procedure. The index starts from 0.

Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write_bytes (id IN PLS_INTEGER, value IN BLOB)**

Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.
write_bytes (id IN PLS_INTEGER, value IN BLOB, offset IN INT, length IN INT)
Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter offset is the initial offset within the byte array, and parameter length is the number of bytes to use. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBoundsException exception is thrown in the Java stored procedure. The index starts from 0. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_char (id IN PLS_INTEGER, value IN CHAR)
Writes a character value to the stream message. If this value has multiple characters, then it is the first character that is written. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_double (id IN PLS_INTEGER, value IN DOUBLE PRECISION)
Writes a double to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_float (id IN PLS_INTEGER, value IN FLOAT)
Writes a float to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_int (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes an INT to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_long (id IN PLS_INTEGER, value IN NUMBER)
Writes a long to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_short (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a short to the stream message. Because there is no short type in PL/SQL, INT is used to represent a short. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_string (id IN PLS_INTEGER, value IN VARCHAR2)
Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_string (id IN PLS_INTEGER, value IN CLOB)
Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.
287.9.6 SYS.AQ$_JMS_OBJECT_MESSAGE Type

This type is the ADT used to store an ObjectMessage in an Oracle Database Advanced Queuing queue.

Syntax

```sql
TYPE aq$_jms_object_message AS object(
    header   aq$_jms_header,
    bytes_len int,
    bytes_raw raw(2000),
    bytes_lob blob);
```

287.9.7 SYS.AQ$_JMS_NAMEARRAY Type

This type represents the name array returned by the `get_names` procedure of `aq$_jms_map_message`.

The maximum number of names this type can hold is 1024. The maximum length of each name is 200 characters.

Syntax

```sql
CREATE OR REPLACE TYPE AQ$_JMS_NAMEARRAY AS VARRAY(1024) OF VARCHAR(100);
```

Usage Notes

If the names array in the message payload is greater than 1024, then use the following function to retrieve the names in multiple portions:

```sql
MEMBER FUNCTION get_names(id IN PLS_INTEGER, names OUT aq$_jms_namearray,
    offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER;
```

287.9.8 SYS.AQ$_JMS_VALUE Type

This type represents the general data returned by the `get_object` procedure of `aq$_jms_map_message` and the `read_object` procedure of `aq$_jms_stream_message`.

The `type` field in this ADT is used to decide which type of data this object is really holding. The following table lists the mapping between the `sys.dbms_jms_plsql` type constants, the corresponding Java type, and the data field of ADT `aq$_jms_value` which effectively holds the data.

Syntax

```sql
CREATE OR REPLACE TYPE AQ$_JMS_VALUE AS object(
    type      number(2),
    num_val   number,
    char_val  char(1),
    text_val  clob,
    bytes_val blob);
```
Table 287-2  AQ$_JMS_VALUE Type Fields and Java Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Java Type</th>
<th>aq$_jms_value Data Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_BYTE</td>
<td>byte</td>
<td>num_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_SHORT</td>
<td>short</td>
<td>num_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_INTEGER</td>
<td>int</td>
<td>num_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_LONG</td>
<td>long</td>
<td>num_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_FLOAT</td>
<td>float</td>
<td>num_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_DOUBLE</td>
<td>double</td>
<td>num_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_BOOLEAN</td>
<td>boolean</td>
<td>num_val: 0 FALSE, 1 TRUE</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_CHARACTER</td>
<td>char</td>
<td>char_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_STRING</td>
<td>java.lang.String</td>
<td>text_val</td>
</tr>
<tr>
<td>DBMS_JMS_PLSQL.DATA_TYPE_BYTES</td>
<td>byte[]</td>
<td>bytes_val</td>
</tr>
</tbody>
</table>

287.9.9 SYS.AQ$_JMS_EXCEPTION Type

This type represents a Java exception thrown on the Java stored procedure side.

The id field is reserved for future use. The exp_name stores the Java exception name, the err_msg field stores the Java exception error message, and the stack field stores the stack trace of the Java exception.

Syntax

```sql
CREATE OR REPLACE TYPE AQ$_JMS_EXCEPTION AS OBJECT (  
id          number, -- Reserved and not used. Right now always return 0.  
exp_name    varchar(200),  
err_msg     varchar(500),  
stack       varchar(4000));
```
JSON Data Structures

To work with JSON data in PL/SQL, you can use these data structures.

- JSON_ELEMENT_T Object Type
- JSON_OBJECT_T Object Type
- JSON_ARRAY_T Object Type
- JSON_SCALAR_T Object Type
- JSON_KEY_LIST Type

Related Topics
- Oracle Database JSON Developer’s Guide
- The JSON standard

288.1 JSON_ELEMENT_T Object Type

JSON_ELEMENT_T is the supertype for the JSON_OBJECT_T, JSON_SCALAR_T, and JSON_ARRAY_T object types.

Description

Note the following:

- To create an instance of JSON_ELEMENT_T, use the parse function. See the “Constructors” section below for details.
- You cannot create an empty JSON_ELEMENT_T instance. To create an empty JSON container, create it based on one of the subtypes.
- To cast a JSON_ELEMENT_T into a subtype (for example, JSON_OBJECT_T), you need to perform an explicit cast using TREAT AS. For example:

  TREAT (elem AS JSON_OBJECT_T)

Constructors

You create a JSON_ELEMENT_T instance using the parse function. The function takes as input VARCHAR2, CLOB, or BLOB data, and returns a JSON_ELEMENT_T instance.

```
STATIC FUNCTION parse(json VARCHAR2) RETURN JSON_ELEMENT_T
STATIC FUNCTION parse(json CLOB) RETURN JSON_ELEMENT_T
STATIC FUNCTION parse(json BLOB) RETURN JSON_ELEMENT_T
```

Only UTF8-encoded JSON is passed as a BLOB.

The parse function takes a JSON string as input and sets up an internal representation of the JSON data. If the provided input is not valid JSON, then an error message is raised. Valid JSON has to pass the lax check of the “IS JSON” SQL condition.
Serialization

Serialization is the inverse of the parse function. The serialization operation takes the in-memory representation of the JSON data and prints it to a string. The serialization functions and procedures are:

- MEMBER FUNCTION to_String    RETURN VARCHAR2
- MEMBER FUNCTION to_Number    RETURN NUMBER
- MEMBER FUNCTION to_Date      RETURN DATE
- MEMBER FUNCTION to_Timestamp RETURN TIMESTAMP
- MEMBER FUNCTION to_Boolean   RETURN BOOLEAN
- MEMBER FUNCTION to_Clob      RETURN CLOB
- MEMBER FUNCTION to_Blob      RETURN BLOB

- MEMBER PROCEDURE to_Clob(c IN OUT CLOB)
- MEMBER PROCEDURE to_Blob(c IN OUT BLOB)

The to_Clob and to_Blob procedures accept a CLOB or BLOB input and enable you to provide a LOB to be used as the serialization destination. For instance, EMPTY_LOB can be provided. The input LOB cannot be NULL.

If the to_Clob function is used, then a new CLOB is created. If you do not want to create a CLOB first, then you can use the to_Clob or to_Blob functions, which take no parameter and generate a temp lob.

to_Blob serializes to UTF8 format only.

Introspection

Introspection enables you to discover properties of JSON objects without modifying them. The introspection functions are:

- MEMBER FUNCTION is_Object    RETURN BOOLEAN
- MEMBER FUNCTION is_Array     RETURN BOOLEAN
- MEMBER FUNCTION is_Scalar    RETURN BOOLEAN
- MEMBER FUNCTION is_String    RETURN BOOLEAN
- MEMBER FUNCTION is_Number    RETURN BOOLEAN
- MEMBER FUNCTION is_Boolean   RETURN BOOLEAN
- MEMBER FUNCTION is_True      RETURN BOOLEAN
- MEMBER FUNCTION is_False     RETURN BOOLEAN
- MEMBER FUNCTION is_Null      RETURN BOOLEAN
- MEMBER FUNCTION is_Date      RETURN BOOLEAN
- MEMBER FUNCTION is_Timestamp RETURN BOOLEAN
- MEMBER FUNCTION get_Size     RETURN NUMBER

The return value of the get_size function depends on the JSON type:

- For a scalar, it returns 1.
- For an object, it returns the number of keys.
- For an array, it returns the number of items.

Note that JSON does not support dates and timestamps natively. Instead they are typically modeled as strings. The DOM API enables you to add dates and timestamps as scalar values and preserve them until serialization to JSON, where they are printed as string following the ISO 8601 format. If a SQL value of type date or timestamp has been added, then the is_Date and is_Timestamp functions return true. If a date has been added as a string (e.g. as ISO 8601), then the is_Date and is_Timestamp functions return false. You can use Oracle conversion function to_Date and
toTimestamp to convert a string representation of a date, and timestamp to the Oracle representation.

Error Handling

You can set the level of error handling for JSON processing. You might not want an error to be raised for every mismatch. The on_Error procedure enables you to specify when errors should be raised:

```sql
MEMBER PROCEDURE on_Error(value NUMBER)
```

The on_Error procedure defines what happens if an error is encountered during a PL/SQL operation, for instance, a get call.

The default is to not raise an error but to return NULL instead.

You invoke on_error on a JSON_ELEMENT_T instance and it sets the error behavior for all subsequent calls. To reset the behavior to the default, you can call on_Error(0).

Values for the value parameter are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset to the default behavior, which is to return NULL instead of raising an error.</td>
</tr>
<tr>
<td>1</td>
<td>Raise all errors.</td>
</tr>
<tr>
<td>2</td>
<td>Raise an error if no value is detected.</td>
</tr>
<tr>
<td>3</td>
<td>Raise an error if the data types do not match, for example, if you call GET_NUMBER on a string value.</td>
</tr>
<tr>
<td>4</td>
<td>Raise an error if the input is invalid, for example, if the array is out of bounds.</td>
</tr>
</tbody>
</table>

You can combine values. For example, you can specify 7 to indicate a combination of 3 and 4.

In the following example, an error is raised because the value of “a” is “xyz”, which cannot be converted to a number. If the on_Error procedure had not been called, then NULL would be returned, and no error would be raised.

```sql
declare
    jo JSON_OBJECT_T;
begin
    jo := JSON_OBJECT_T.parse('{}a:"xyz"');
    jo.On_error(1);
    dbms_output.put_line(jo.get_Number('a'));
end;
/
```

288.2 JSON_OBJECT_T Object Type

JSON_OBJECT_T is a subtype of the JSON_ELEMENT_T object type. It corresponds to the JSON object structure.

Constructors

You can create an empty JSON_OBJECT_T instance using the following constructor.
CONSTRUCTOR FUNCTION JSON_OBJECT_T RETURN SELF AS RESULT

You can create a JSON_OBJECT_T instance using one of the following parse functions:

STATIC FUNCTION parse(json VARCHAR2) RETURN JSON_OBJECT_T
STATIC FUNCTION parse(json CLOB) RETURN JSON_OBJECT_T
STATIC FUNCTION parse(json BLOB) RETURN JSON_OBJECT_T

You can also create a JSON_OBJECT_T instance using one of the following constructors:

CONSTRUCTOR FUNCTION JSON_OBJECT_T(json VARCHAR2) RETURN SELF AS RESULT,
CONSTRUCTOR FUNCTION JSON_OBJECT_T(json CLOB) RETURN SELF AS RESULT,
CONSTRUCTOR FUNCTION JSON_OBJECT_T(json BLOB) RETURN SELF AS RESULT,
CONSTRUCTOR FUNCTION JSON_OBJECT_T(json JSON_ELEMENT_T) RETURN SELF AS RESULT

Only UTF8-encoded JSON is passed as a BLOB.

The parse function takes a JSON string as input and sets up an internal representation of the JSON data. If the provided input is not a valid JSON object, then an error message is raised. The input has to specify a JSON object, not an array.

Get Functions and Procedures

The following functions and procedures enable you to retrieve the value of the JSON object:

MEMBER FUNCTION get(key VARCHAR2) RETURN JSON_ELEMENT_T
MEMBER FUNCTION get_String(key VARCHAR2) RETURN VARCHAR2
MEMBER FUNCTION get_Number(key VARCHAR2) RETURN NUMBER
MEMBER FUNCTION get_Date(key VARCHAR2) RETURN DATE
MEMBER FUNCTION get_Timestamp(key VARCHAR2) RETURN TIMESTAMP
MEMBER FUNCTION get_Boolean(key VARCHAR2) RETURN BOOLEAN
MEMBER FUNCTION get_Clob(key VARCHAR2) RETURN CLOB
MEMBER FUNCTION get_Blob(key VARCHAR2) RETURN BLOB
MEMBER FUNCTION get_Object(key VARCHAR2) RETURN JSON_OBJECT_T
MEMBER FUNCTION get_Array(key VARCHAR2) RETURN JSON_ARRAY_T
MEMBER PROCEDURE get_Clob(key NUMBER, c IN OUT CLOB)
MEMBER PROCEDURE get_Blob(key NUMBER, c IN OUT BLOB)

Note:

- The get function has reference semantics. This means that if the returned JSON_ELEMENT_T is modified, then the containing JSON_ELEMENT_T is also changed. See the Reference Semantics section below for details.

- The GET_STRING function converts the value to a string if it is not already a string. Thus, the GET_STRING function returns a non-null value even if IS_STRING returns false.

- All the "get" functions perform a conversion if possible. If no conversion is possible, then an error might be raised, depending on what ON_ERROR is set to.

The GET_CLOB and GET_BLOB procedures, which accept a CLOB or BLOB as input, enable you to provide a LOB to be used as serialization destination. For instance, EMPTY_LOB can be provided. If you use the GET_CLOB function instead, then a new CLOB will be created implicitly. The input LOB cannot be NULL. If you do not want to create a BLOB or CLOB first, you can use the GET_CLOB or GET_BLOB functions, which take no parameter and generates a temp LOB.
Chapter 288

JSON_OBJECT_T Object Type

GET_BLOB serializes to UTF8 format only.
Set Procedures
The following procedures enable you to set the value of a JSON object. Existing values, if
present, will be overwritten.
MEMBER
MEMBER
MEMBER
MEMBER
MEMBER
MEMBER
MEMBER

PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE

put(key VARCHAR2, value
put(key VARCHAR2, value
put(key VARCHAR2, value
put(key VARCHAR2, value
put(key VARCHAR2, value
put(key VARCHAR2, value
put_Null(key VARCHAR2)

JSON_ELEMENT_T)
VARCHAR2)
NUMBER)
BOOLEAN)
DATE)
TIMESTAMP)

Introspection Functions
Introspection enables you to discover properties of JSON objects without modifying them.
The introspection functions are:
MEMBER FUNCTION has(key VARCHAR2)
RETURN BOOLEAN
MEMBER FUNCTION get_Type(key VARCHAR2) RETURN VARCHAR2
MEMBER FUNCTION get_Keys
RETURN JSON_KEY_LIST

The get_Keys function returns an object type of JSON_KEY_LIST, which is a varray of
VARCHAR2(4000). The varray contains the names of keys in the JSON object. The get_Keys
function returns at most 32767 field names for a given JSON object. An error is raised if it is
applied to an object with more than 32767 fields.
The example below walks through the items of the varray to build a JSON_ARRAY_T object
that contains all key names.
declare
jo JSON_OBJECT_T;
ja JSON_ARRAY_T;
keys JSON_KEY_LIST;
keys_string VARCHAR2(100);
begin
ja := new JSON_ARRAY_T;
jo := JSON_OBJECT_T.parse('{"name":"fred",
"jobTitle":"codemonkey",
"projects":["json", "xml"]}');
keys := jo.get_keys;
for i in 1..keys.count loop
ja.append(keys(i));
end loop;
keys_string := ja.to_string;
dbms_output.put_line(keys_string);
end;
/

The output is:
["name","jobTitle","projects"]

Modification Procedures
The following procedures enable you to remove or rename keys in a JSON object.

288-5


MEMBER PROCEDURE remove(key VARCHAR2)
MEMBER PROCEDURE rename_Key(keyOld VARCHAR2, keyNew VARCHAR2)

Duplicate key names are not supported and will raise an error.

Clone Function

This function makes a copy of the JSON object. Reference semantics is changed to value semantics.

MEMBER FUNCTION clone RETURN JSON_OBJECT_T

Reference Semantics

Calling the get function that returns a JSON_ELEMENT_T object always returns a reference to the complex values instead of a copy. This means that changing the returned value affects its container. See the example below.

declare
    data    JSON_OBJECT_T;
    address JSON_OBJECT_T;
    zip     number;
begin
    data := new JSON_OBJECT_T('{
        "first": "John",
        "last": "Doe",
        "address": {
            "country": "USA",
            "zip": "94065"
        }
    }');
    address := data.get_object('address');
    dbms_output.put_line(address.to_string);
    -- 1) VALUE SEMANTICS for scalar values
    -- (changing the value has no effect on container)
    zip := address.get_number('zip');
    dbms_output.put_line(zip);
    zip := 12345;
    dbms_output.put_line(zip);
    -- address is still the same
    dbms_output.put_line(address.to_string);

    -- 2) REFERENCE SEMANTICS for complex values
    -- 'address' is a reference to the complex address values inside 'data'
    address.put('zip', 12345);
    address.put('street', 'Detour Road');
    dbms_output.put_line(data.to_string);
end;
/

In cases where you do not want the reference semantics, you can use the clone function to create a copy of the returned object. This decouples the value from its container. In the example above, you can create a copy of the "address" object by replacing this line:

address := data.get_object('address');
with this line:

```
address := data.get_object('address').clone;
```

After this, changing the address with have no effect on the value of the ‘data’ containing object.

Update Example

The following example updates the price of an item by 10%.

```
WITH
FUNCTION updatePrice(jsonTxt in VARCHAR2 ) RETURN VARCHAR2 IS
  jo JSON_OBJECT_T;
  oldPrice NUMBER;
BEGIN
  jo := new JSON_OBJECT_T(jsonTxt);
  oldPrice := jo.get_number('price');
  jo.put('price', oldPrice * 1.1);
  RETURN jo.to_string();
END;
```

SELECT updatePrice(col) FROM t1;

288.3 JSON_ARRAY_T Object Type

JSON_ARRAY_T is a subtype of the JSON_ELEMENT_T object type. JSON_ARRAY_T corresponds to the JSON array structure.

Constructors

You can create an empty JSON_ARRAY_T instance using the following constructor.

```
CONSTRUCTOR FUNCTION JSON_ARRAY_T RETURN SELF AS RESULT
```

You can create a JSON_ARRAY_T instance using one the following parse functions:

```
STATIC FUNCTION parse(json VARCHAR2) RETURN JSON_ARRAY_T
STATIC FUNCTION parse(json CLOB)     RETURN JSON_ARRAY_T
STATIC FUNCTION parse(json BLOB)     RETURN JSON_ARRAY_T
```

You can also create a JSON_ARRAY_T instance using one the following constructors:

```
CONSTRUCTOR FUNCTION JSON_Array_T(jsn VARCHAR2) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_Array_T(jsn CLOB)     RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_Array_T(jsn BLOB)     RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_Array_T(e JSON_ELEMENT_T) RETURN SELF AS RESULT
```

Only UTF8-encoded JSON is passed as a BLOB.

The parse function takes a JSON string as input and sets up an internal representation of the JSON data. If the provided input is not valid JSON, then an error message is raised. The input has to specify a JSON array, not an object.

Get Functions and Procedures

The following functions and procedures enable you to retrieve the value of the JSON array:
MEMBER FUNCTION get(pos NUMBER) RETURN JSON_ELEMENT_T
MEMBER FUNCTION get_String(pos NUMBER) RETURN VARCHAR2
MEMBER FUNCTION get_Number(pos NUMBER) RETURN NUMBER
MEMBER FUNCTION get_Date(pos NUMBER) RETURN DATE
MEMBER FUNCTION get_Timestamp(pos NUMBER) RETURN TIMESTAMP
MEMBER FUNCTION get_Boolean(pos NUMBER) RETURN BOOLEAN
MEMBER FUNCTION get_Clob(pos NUMBER) RETURN CLOB
MEMBER FUNCTION get_Blob(pos NUMBER) RETURN BLOB

MEMBER PROCEDURE get_Clob(pos NUMBER, c IN OUT CLOB)
MEMBER PROCEDURE get_Blob(pos NUMBER, c IN OUT BLOB)

Note:

- The get function has reference semantics. This means that if the returned JSON_ELEMENT_T is modified, then the containing JSON_ELEMENT_T is changed too.
- The GET_STRING function converts the value to a string if it is not already a string. The function returns a non-null value even if IS_STRING returns false.
- All the “get” functions perform a conversion if possible. If no conversion is possible, then an error might be raised, depending on what ON_ERROR is set to.

The GET_CLOB and GET_BLOB procedures, which accept a CLOB or BLOB as input, enable you to provide a LOB to be used as serialization destination. For instance, EMPTY_LOB can be provided. If you use the GET_CLOB function instead, then a new CLOB will be created implicitly. The input LOB cannot be NULL. If you do not want to create a BLOB or CLOB first, you can use the GET_CLOB or GET_BLOB functions, which take no parameter and generates a temp LOB.

GET_BLOB serializes to UTF8 format only.

Set Procedures

The following procedures enable you to set the value at the specified position in the JSON array. These procedures insert (not overwrite) at the specified position unless overwrite is requested.

MEMBER PROCEDURE put(pos NUMBER, value VARCHAR2, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, value NUMBER, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, value BOOLEAN, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, value JSON_ELEMENT_T, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, value DATE, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, value TIMESTAMP, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put_Null(pos NUMBER, overwrite BOOLEAN DEFAULT FALSE)

The following procedures append the specified value to the end of the JSON array:

MEMBER PROCEDURE append(value JSON_ELEMENT_T)
MEMBER PROCEDURE append(value VARCHAR2)
MEMBER PROCEDURE append(value NUMBER)
MEMBER PROCEDURE append(value BOOLEAN)
MEMBER PROCEDURE append(value DATE)
MEMBER PROCEDURE append(value TIMESTAMP)
MEMBER PROCEDURE append_Null
Introspection Function
Introspection enables you to discover properties of the JSON array without modifying them.

```sql
MEMBER FUNCTION get_Type(pos NUMBER) RETURN VARCHAR2
```

Modification Procedure
The following procedure enables you to remove the value at the specified position in the JSON array.

```sql
MEMBER PROCEDURE remove(pos NUMBER)
```

Clone Function
This function makes a copy of the JSON array. Reference semantics is changed to value semantics.

```sql
MEMBER FUNCTION clone RETURN JSON_ARRAY_T
```

### 288.4 JSON_SCALAR_T Object Type

**JSON_SCALAR_T** is a subtype of the **JSON_ELEMENT_T** object type.

**Description**

A **JSON_SCALAR_T** instance captures one scalar value, for example, the string “fred” or the number 1. This type does not have any functions or procedures other than the ones inherited from **JSON_ELEMENT_T**. You cannot create an instance of this type directly.

### 288.5 JSON_KEY_LIST Type

**JSON_KEY_LIST** is a varray of **VARCHAR2(4000)**.

**Description**

This type is used by the **get_Keys** function in the **JSON_OBJECT_T** Object Type.
This chapter describes the logical change record (LCR) types. This chapter contains these topics:

- Overview
- Security Model
- Summary of Logical Change Record Types
- Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD

### 289.1 Logical Change Record TYPES Overview

In Oracle Streams, logical change records (LCRs) are message payloads that contain information about changes to a database. These changes can include changes to the data, which are data manipulation language (DML) changes, and changes to database objects, which are data definition language (DDL) changes.

When you use Oracle Streams, the capture process captures changes in the form of LCRs and enqueues them into a queue. These LCRs can be propagated from a queue in one database to a queue in another database. Finally, the apply process can apply LCRs at a destination database. You also have the option of creating, enqueuing, and dequeuing LCRs manually.

*See Also:*

*Oracle Streams Concepts and Administration* for more information about LCRs

### 289.2 Logical Change Record Types Security Model

PUBLIC is granted **EXECUTE** privilege on the types described in this chapter.

### 289.3 Summary of Logical Change Record Types

This table lists the Logical Change Record TYPES and briefly describes them.

<table>
<thead>
<tr>
<th>Table 289-1 Logical Change Record (LCR) Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>LCR$_DDL_RECORD Type</td>
</tr>
<tr>
<td>LCR$_ROW_RECORD Type</td>
</tr>
</tbody>
</table>
Table 289-1 (Cont.) Logical Change Record (LCR) Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCR$_ROW_LIST Type</td>
<td>Identifies a list of column values for a row in a table</td>
</tr>
<tr>
<td>LCR$_ROW_UNIT Type</td>
<td>Identifies the value for a column in a row</td>
</tr>
</tbody>
</table>

These logical change record (LCR) types can be used with the following Oracle-supplied PL/SQL packages:

- DBMS_APPLY_ADM
- DBMS_AQ
- DBMS_AQADM
- DBMS_CAPTURE_ADM
- DBMS_PROPAGATION_ADM
- DBMS_RULE
- DBMS_RULE_ADM
- DBMS_STREAMS
- DBMS_STREAMS_ADM
- DBMS_TRANSFORM

289.3.1 LCR$_DDL_RECORD Type

This type represents a data definition language (DDL) change to a database object.

If you create or modify a DDL logical change record (DDL LCR), then make sure the `ddl_text` is consistent with the `base_table_name`, `base_table_owner`, `object_type`, `object_owner`, `object_name`, and `command_type` attributes.

This topic contains information about the constructor for row LCRs and information about the member subprograms for this type.

**Note:**

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
- The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.
LCR$_DDL_RECORD Constructor

Creates a SYS.LCR$_DDL_RECORD object with the specified information.

```sql
STATIC FUNCTION CONSTRUCT(
    source_database_name  IN  VARCHAR2,
    command_type          IN  VARCHAR2,
    object_owner          IN  VARCHAR2,
    object_name           IN  VARCHAR2,
    object_type           IN  VARCHAR2,
    ddl_text              IN  CLOB,
    logon_user            IN  VARCHAR2,
    current_schema        IN  VARCHAR2,
    base_table_owner      IN  VARCHAR2,
    base_table_name       IN  VARCHAR2,
    tag                   IN  RAW       DEFAULT NULL,
    transaction_id        IN  VARCHAR2  DEFAULT NULL,
    scn                   IN  NUMBER    DEFAULT NULL,
    position              IN  RAW       DEFAULT NULL,
    edition_name          IN  VARCHAR2  DEFAULT NULL,
    root_name             IN  VARCHAR2  DEFAULT NULL)
RETURN SYS.LCR$_DDL_RECORD;
```

LCR$_DDL_RECORD Constructor Function Parameters

**Table 289-2  Constructor Function Parameters for LCR$_DDL_RECORD**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database_name</td>
<td>The database where the DDL statement occurred</td>
</tr>
<tr>
<td></td>
<td>If the LCRs originated in a multitenant container database (CDB), then</td>
</tr>
<tr>
<td></td>
<td>this field specifies the global name of the container where the DDL change</td>
</tr>
<tr>
<td></td>
<td>occurred.</td>
</tr>
<tr>
<td></td>
<td>If you do not include the domain name, then the function appends the</td>
</tr>
<tr>
<td></td>
<td>local domain to the database name automatically. For example, if you</td>
</tr>
<tr>
<td></td>
<td>specify DBS1 and the local domain is EXAMPLE.COM, then the function</td>
</tr>
<tr>
<td></td>
<td>specifies DBS1.EXAMPLE.COM automatically. Set this parameter to a</td>
</tr>
<tr>
<td></td>
<td>non-NULL value.</td>
</tr>
</tbody>
</table>
Table 289-2 (Cont.) Constructor Function Parameters for LCR$_DDL_RECORD

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_type</td>
<td>The type of command executed in the DDL statement</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to a non-NULL value.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> The &quot;SQL Command Codes&quot; table in the Oracle Call Interface Programmer's Guide for a complete list of command types</td>
</tr>
<tr>
<td></td>
<td>The following command types are not supported in DDL LCRs:</td>
</tr>
<tr>
<td></td>
<td>ALTER MATERIALIZED VIEW</td>
</tr>
<tr>
<td></td>
<td>ALTER MATERIALIZED VIEW LOG</td>
</tr>
<tr>
<td></td>
<td>ALTER SUMMARY</td>
</tr>
<tr>
<td></td>
<td>CREATE SCHEMA</td>
</tr>
<tr>
<td></td>
<td>CREATE MATERIALIZED VIEW</td>
</tr>
<tr>
<td></td>
<td>CREATE MATERIALIZED VIEW LOG</td>
</tr>
<tr>
<td></td>
<td>CREATE SUMMARY</td>
</tr>
<tr>
<td></td>
<td>DROP MATERIALIZED VIEW</td>
</tr>
<tr>
<td></td>
<td>DROP MATERIALIZED VIEW LOG</td>
</tr>
<tr>
<td></td>
<td>DROP SUMMARY</td>
</tr>
<tr>
<td></td>
<td>RENAME</td>
</tr>
<tr>
<td></td>
<td>The snapshot equivalents of the materialized view command types are also not supported.</td>
</tr>
<tr>
<td>object_owner</td>
<td>The user who owns the object on which the DDL statement was executed</td>
</tr>
<tr>
<td>object_name</td>
<td>The database object on which the DDL statement was executed</td>
</tr>
<tr>
<td>object_type</td>
<td>The type of object on which the DDL statement was executed</td>
</tr>
<tr>
<td></td>
<td>The following are valid object types:</td>
</tr>
<tr>
<td></td>
<td>CLUSTER</td>
</tr>
<tr>
<td></td>
<td>FUNCTION</td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td></td>
<td>LINK</td>
</tr>
<tr>
<td></td>
<td>OUTLINE</td>
</tr>
<tr>
<td></td>
<td>PACKAGE</td>
</tr>
<tr>
<td></td>
<td>PACKAGE BODY</td>
</tr>
<tr>
<td></td>
<td>PROCEDURE</td>
</tr>
<tr>
<td></td>
<td>SEQUENCE</td>
</tr>
<tr>
<td></td>
<td>SYNONYM</td>
</tr>
<tr>
<td></td>
<td>TABLE</td>
</tr>
<tr>
<td></td>
<td>TRIGGER</td>
</tr>
<tr>
<td></td>
<td>TYPE</td>
</tr>
<tr>
<td></td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td>VIEW</td>
</tr>
<tr>
<td></td>
<td>LINK represents a database link.</td>
</tr>
<tr>
<td></td>
<td>NULL is also a valid object type. Specify NULL for all object types not listed. The GET_OBJECT_TYPE member procedure returns NULL for object types not listed.</td>
</tr>
<tr>
<td>ddl_text</td>
<td>The text of the DDL statement</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to a non-NULL value.</td>
</tr>
<tr>
<td>logon_user</td>
<td>The user whose session executed the DDL statement</td>
</tr>
</tbody>
</table>
Table 289-2  (Cont.) Constructor Function Parameters for LCR$_\text{DDL\_RECORD}$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_schema</td>
<td>The schema that is used if no schema is specified explicitly for the modified database objects in $\text{ddl_text}$. If a schema is specified in $\text{ddl_text}$ that differs from the one specified for current_schema, then the function uses the schema specified in $\text{ddl_text}$. Set this parameter to a non-NULL value.</td>
</tr>
<tr>
<td>base_table_owner</td>
<td>If the DDL statement is a table-related DDL (such as CREATE TABLE and ALTER TABLE), or if the DDL statement involves a table (such as creating a trigger on a table), then base_table_owner specifies the owner of the table involved. Otherwise, base_table_owner is NULL.</td>
</tr>
<tr>
<td>base_table_name</td>
<td>If the DDL statement is a table-related DDL (such as CREATE TABLE and ALTER TABLE), or if the DDL statement involves a table (such as creating a trigger on a table), then base_table_name specifies the name of the table involved. Otherwise, base_table_name is NULL.</td>
</tr>
<tr>
<td>tag</td>
<td>A binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the DDL statement if apply forwarding is used. See Also: Oracle Streams Replication Administrator’s Guide</td>
</tr>
<tr>
<td>transaction_id</td>
<td>The identifier of the transaction.</td>
</tr>
<tr>
<td>scn</td>
<td>The SCN at the time when the change record for a captured LCR was written to the redo log. The SCN value is meaningless for a user-created LCR.</td>
</tr>
<tr>
<td>position</td>
<td>The position of the LCR. LCR position is commonly used in XStream configurations. Using XStream requires purchasing a license for the Oracle GoldenGate product. See Also: Oracle Database XStream Guide</td>
</tr>
<tr>
<td>edition_name</td>
<td>The name of the edition in which the DDL statement was executed.</td>
</tr>
<tr>
<td>root_name</td>
<td>If the LCRs is associated with a CDB, then this field specifies the global name of the root in the CDB. If the LCR is associated with a non-CDB, then this field is NULL.</td>
</tr>
</tbody>
</table>

Summary of LCR$_\text{DDL\_RECORD}$ Subprograms

Table 289-3  LCR$_\text{DDL\_RECORD}$ Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE Member Procedure</td>
<td>Executes the LCR under the security domain of the current user</td>
</tr>
<tr>
<td>GET_BASE_TABLE_NAME Member Function</td>
<td>Gets the base (dependent) table name</td>
</tr>
<tr>
<td>GET_BASE_TABLE_OWNER Member Function</td>
<td>Gets the base (dependent) table owner</td>
</tr>
<tr>
<td>GET_CURRENT_SCHEMA Member Function</td>
<td>Gets the default schema (user) name</td>
</tr>
</tbody>
</table>
### Table 289-3  (Cont.) LCR$_DDL$ RECORD Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_DDL_TEXT Member Procedure</td>
<td>Gets the DDL text in a <strong>CLOB</strong></td>
</tr>
<tr>
<td>GET_EDITION_NAME Member Function</td>
<td>Gets the name of the edition in which the DDL statement was executed</td>
</tr>
<tr>
<td>GET_LOGON_USER Member Function</td>
<td>Gets the logon user name</td>
</tr>
<tr>
<td>GET_OBJECT_TYPE Member Function</td>
<td>Gets the type of the object involved for the DDL</td>
</tr>
<tr>
<td>SET_BASE_TABLE_NAME Member Procedure</td>
<td>Sets the base (dependent) table name</td>
</tr>
<tr>
<td>SET_BASE_TABLE_OWNER Member Procedure</td>
<td>Sets the base (dependent) table owner</td>
</tr>
<tr>
<td>SET_CURRENT_SCHEMA Member Procedure</td>
<td>Sets the default schema (user) name</td>
</tr>
<tr>
<td>SET_DDL_TEXT Member Procedure</td>
<td>Sets the DDL text</td>
</tr>
<tr>
<td>SET_EDITION_NAME Member Procedure</td>
<td>Sets the name of the edition in which the DDL statement was executed</td>
</tr>
<tr>
<td>SET_LOGON_USER Member Procedure</td>
<td>Sets the logon user name</td>
</tr>
<tr>
<td>SET_OBJECT_TYPE Member Procedure</td>
<td>Sets the object type</td>
</tr>
<tr>
<td>Common Subprograms</td>
<td>See &quot;Common Subprograms for LCR$_DDL$ RECORD and LCR$_ROW$ RECORD&quot; for a list of subprograms common to the <code>SYS.LCR$\_ROW$ RECORD</code> and <code>SYS.LCR$\_DDL$ RECORD</code> types</td>
</tr>
</tbody>
</table>

### EXECUTE Member Procedure

Executes the DDL LCR under the security domain of the current user. Apply handlers are not run when the LCR is applied using this procedure.

**Note:**

The **EXECUTE** member procedure can be invoked only in an apply handler for an Oracle Streams apply process.

**Syntax**

```sql
MEMBER PROCEDURE EXECUTE;
```

**GET_BASE_TABLE_NAME Member Function**

Gets the base (dependent) table name.

**Syntax**

```sql
MEMBER FUNCTION GET_BASE_TABLE_NAME()
RETURN VARCHAR2;
```
GET_BASE_TABLE_OWNER Member Function

Gets the base (dependent) table owner.

Syntax

MEMBER FUNCTION GET_BASE_TABLE_OWNER()
RETURN VARCHAR2;

GET_CURRENT_SCHEMA Member Function

Gets the current schema name.

Syntax

MEMBER FUNCTION GET_CURRENT_SCHEMA()
RETURN VARCHAR2;

GET_DDL_TEXT Member Procedure

Gets the DDL text in a CLOB.

For example, the following PL/SQL code uses this procedure to get the DDL text in a DDL LCR:

CREATE OR REPLACE PROCEDURE ddl_in_lcr (ddl_lcr in SYS.LCR$_DDL_RECORD)
IS
  ddl_text  CLOB;
BEGIN
  DBMS_OUTPUT.PUT_LINE( '  -----------------------------------------' );
  DBMS_OUTPUT.PUT_LINE( '  Displaying DDL text in a DDL LCR: ' );
  DBMS_OUTPUT.PUT_LINE( '  -----------------------------------------' );
  DBMS_LOB.CREATETEMPORARY(ddl_text, true);
  ddl_lcr.GET_DDL_TEXT(ddl_text);
  DBMS_OUTPUT.PUT_LINE('DDL text:' || ddl_text);
  DBMS_LOB.FREETEMPORARY(ddl_text);
END;
/

Note:

GET_DDL_TEXT is a member procedure and not a member function to make it easier for you to manage the space used by the CLOB. Notice that the previous example creates temporary space for the CLOB and then frees the temporary space when it is no longer needed.

Syntax

MEMBER FUNCTION GET_DDL_TEXT(
  ddl_text  IN/OUT  CLOB);
Parameter

Table 289-4 GET_DDL_TEXT Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddl_text</td>
<td>The DDL text in the DDL LCR</td>
</tr>
</tbody>
</table>

GET_EDITION_NAME Member Function

Gets the name of the edition in which the DDL statement was executed.

See Also:

Oracle Database Development Guide

Syntax

MEMBER FUNCTION GET_EDITION_NAME() RETURN VARCHAR2;

GET_LOGON_USER Member Function

Gets the logon user name.

Syntax

MEMBER FUNCTION GET_LOGON_USER() RETURN VARCHAR2;

GET_OBJECT_TYPE Member Function

Gets the type of the object involved for the DDL.

Syntax

MEMBER FUNCTION GET_OBJECT_TYPE() RETURN VARCHAR2;

SET_BASE_TABLE_NAME Member Procedure

Sets the base (dependent) table name.

Syntax

MEMBER PROCEDURE SET_BASE_TABLE_NAME(
    base_table_name IN VARCHAR2);
Parameter

Table 289-5  SET_BASE_TABLE_NAME Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_table_name</td>
<td>The name of the base table</td>
</tr>
</tbody>
</table>

SET_BASE_TABLE_OWNER Member Procedure

Sets the base (dependent) table owner.

Syntax

MEMBER PROCEDURE SET_BASE_TABLE_OWNER(
    base_table_owner  IN  VARCHAR2);

Parameter

Table 289-6  SET_BASE_TABLE_OWNER Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_table_owner</td>
<td>The name of the base table owner</td>
</tr>
</tbody>
</table>

SET_CURRENT_SCHEMA Member Procedure

Sets the default schema (user) name.

Syntax

MEMBER PROCEDURE SET_CURRENT_SCHEMA(
    current_schema  IN  VARCHAR2);

Parameter

Table 289-7  SET_CURRENT_SCHEMA Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_schema</td>
<td>The name of the schema to set as the current schema</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to a non-NULL value.</td>
</tr>
</tbody>
</table>

SET_DDL_TEXT Member Procedure

Sets the DDL text.

Syntax

MEMBER PROCEDURE SET_DDL_TEXT(
    ddl_text  IN  CLOB);
Table 289-8  SET_DDL_TEXT Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddl_text</td>
<td>The DDL text&lt;br&gt;Set this parameter to a non-NULL value.</td>
</tr>
</tbody>
</table>

SET_EDITION_NAME Member Procedure

Sets the name of the edition in which the DDL statement was executed.

See Also:
Oracle Database Development Guide

Syntax

```sql
MEMBER PROCEDURE SET_EDITION_NAME(
    edition_name  IN  VARCHAR2);
```

Table 289-9  SET_EDITION_NAME Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>edition_name</td>
<td>Name of the edition</td>
</tr>
</tbody>
</table>

SET_LOGON_USER Member Procedure

Sets the logon user name.

Syntax

```sql
MEMBER PROCEDURE SET_LOGON_USER(
    logon_user  IN  VARCHAR2);
```

Table 289-10  SET_LOGON_USER Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logon_user</td>
<td>The name of the schema to set as the logon user</td>
</tr>
</tbody>
</table>

SET_OBJECT_TYPE Member Procedure

Sets the object type.
Syntax

MEMBER PROCEDURE SET_OBJECT_TYPE(
    object_type  IN  VARCHAR2);

Parameter

**Table 289-11  SET_OBJECT_TYPE Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>The object type</td>
</tr>
<tr>
<td></td>
<td>The following are valid object types:</td>
</tr>
<tr>
<td></td>
<td>CLUSTER</td>
</tr>
<tr>
<td></td>
<td>FUNCTION</td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td></td>
<td>LINK</td>
</tr>
<tr>
<td></td>
<td>OUTLINE</td>
</tr>
<tr>
<td></td>
<td>PACKAGE</td>
</tr>
<tr>
<td></td>
<td>PACKAGE BODY</td>
</tr>
<tr>
<td></td>
<td>PROCEDURE</td>
</tr>
<tr>
<td></td>
<td>SEQUENCE</td>
</tr>
<tr>
<td></td>
<td>SYNONYM</td>
</tr>
<tr>
<td></td>
<td>TABLE</td>
</tr>
<tr>
<td></td>
<td>TRIGGER</td>
</tr>
<tr>
<td></td>
<td>TYPE</td>
</tr>
<tr>
<td></td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td>VIEW</td>
</tr>
</tbody>
</table>

LINK represents a database link.

NULL is also a valid object type. Specify NULL for all object types not listed. The GET_OBJECT_TYPE member procedure returns NULL for object types not listed.

### 289.3.2 LCR$-_ROW_RECORD Type

This type represents a data manipulation language (DML) change to a row in a table. This type uses the LCR$-_ROW_LIST type.

If you create or modify a row logical change record (row LCR), then make sure the command_type attribute is consistent with the presence or absence of old column values and the presence or absence of new column values.

This topic contains information about the constructor for DDL LCRs and information about the member subprograms for this type.
Note:

• When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.

• The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

See Also:

LCR$ ROW LIST Type

LCR$ ROW RECORD Constructor

Creates a SYS.LCR$ ROW_RECORD object with the specified information.

STATIC FUNCTION CONSTRUCT(
  source_database_name  IN  VARCHAR2,
  command_type          IN  VARCHAR2,
  object_owner          IN  VARCHAR2,
  object_name           IN  VARCHAR2,
  tag                   IN  RAW                DEFAULT NULL,
  transaction_id        IN  VARCHAR2           DEFAULT NULL,
  scn                   IN  NUMBER             DEFAULT NULL,
  old_values            IN  SYS.LCR$ ROW LIST  DEFAULT NULL,
  new_values            IN  SYS.LCR$ ROW LIST  DEFAULT NULL,
  position              IN  RAW                DEFAULT NULL,
  statement             IN  VARCHAR2           DEFAULT NULL,
  bind_variables        IN  SYS.LCR$ ROW LIST  DEFAULT NULL,
  bind_by_position      IN  VARCHAR2           DEFAULT 'N',
  root_name             IN  VARCHAR2           DEFAULT NULL)
RETURN SYS.LCR$ ROW_RECORD;
LCR$\_ROW\_RECORD Constructor Function Parameters

Table 289-12  Constructor Function Parameters for LCR$\_ROW\_RECORD

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database_name</td>
<td>The database where the row change occurred                                                                                                                                   If the LCRs originated in a CDB, then this field specifies the global name of the container where the row change occurred. If you do not include the domain name, then the function appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is EXAMPLE.COM, then the function specifies DBS1.EXAMPLE.COM automatically. Set this parameter to a non-NULL value.</td>
</tr>
<tr>
<td>command_type</td>
<td>The type of command executed in the DML statement                                                                                                                             Set this parameter to a non-NULL value. Valid values are the following: INSERT, UPDATE, DELETE, LOB ERASE, LOB WRITE, LOB TRIM. If INSERT, then ensure that the LCR has a new_values collection that is not empty and an empty or NULL old_values collection. If UPDATE, then ensure that the LCR has a new_values collection that is not empty and an old_values collection that is not empty. If DELETE, then ensure that the LCR has a NULL or empty new_values collection and an old_values collection that is not empty. If LOB ERASE, LOB WRITE, or LOB TRIM, then ensure that the LCR has a new_values collection that is not empty and an empty or NULL old_values collection.</td>
</tr>
<tr>
<td>object_owner</td>
<td>The user who owns the table on which the row change occurred                                                                                                               Set this parameter to a non-NULL value.</td>
</tr>
<tr>
<td>object_name</td>
<td>The table on which the DML statement was executed                                                                                                                            Set this parameter to a non-NULL value.</td>
</tr>
<tr>
<td>tag</td>
<td>A binary tag that enables tracking of the LCR                                                                                                                                For example, this tag can be used to determine the original source database of the DML change when apply forwarding is used. See Also: Oracle Streams Replication Administrator’s Guide</td>
</tr>
<tr>
<td>transaction_id</td>
<td>The identifier of the transaction</td>
</tr>
<tr>
<td>scn</td>
<td>The SCN at the time when the change record was written to the redo log                                                                                                         The SCN value is meaningless for a user-created LCR.</td>
</tr>
</tbody>
</table>
Table 289-12  (Cont.) Constructor Function Parameters for LCR$_ROW_RECORD

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| old_values      | The column values for the row before the DML change  
If the DML statement is an **UPDATE** or a **DELETE** statement, then this parameter contains the values of columns in the row before the DML statement. If the DML statement is an **INSERT** statement, then there are no old values. |
| new_values      | The column values for the row after the DML change  
If the DML statement is an **UPDATE** or an **INSERT** statement, then this parameter contains the values of columns in the row after the DML statement. If the DML statement is a **DELETE** statement, then there are no new values.  
If the LCR reflects a LOB operation, then this parameter contains the supplementally logged columns and any relevant LOB information. |
| position        | The position of the LCR  
LCR position is commonly used in XStream configurations. Using XStream requires purchasing a license for the Oracle GoldenGate product. |
| statement       | This parameter is reserved for internal use only. |
| bind_variables  | This parameter is reserved for internal use only. |
| bind_by_position| This parameter is reserved for internal use only. |
| root_name       | If the LCRs is associated with a CDB, then this field specifies the global name of the root in the CDB.  
If the LCR is associated with a non-CDB, then this field is **NULL**. |

Summary of LCR$_ROW_RECORD Subprograms

Table 289-13  LCR$_ROW_RECORD Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADD_COLUMN Member Procedure</strong></td>
<td>Adds the value as old or new, depending on the value type specified, for the column</td>
</tr>
<tr>
<td><strong>CONVERT_LONG_TO_LOB_CHUNK Member Procedure</strong></td>
<td>Converts <strong>LONG</strong> data in a row LCR into fixed width <strong>CLOB</strong>, or converts <strong>LONG RAW</strong> data in a row LCR into a <strong>BLOB</strong></td>
</tr>
<tr>
<td><strong>DELETE_COLUMN Member Procedure</strong></td>
<td>Deletes the old value, the new value, or both, for the specified column, depending on the value type specified</td>
</tr>
<tr>
<td><strong>EXECUTE Member Procedure</strong></td>
<td>Executes the LCR under the security domain of the current user</td>
</tr>
<tr>
<td><strong>GET_LOB_INFORMATION Member Function</strong></td>
<td>Gets the LOB information for the column</td>
</tr>
<tr>
<td><strong>GET_LOB_OFFSET Member Function</strong></td>
<td>Gets the LOB offset for the specified column</td>
</tr>
<tr>
<td><strong>GET_LOB_OPERATION_SIZE Member Function</strong></td>
<td>Gets the operation size for the <strong>LOB</strong> column</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GET_LONG_INFORMATION Member Function</td>
<td>Gets the LONG information for the column</td>
</tr>
<tr>
<td>GET_ROW_TEXT Member Procedure</td>
<td>Gets the SQL statement for the change that is encapsulated in the LCR</td>
</tr>
<tr>
<td>GET_VALUE Member Function</td>
<td>Gets the old or new value for the specified column, depending on the value type specified</td>
</tr>
<tr>
<td>GET_VALUES Member Function</td>
<td>Gets a list of old or new values, depending on the value type specified</td>
</tr>
<tr>
<td>GET_WHERE_CLAUSE Member Procedure</td>
<td>Gets a WHERE clause for the change that is encapsulated in the row LCR</td>
</tr>
<tr>
<td>GET_XML_INFORMATION Member Function</td>
<td>Gets the XML information for the specified column</td>
</tr>
<tr>
<td>IS_STATEMENT_LCR Member Function</td>
<td>Reserved for internal use only</td>
</tr>
<tr>
<td>KEEP_COLUMNS Member Procedure</td>
<td>Keeps a list of columns a row LCR</td>
</tr>
<tr>
<td>RENAME_COLUMN Member Procedure</td>
<td>Renames a column in an LCR</td>
</tr>
<tr>
<td>SET_LOB_INFORMATION Member Procedure</td>
<td>Sets LOB information for the column</td>
</tr>
<tr>
<td>SET_LOB_OFFSET Member Procedure</td>
<td>Sets the LOB offset for the specified column</td>
</tr>
<tr>
<td>SET_LOB_OPERATION_SIZE Member Procedure</td>
<td>Sets the operation size for the LOB column</td>
</tr>
<tr>
<td>SET_ROW_TEXT Member Procedure</td>
<td>Reserved for internal use only</td>
</tr>
<tr>
<td>SET_VALUE Member Procedure</td>
<td>Overwrites the value of the specified column</td>
</tr>
<tr>
<td>SET_VALUES Member Procedure</td>
<td>Replaces the existing old or new values for the LCR, depending on the value type specified</td>
</tr>
<tr>
<td>SET_XML_INFORMATION Member Procedure</td>
<td>Sets the XML information for the column</td>
</tr>
</tbody>
</table>

**Common Subprograms**

See Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD for a list of subprograms common to the SYS.LCR$_ROW_RECORD and SYS.LCR$_DDL_RECORD types.

---

**ADD_COLUMN Member Procedure**

Adds the value as old or new, depending on the value type specified, for the column. An error is raised if a value of the same type already exists for the column.

---

**Note:**

To set a column value that already exists, run **SET_VALUE**.
Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler and the handler is using LOB assembly (the assemble_lob parameter is set to TRUE for the handler), you use this member procedure in the handler procedure to add a LOB column to a row LCR. If assemble_lob is set to FALSE for the handler, then you cannot use this member procedure to add a LOB column to a row LCR.

To use a DML or error handler to add a LOB column, specify the LOB locator for the column_value parameter in the member procedure. The ADD_COLUMN member procedure verifies that an ANYDATA encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an ANYDATA encapsulated LOB locator.
- An attempt is made to add an LOB column that is set incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure to add a LOB column.
- When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure to add a LONG or LONG RAW column.

Syntax

MEMBER PROCEDURE ADD_COLUMN(
    value_type    IN  VARCHAR2,
    column_name   IN  VARCHAR2,
    column_value  IN  ANYDATA);

Parameters

Table 289-14  ADD_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to add for the column</td>
</tr>
<tr>
<td></td>
<td>Specify old to add the old value of the column. Specify new to add the</td>
</tr>
<tr>
<td></td>
<td>new value of the column.</td>
</tr>
<tr>
<td>column_name</td>
<td>The column name</td>
</tr>
<tr>
<td></td>
<td>This name is not validated. An error can be raised during application of</td>
</tr>
<tr>
<td></td>
<td>the LCRs if an invalid name is specified.</td>
</tr>
<tr>
<td>column_value</td>
<td>The value of the column</td>
</tr>
<tr>
<td></td>
<td>If NULL, then this procedure raises an error.</td>
</tr>
<tr>
<td></td>
<td>If the member procedure is used in a procedure DML handler or error</td>
</tr>
<tr>
<td></td>
<td>handler that uses LOB assembly, then a LOB locator can be specified.</td>
</tr>
<tr>
<td></td>
<td>A NULL column value can be specified by encapsulating the NULL</td>
</tr>
<tr>
<td></td>
<td>value in an ANYDATA wrapper.</td>
</tr>
</tbody>
</table>

CONVERT_LONG_TO_LOB_CHUNK Member Procedure

Converts LONG data in a row LCR into a CLOB, or converts LONG RAW data in a row LCR into a BLOB. This procedure can change the operation code from LONG WRITE to LOB WRITE for the row LCR.

This member procedure can be used in rule-based transformations.

The following restrictions apply to this member procedure:

- This member procedure cannot be used in apply handlers.
- LONG data can be sent as a part of a row LCR with one of the following operation codes: INSERT, UPDATE, or LONG WRITE. Because LONG data can be sent in multiple pieces, make sure that this method is invoked on either none or all LONG pieces.
- LOB to LONG conversion is not supported.
- A row LCR on which this procedure is executed must have been created by a capture process. That is, this procedure does not support persistent row LCRs.

See Also:

Oracle Streams Replication Administrator's Guide

Syntax

MEMBER PROCEDURE CONVERT_LONG_TO_LOB_CHUNK;
DELETE_COLUMN Member Procedure

Deletes the old value, the new value, or both, for the specified column, depending on the value type specified.

Syntax

MEMBER PROCEDURE DELETE_COLUMN(
    column_name  IN  VARCHAR2,
    value_type   IN  VARCHAR2  DEFAULT '*');

Parameters

Table 289-15  DELETE_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| column_name | The column name  
  An error is raised if the column does not exist in the LCR. |
| value_type | The type of value to delete for the column  
  Specify old to delete the old value of the column. Specify new to delete the new value of the column. If * is specified, then the procedure deletes both the old and new values. |

EXECUTE Member Procedure

Executes the row LCR under the security domain of the current user. Any apply handlers that would be run for an LCR are not run when the LCR is applied using this procedure.

This member procedure can be run on a row LCR under any of the following conditions:

- The LCR is being processed by an apply handler.
- The LCR is in a queue and was last enqueued by a mechanism other than an Oracle Streams capture process, such as an Oracle Streams apply process or an application.
- The LCR has been constructed using the LCR$_ROW_RECORD constructor function but has not been enqueued.
- The LCR is in the error queue.

Note:

Do not run this member procedure in a custom rule-based transformation on a row LCR. Doing so could execute the row LCR outside of its transactional context.

Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler, and the handler is using LOB assembly (the assemble_lob parameter is set
to TRUE for the handler), this member procedure executes the assembled row LCR. An assembled row LCR represents a LOB value with a LOB locator or NULL.

If assemble_lobs is set to FALSE for the handler, then this member procedure executes the nonassembled row LCRs. Nonassembled row LCRs represent LOB values with VARCHAR2 and RAW data types. These nonassembled row LCRs might have been modified by the handler.

An error is raised under the following conditions:

- A DML or error handler configured with assemble_lobs set to FALSE attempts to execute a row LCR that contains a LOB locator.
- A DML or error handler configured with assemble_lobs set to TRUE attempts to execute a row LCR that contains one or more LOB values represented with VARCHAR2 or RAW data types.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Syntax

MEMBER PROCEDURE EXECUTE(
    conflict_resolution  IN  BOOLEAN);

Parameters

Table 289-16 EXECUTE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict_resolution</td>
<td>If TRUE, then any conflict resolution defined for the table using the SET_UPDATE_CONFLICT_HANDLER procedure in the DBMS_APPLY_ADM package is used to resolve conflicts resulting from the execution of the LCR. If FALSE, then conflict resolution is not used. An error is raised if this parameter is not specified or is set to NULL.</td>
</tr>
</tbody>
</table>

GET_LOB_INFORMATION Member Function

Gets the LOB information for the column.

The return value can be one of the following:

- DBMS_LCR.NOT_A_LOB CONSTANT NUMBER := 1;
- DBMS_LCR.NULL_LOB CONSTANT NUMBER := 2;
- DBMS_LCR.INLINE_LOB CONSTANT NUMBER := 3;
- DBMS_LCR.EMPTY_LOB CONSTANT NUMBER := 4;
- DBMS_LCR.LOB_CHUNK CONSTANT NUMBER := 5;
- DBMS_LCR.LAST_LOB_CHUNK CONSTANT NUMBER := 6;

Returns NULL if the specified column does not exist.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the value of the columns.
Syntax

MEMBER FUNCTION GET_LOB_INFORMATION(
    value_type   IN  VARCHAR2,
    column_name  IN  VARCHAR2,
    use_old      IN  VARCHAR2  DEFAULT 'Y')
RETURN NUMBER;

Parameters

Table 289-17  GET_LOB_INFORMATION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to return for the column, either old or new</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column</td>
</tr>
<tr>
<td>use_old</td>
<td>If Y and value_type is new, and no new value exists, then the function returns the corresponding old value. If N and value_type is new, then the function does not return the old value if no new value exists. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.</td>
</tr>
</tbody>
</table>

GET_LOB_OFFSET Member Function

Gets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB. That is, the information is DBMS_LCR.LAST_LOB_CHUNK or DBMS_LCR.LOB_CHUNK
- The command type is LOB_ERASE or LOB_WRITE

Otherwise, returns NULL.

Syntax

GET_LOB_OFFSET(
    value_type   IN  VARCHAR2,
    column_name  IN  VARCHAR2)
RETURN NUMBER;

Parameters

Table 289-18  GET_LOB_OFFSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to return for the column Currently, only new can be specified.</td>
</tr>
</tbody>
</table>
Table 289-18 (Cont.) GET_LOB_OFFSET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_name</td>
<td>The name of the LOB column</td>
</tr>
</tbody>
</table>

GET_LOB_OPERATION_SIZE Member Function

Gets the operation size for the LOB column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB
- The command type is LOB ERASE or LOB TRIM
- The information is DBMS_LCR.LAST_LOB_CHUNK

Otherwise, returns NULL.

Syntax

```sql
MEMBER FUNCTION GET_LOB_OPERATION_SIZE(
    value_type   IN  VARCHAR2,
    column_name  IN  VARCHAR2)
RETURN NUMBER,
```

Parameters

Table 289-19 GET_LOB_OPERATION_SIZE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to return for the column</td>
</tr>
<tr>
<td></td>
<td>Currently, only new can be specified.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the LOB column</td>
</tr>
</tbody>
</table>

GET_LONG_INFORMATION Member Function

Gets the LONG information for the column.

The return value can be one of the following:

- DBMS_LCR.NOT_A_LONG CONSTANT NUMBER := 1;
- DBMS_LCR.NULL_LONG CONSTANT NUMBER := 2;
- DBMS_LCR.INLINE_LONG CONSTANT NUMBER := 3;
- DBMS_LCR.LONG_CHUNK CONSTANT NUMBER := 4;
- DBMS_LCR.LAST_LONG_CHUNK CONSTANT NUMBER := 5;

Returns NULL if the specified column does not exist.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the value of the columns.
Syntax

MEMBER FUNCTION GET_LONG_INFORMATION(
    value_type   IN  VARCHAR2,
    column_name  IN  VARCHAR2,
    use_old      IN  VARCHAR2  DEFAULT 'Y')
RETURN NUMBER;

Parameters

Table 289-20  GET_LONG_INFORMATION Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to return for the column, either old or new</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column</td>
</tr>
<tr>
<td>use_old</td>
<td>If Y and value_type is new, and no new value exists, then the function returns the corresponding old value. If N and value_type is new, then the function does not return the old value if no new value exists. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.</td>
</tr>
</tbody>
</table>

GET_ROW_TEXT Member Procedure

Gets the SQL statement for the change that is encapsulated in the row LCR. This method performs SQL generation in PL/SQL.

This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

The following procedure returns the SQL statement in a CLOB datatype.

MEMBER PROCEDURE GET_ROW_TEXT(
    row_text  IN/OUT  CLOB); 

The following procedure returns the SQL statement with bind variables in a CLOB datatype.

MEMBER PROCEDURE GET_ROW_TEXT(
    row_text         IN/OUT  CLOB,
    variable_list    IN/OUT  LCR$_ROW_LIST,
    bind_var_syntax  IN      VARCHAR2  DEFAULT ':');
### Parameters

**Table 289-21 GET_ROW_TEXT Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_text</td>
<td>The SQL statement for the change that is encapsulated in the LCR</td>
</tr>
<tr>
<td>variable_list</td>
<td>The values for the bind variables in the order of the bind variables</td>
</tr>
<tr>
<td>bind_var_syntax</td>
<td>The syntax for the bind variables</td>
</tr>
<tr>
<td></td>
<td>One of the following values is valid:</td>
</tr>
<tr>
<td></td>
<td>• Specify :, the default, for bind values to be in the form :1, :2, and so on.</td>
</tr>
<tr>
<td></td>
<td>• Specify ? for bind values to be in the form ?.</td>
</tr>
</tbody>
</table>

#### GET_VALUE Member Function

Gets the old or new value for the specified column, depending on the value type specified.

If the command type of the row LCR is `UPDATE`, then specifying 'Y' for the `use_old` parameter is a convenient way to get the value of a column.

**Syntax**

```sql
MEMBER FUNCTION GET_VALUE(
    value_type   IN  VARCHAR2,
    column_name  IN  VARCHAR2,
    use_old      IN  VARCHAR2  DEFAULT 'Y')
RETURN ANYDATA;
```

### Parameters

**Table 289-22 GET_VALUE Function Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to return for the column</td>
</tr>
<tr>
<td></td>
<td>Specify old to get the old value for the column. Specify new to get the new value for the column.</td>
</tr>
<tr>
<td>column_name</td>
<td>The column name</td>
</tr>
<tr>
<td></td>
<td>If the column is present and has a NULL value, then the function returns an ANYDATA instance containing a NULL value. If the column value is absent, then the function returns a NULL.</td>
</tr>
<tr>
<td>use_old</td>
<td>If Y and value_type is new, and no new value exists, then the function returns the corresponding old value.</td>
</tr>
<tr>
<td></td>
<td>If N and value_type is new, then the function returns NULL if no new value exists.</td>
</tr>
<tr>
<td></td>
<td>If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter.</td>
</tr>
<tr>
<td></td>
<td>NULL is not a valid specification for the use_old parameter.</td>
</tr>
</tbody>
</table>
GET_VALUES Member Function

Gets a list of old or new values, depending on the value type specified.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the values of all columns.

Syntax

MEMBER FUNCTION GET_VALUES(
    value_type  IN  VARCHAR2,
    use_old     IN  VARCHAR2  DEFAULT 'Y')
RETURN SYS.LCR$_ROW_LIST;

Parameters

Table 289-23  GET_VALUES Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of values to return. Specify old to return a list of old values. Specify new to return a list of new values.</td>
</tr>
<tr>
<td>use_old</td>
<td>If Y and value_type is new, then the function returns a list of all new values in the LCR. If a new value does not exist in the list, then the function returns the corresponding old value. Therefore, the returned list contains all existing new values and the old values where there are no new values. If N and value_type is new, then the function returns a list of all new values in the LCR without returning any old values. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.</td>
</tr>
</tbody>
</table>

GET_WHERE_CLAUSE Member Procedure

Gets a WHERE clause for the change that is encapsulated in the row LCR.

Use the WHERE clause returned by GET_WHERE_CLAUSE instead of using the ROWID, because the ROWID is not ANSI compatible. The generated WHERE clause might not match the WHERE clause in the original DML operation.

The ROWID of an INSERT statement is the ROWID of the new row created by the INSERT. The WHERE clause generated for an INSERT operation identifies the new row. Therefore, the generated WHERE clause includes all of the new values inserted.

For example, consider the following insert into the hr.departments table:

```
INSERT INTO hr.departments (  
    department_id, department_name, manager_id, location_id)  
VALUES (10, 'HR', 20, 40);
```

The generated WHERE clause represents the row with the values 10, 'HR', 20, and 40. Hence, the generated WHERE clause is the following:
WHERE "DEPARTMENT_ID" = 10 AND "DEPARTMENT_NAME" = 'HR' AND
"MANAGER_ID" = 20 AND "LOCATION_ID" = 40

The ROWID of an UPDATE statement is the ROWID of the row that was updated. The WHERE clause generated for an UPDATE operation identifies the row after the UPDATE executes. The generated WHERE clause is based on the old and new values of the UPDATE.

For example, consider the following update to the hr.departments table:

UPDATE hr.departments SET department_name='Management'
WHERE department_name='Administration' AND location_id = 20 AND
    manager_id = 30 AND department_id = 10;

The values of the row after the UPDATE are 10, 'Management', 30, and 20. Hence, the generated WHERE clause to identify the row is the following:

WHERE "DEPARTMENT_ID" = 10 AND "DEPARTMENT_NAME" = 'MANAGEMENT' AND
"MANAGER_ID" = 30 AND "LOCATION_ID" = 20

Notice that the new value is used for "DEPARTMENT_NAME", because the new value is the value of the column after the UPDATE. For the rest of the columns, the old values are used.

The ROWID of a DELETE operation is the row that existed before it was deleted. The generated WHERE clause consists of all the old column values present in the DELETE operation.

LOB columns do not appear in generated WHERE clauses. The generated WHERE clause is not affected by the presence of LOB columns in the LCR.

This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

The following procedure returns the WHERE clause of a SQL statement in a CLOB datatype.

MEMBER PROCEDURE GET_WHERE_CLAUSE(
    where_clause  IN/OUT  CLOB);

The following procedure returns the WHERE clause of a SQL statement with bind variables in a CLOB datatype.

MEMBER PROCEDURE GET_WHERE_CLAUSE(
    where_clause  IN/OUT  CLOB,
    variable_list IN/OUT  LCR$_ROW_LIST,
    bind_var_syntax IN      VARCHAR2  DEFAULT ':');

See Also:

- LCR$_ROW_LIST Type
- Oracle Streams Concepts and Administration
Parameters

Table 289-24  GET_WHERE_CLAUSE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>where_clause</td>
<td>The <strong>WHERE</strong> clause of the SQL statement for the change that is encapsulated in the LCR</td>
</tr>
<tr>
<td>variable_list</td>
<td>The values for the bind variables in the order of the bind variables</td>
</tr>
<tr>
<td>bind_var_syntax</td>
<td>The syntax for the bind variables</td>
</tr>
<tr>
<td></td>
<td>One of the following values is valid:</td>
</tr>
<tr>
<td></td>
<td>• Specify ( :, ) the default, for bind values to be in the form ( :1, :2 ) and so on.</td>
</tr>
<tr>
<td></td>
<td>• Specify ( ? ) for bind values to be in the form ( ? ).</td>
</tr>
</tbody>
</table>

GET_XML_INFORMATION Member Function

Gets the XML information for the specified column.

The return value can be one of the following:

- `DBMS_LCR.NOT_XML` CONSTANT NUMBER := 1;
- `DBMS_LCR.XML_DOC` CONSTANT NUMBER := 2;
- `DBMS_LCR.XML_DIFF` CONSTANT NUMBER := 3;

`DBMS_LCR.NOT_XML` indicates that the column is not an `XMLType` column.

`DBMS_LCR.XML_DOC` indicates that the column contains an XML document.

`DBMS_LCR.XML_DIFF` indicates that the column contains an XML document that describes a change made by an update operation. This XML document describes the differences in the column's XML document. The entire XML document is not replaced.

`DBMS_LCR.XML_DIFF` indicates that the column contains differences between old and new XML documents for an update operation.

Returns `NULL` if the specified column does not exist.

Syntax

```
MEMBER FUNCTION GET_XML_INFORMATION(
    column_name  IN  VARCHAR2)
RETURN NUMBER;
```

Parameter

Table 289-25  GET_XML_INFORMATION Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_name</td>
<td>The column name</td>
</tr>
</tbody>
</table>
IS_STATEMENT_LCR Member Function

This function is reserved for internal use only.

KEEP_COLUMNS Member Procedure

This procedure keeps a list of columns in a row LCR. The procedure deletes columns that are not in the list from the row LCR.

Syntax

MEMBER_PROCEDURE KEEP_COLUMNS(
    column_list  IN  VARCHAR2,
    value_type   IN  VARCHAR2  DEFAULT '*');

Parameters

Table 289-26  KEEP_COLUMNS Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_list</td>
<td>The names of the columns kept for the row LCR. Specify a comma-delimited list of type VARCHAR2. This procedure removes columns that are not in the list from the current row LCR.</td>
</tr>
<tr>
<td>value_type</td>
<td>The type of value for which to keep the columns. Specify old to keep the old values of the columns. An error is raised if the old values do not exist in the LCR. Specify new to keep the new values of the columns. An error is raised if the new values do not exist in the LCR. If * is specified, then the procedure keeps both the old and the new columns.</td>
</tr>
</tbody>
</table>

RENAME_COLUMN Member Procedure

Renames a column in a row LCR.

Syntax

MEMBER_PROCEDURE RENAME_COLUMN(
    from_column_name  IN  VARCHAR2,
    to_column_name    IN  VARCHAR2,
    value_type        IN  VARCHAR2  DEFAULT '*');

Parameters

Table 289-27  RENAME_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from_column_name</td>
<td>The existing column name</td>
</tr>
<tr>
<td>to_column_name</td>
<td>The new column name</td>
</tr>
<tr>
<td></td>
<td>An error is raised if a column with the specified name already exists.</td>
</tr>
</tbody>
</table>
Table 289-27  (Cont.) RENAME_COLUMN Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value for which to rename the column</td>
</tr>
<tr>
<td></td>
<td>Specify <code>old</code> to rename the old value of the column. An error is raised if</td>
</tr>
<tr>
<td></td>
<td>the old value does not exist in the LCR.</td>
</tr>
<tr>
<td></td>
<td>Specify <code>new</code> to rename the new value of the column. An error is raised if</td>
</tr>
<tr>
<td></td>
<td>the new value does not exist in the LCR.</td>
</tr>
<tr>
<td></td>
<td>If <code>*</code> is specified, then the procedure renames the column names for both</td>
</tr>
<tr>
<td></td>
<td>old and new value. The procedure raises an error if either column value does</td>
</tr>
<tr>
<td></td>
<td>not exist in the LCR.</td>
</tr>
</tbody>
</table>

SET_LOB_INFORMATION Member Procedure

Sets LOB information for the column.

**Note:**

When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

Syntax

```sql
MEMBER PROCEDURE SET_LOB_INFORMATION(
    value_type           IN  VARCHAR2,
    column_name          IN  VARCHAR2,
    lob_information      IN  NUMBER);
```

Parameters

Table 289-28  SET_LOB_INFORMATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to set for the column, either <code>old</code> or <code>new</code></td>
</tr>
<tr>
<td></td>
<td>Specify <code>old</code> only if <code>lob_information</code> is set to <code>DBMS_LCR.NOT_A_LOB</code>.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column.</td>
</tr>
<tr>
<td></td>
<td>An exception is raised if the column value does not exist. You might need</td>
</tr>
<tr>
<td></td>
<td>to set this parameter for non-LOB columns.</td>
</tr>
<tr>
<td>lob_information</td>
<td>Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_LCR.NOT_A_LOB</code> CONSTANT NUMBER := 1;</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_LCR.NULL_LOB</code> CONSTANT NUMBER := 2;</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_LCR.INLINE_LOB</code> CONSTANT NUMBER := 3;</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_LCR.EMPTY_LOB</code> CONSTANT NUMBER := 4;</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_LCR.LOB_CHUNK</code> CONSTANT NUMBER := 5;</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_LCR.LAST_LOB_CHUNK</code> CONSTANT NUMBER := 6;</td>
</tr>
</tbody>
</table>
SET_LOB_OFFSET Member Procedure

Sets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns.

**Note:**

When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

**Syntax**

```
MEMBER_PROCEDURE_SET_LOB_OFFSET(
    value_type   IN  VARCHAR2,
    column_name  IN  VARCHAR2,
    lob_offset   IN  NUMBER);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to set for the column</td>
</tr>
<tr>
<td></td>
<td>Currently, only <strong>new</strong> can be specified.</td>
</tr>
<tr>
<td>column_name</td>
<td>The column name</td>
</tr>
<tr>
<td></td>
<td>An error is raised if the column value does not exist in the LCR.</td>
</tr>
<tr>
<td>lob_offset</td>
<td>The LOB offset number</td>
</tr>
<tr>
<td></td>
<td>Valid values are <strong>NULL</strong> or a positive integer less than or equal to</td>
</tr>
<tr>
<td></td>
<td><strong>DBMS_LOB.LOBMAXSIZE</strong>.</td>
</tr>
</tbody>
</table>

SET_LOB_OPERATION_SIZE Member Procedure

Sets the operation size for the LOB column in the number of characters for CLOB columns and bytes for BLOB columns.

**Note:**

When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

**Syntax**

```
MEMBER_PROCEDURE_SET_LOB_OPERATION_SIZE(
    value_type          IN  VARCHAR2,
    column_name         IN  VARCHAR2,
    lob_operation_size  IN  NUMBER);
```

Chapter 289

Summary of Logical Change Record Types
Parameters

Table 289-30  SET_LOB_OPERATION_SIZE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to set for the column</td>
</tr>
<tr>
<td></td>
<td>Currently, only new can be specified.</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the LOB column</td>
</tr>
<tr>
<td></td>
<td>An exception is raised if the column value does not exist in the LCR.</td>
</tr>
<tr>
<td>lob_operation_size</td>
<td>If lob_information for the LOB is or will be DBMS_LCR.LAST_LOB_CHUNK, then this parameter can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE. Otherwise, set to NULL.</td>
</tr>
</tbody>
</table>

SET_ROW_TEXT Member Procedure

This procedure is reserved for internal use only.

SET_VALUE Member Procedure

Overwrites the old or new value of the specified column.

One reason to overwrite an old value for a column is to resolve an error that resulted from a conflict.

Note:

To add a column to a row LCR, run ADD_COLUMN.

See Also:

ADD_COLUMN Member Procedure

Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler, and the handler is using LOB assembly (the assemble_lob parameter is set to TRUE for the handler), you can use this member procedure in the handler procedure on a LOB column in a row LCR. If assemble_lob is set to FALSE for the handler, then you cannot use this member procedure on a LOB column.

To use a DML or error handler to set the value of a LOB column, specify the LOB locator for the column_value parameter in the member procedure. The SET_VALUE
member procedure verifies that an ANYDATA encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an ANYDATA encapsulated LOB locator.
- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

**Note:**

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on a LOB column.
- When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure on a LONG or LONG RAW column.

**Considerations for XMLType Columns**

When processing a row LCR with XMLType columns with a procedure DML handler or error handler, any XMLType columns and LOB columns in the LCR are always assembled using LOB assembly. You can use this member procedure in the handler procedure on a row LCR that contains one or more XMLType columns.

To use a DML or error handler to set the value an XMLType column, specify the XMLType for the column_value parameter. The SET_VALUE member procedure verifies that an ANYDATA encapsulated XMLType is processed with a DML or error handler. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an ANYDATA encapsulated XMLType.
- An attempt is made to set a XMLType column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the XMLType column is represented by the original (nonassembled) row LCRs.

**Note:**

- Database compatibility must be 11.1.0 or higher to process row LCRs with XMLType columns.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on XMLType columns.
Syntax

MEMBER PROCEDURE SET_VALUE(
    value_type IN VARCHAR2,
    column_name IN VARCHAR2,
    column_value IN ANYDATA);

Parameters

Table 289-31  SET_VALUE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of value to set</td>
</tr>
<tr>
<td></td>
<td>Specify old to set the old value of the column. Specify new to set the</td>
</tr>
<tr>
<td></td>
<td>new value of the column.</td>
</tr>
<tr>
<td>column_name</td>
<td>The column name</td>
</tr>
<tr>
<td></td>
<td>An error is raised if the specified column_value does not exist in the</td>
</tr>
<tr>
<td></td>
<td>LCR for the specified column_type.</td>
</tr>
<tr>
<td>column_value</td>
<td>The new value of the column</td>
</tr>
<tr>
<td></td>
<td>If NULL is specified, then this procedure raises an error. To set the</td>
</tr>
<tr>
<td></td>
<td>value to NULL, encapsulate the NULL in an ANYDATA instance.</td>
</tr>
<tr>
<td></td>
<td>If the member procedure is used in a procedure DML handler or error</td>
</tr>
<tr>
<td></td>
<td>handler that uses LOB assembly, then specify a LOB locator for LOB columns.</td>
</tr>
</tbody>
</table>

SET_VALUES Member Procedure

Replaces all old values or all new values for the LCR, depending on the value type specified.

Considerations for LOB Columns

You can use this procedure when processing a row LCR with LOB columns with a procedure DML handler or error handler. If the handler is using LOB assembly (the assemble_lob parameter is set to TRUE for the handler), then you can use this member procedure in the handler procedure. If assemble_lob is set to FALSE for the handler, then you cannot use this member procedure on a row LCR.

To use a DML or error handler to set the value of one or more LOB columns in a row LCR, specify a LOB locator for each LOB column in the value_list parameter. The SET_VALUES member procedure verifies that an ANYDATA encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an ANYDATA encapsulated LOB locator.
- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB columns are represented by the original (nonassembled) row LCRs.
Considerations for XMLType Columns

When processing a row LCR with XMLType columns with a procedure DML handler or error handler, any XMLType and LOB columns in the LCR are always assembled using LOB assembly. You can use this member procedure in the handler procedure on a row LCR that contains one or more XMLType columns.

To use a DML or error handler to set the value of one or more XMLType columns in a row LCR, specify an XMLType for each XMLType column in the value_list parameter. The SET_VALUES member procedure verifies that an ANYDATA encapsulated XMLType is processed with a DML or error handler. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an ANYDATA encapsulated XMLType.
- An attempt is made to set a XMLType incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the XMLType columns are represented by the original (nonassembled) row LCRs.

Syntax

```sql
MEMBER_PROCEDURE SET_VALUES(
    value_type  IN  VARCHAR2,
    value_list  IN  SYS.LCR$_ROW_LIST);
```
Parameters

Table 289-32  SET_VALUES Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_type</td>
<td>The type of values to replace</td>
</tr>
<tr>
<td></td>
<td>Specify old to replace the old values. Specify new to replace the new values.</td>
</tr>
<tr>
<td>value_list</td>
<td>List of values to replace the existing list</td>
</tr>
<tr>
<td></td>
<td>Use a NULL or an empty list to remove all values.</td>
</tr>
<tr>
<td></td>
<td>If the member procedure is used in a procedure DML handler or error handler that uses LOB assembly, then specify one or more LOB locators for LOB columns.</td>
</tr>
</tbody>
</table>

SET_XML_INFORMATION Member Procedure

Sets the XML information for the column.

Syntax

MEMBER PROCEDURE SET_XML_INFORMATION(
    column_name      IN  VARCHAR2,
    xml_information  IN  NUMBER);

Parameters

Table 289-33  SET_XML_INFORMATION Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_name</td>
<td>The name of the column</td>
</tr>
<tr>
<td></td>
<td>An exception is raised if the column value does not exist in the LCR.</td>
</tr>
<tr>
<td>xml_information</td>
<td>Specify one of the following values:</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.NOT_XML CONSTANT NUMBER := 1;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DOC CONSTANT NUMBER := 2;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DIFF CONSTANT NUMBER := 3;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DIFF CONSTANT NUMBER := 3;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DIFF CONSTANT NUMBER := 3;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DIFF CONSTANT NUMBER := 3;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.NOT_XML indicates that the column is not an XMLType column.</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DOC indicates that the column contains an XML document.</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DIFF indicates that the column contains differences between old and new XML documents for an update operation.</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.XML_DIFF indicates that the column contains an XML document that describes a change made by an update operation. This XML document describes the differences in the column's XML document. The entire XML document is not replaced.</td>
</tr>
</tbody>
</table>
289.3.3 Common Subprograms for LCR$_DDL$ RECORD and LCR$_ROW$ RECORD

These functions and procedures are common to both the LCR$_DDL$ RECORD and LCR$_ROW$ RECORD type.

See Also:

For descriptions of the subprograms for these types that are exclusive to each type:

- "LCR$_DDL$ RECORD Type"
- "LCR$_ROW$ RECORD Type"

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_COMMAND_TYPE Member Function</td>
<td>Gets the command type of the logical change record (LCR)</td>
</tr>
<tr>
<td>GET_COMMIT_SCN Member Function</td>
<td>Gets the commit system change number (SCN) of the transaction to which the current LCR belongs</td>
</tr>
<tr>
<td>GET_COMMIT_SCN_FROM_POSITION Static Function</td>
<td>Gets the commit SCN of a transaction from the input position, which is generated by an XStream outbound server</td>
</tr>
<tr>
<td>GET_COMMIT_TIME</td>
<td>Gets the commit time of the transaction to which the current LCR belongs</td>
</tr>
<tr>
<td>GET_COMPATIBLE Member Function</td>
<td>Gets the minimal database compatibility required to support the LCR</td>
</tr>
<tr>
<td>GET_EXTRA_ATTRIBUTE Member Function</td>
<td>Gets the value for the specified extra attribute in the LCR</td>
</tr>
<tr>
<td>GET_OBJECT_NAME Member Function</td>
<td>Gets the name of the object that is changed by the LCR</td>
</tr>
<tr>
<td>GET_OBJECT_OWNER Member Function</td>
<td>Gets the owner of the object that is changed by the LCR</td>
</tr>
<tr>
<td>GET_POSITION Member Function</td>
<td>Gets the position of the current LCR</td>
</tr>
<tr>
<td>GET_ROOT_NAME Member Function</td>
<td>Gets the global name of the root for a CDB.</td>
</tr>
<tr>
<td>GET_SCN Member Function</td>
<td>Gets the SCN of the LCR</td>
</tr>
<tr>
<td>GET_SCN_FROM_POSITION Static Function</td>
<td>Gets the SCN from the input position, which is generated by an XStream outbound server</td>
</tr>
<tr>
<td>GET_SOURCE_DATABASE_NAME Member Function</td>
<td>Gets the source database name.</td>
</tr>
<tr>
<td>GET_SOURCE_TIME Member Function</td>
<td>Gets the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created</td>
</tr>
<tr>
<td>GET_TAG Member Function</td>
<td>Gets the tag for the LCR</td>
</tr>
<tr>
<td>GET_THREAD_NUMBER Member Function</td>
<td>Gets the thread number of the database instance that made the change that is encapsulated in the LCR</td>
</tr>
</tbody>
</table>
### Table 289-34    (Cont.) Summary of Common Subprograms for DDL and Row LCR Types

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_TRANSACTION_ID Member Function</td>
<td>Gets the transaction identifier of the LCR</td>
</tr>
<tr>
<td>IS_NULL_TAG Member Function</td>
<td>Returns Y if the tag for the LCR is NULL, or returns N if the tag for the LCR is not NULL</td>
</tr>
<tr>
<td>SET_COMMAND_TYPE Member Procedure</td>
<td>Sets the command type in the LCR</td>
</tr>
<tr>
<td>SET_EXTRA_ATTRIBUTE Member Procedure</td>
<td>Sets the value for the specified extra attribute in the LCR</td>
</tr>
<tr>
<td>SET_OBJECT_NAME Member Procedure</td>
<td>Sets the name of the object that is changed by the LCR</td>
</tr>
<tr>
<td>SET_OBJECT_OWNER Member Procedure</td>
<td>Sets the owner of the object that is changed by the LCR</td>
</tr>
<tr>
<td>SET_ROOT_NAME Member Procedure</td>
<td>Sets the global name of the root in a CDB.</td>
</tr>
<tr>
<td>SET_SOURCE_DATABASE_NAME Member Procedure</td>
<td>Sets the source database name of the object that is changed by the LCR</td>
</tr>
<tr>
<td>SET_TAG Member Procedure</td>
<td>Sets the tag for the LCR</td>
</tr>
</tbody>
</table>

### GET_COMMAND_TYPE Member Function

Gets the command type of the LCR.

**See Also:**

The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

**Syntax**

```sql
MEMBER FUNCTION GET_COMMAND_TYPE()
RETURN VARCHAR2;
```

### GET_COMMIT_SCN Member Function

Gets the commit system change number (SCN) of the transaction to which the current LCR belongs.

The commit SCN for a transaction is available only during apply or during error transaction execution. This function can be used only in a procedure DML handler, DDL handler, or error handler.

The commit SCN might not be available for an LCR that is part of an incomplete transaction. For example, persistent LCRs might not have a commit SCN. If the commit SCN is not available for an LCR, then this function returns `NULL`.

**Syntax**

```sql
MEMBER FUNCTION GET_COMMIT_SCN()
RETURN NUMBER;
```
GET_COMMIT_SCN_FROM_POSITION Static Function

Gets the commit system change number (SCN) of a transaction from the input position, which is generated by an XStream outbound server.

Syntax

```
STATIC FUNCTION GET_COMMIT_SCN_FROM_POSITION(
    position IN RAW)
RETURN NUMBER;
```

Parameters

Table 289-35 GET_COMMIT_SCN_FROM_POSITION Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>The position You can obtain the position by using the GET_POSITION member function or by querying the DBA_XSTREAM_OUTBOUND_PROGRESS data dictionary view.</td>
</tr>
</tbody>
</table>

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.

GET_COMMIT_TIME

Gets the commit time of the transaction to which the current LCR belongs.

The commit time for a transaction is available only during apply or during error transaction execution. This function can be used only in a procedure DML handler, DDL handler, or error handler.

The commit time might not be available for an LCR that is part of an incomplete transaction. For example, persistent LCRs might not have a commit time. If the commit time is not available for an LCR, then this function returns NULL.

Syntax

```
MEMBER FUNCTION GET_COMMIT_TIME()
RETURN DATE;
```

GET_COMPATIBLE Member Function

Gets the minimal database compatibility required to support the LCR. You control the compatibility of an Oracle database using the COMPATIBLE initialization parameter.

The return value for this function can be one of the following:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>COMPATIBLE Initialization Parameter Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS_STREAMS.COMPATIBLE_9_2</td>
<td>9.2.0</td>
</tr>
</tbody>
</table>
Return Value | COMPATIBLE Initialization Parameter Equivalent
--- | ---
DBMS_STREAMS.COMPATIBLE_10_1 | 10.1.0
DBMS_STREAMS.COMPATIBLE_10_2 | 10.2.0
DBMS_STREAMS.COMPATIBLE_11_1 | 11.1.0
DBMS_STREAMS.COMPATIBLE_11_2 | 11.2.0

DDL LCRs always return DBMS_STREAMS.COMPATIBLE_9_2.

You can use the following functions in the DBMS_STREAMS package for constant compatibility return values:

- The COMPATIBLE_9_2 function returns the DBMS_STREAMS.COMPATIBLE_9_2 constant.
- The COMPATIBLE_10_1 function returns DBMS_STREAMS.COMPATIBLE_10_1 constant.
- The COMPATIBLE_10_2 function returns DBMS_STREAMS.COMPATIBLE_10_2 constant.
- The COMPATIBLE_11_1 function returns DBMS_STREAMS.COMPATIBLE_11_1 constant.
- The COMPATIBLE_11_2 function returns DBMS_STREAMS.COMPATIBLE_11_2 constant.
- The MAX_COMPATIBLE function returns an integer that is greater than the highest possible compatibility constant for the current release of Oracle Database.

You can use these functions with the GET_COMPATIBLE member function for an LCR in rule conditions and apply handlers.

**Note:**

You can determine which database objects in a database are not supported by Oracle Streams by querying the DBA_STREAMS_UNSUPPORTED data dictionary view.

**See Also:**

- Oracle Streams Concepts and Administration for examples of rules that discard changes that are not supported by Oracle Streams
- DBMS_STREAMS and DBMS_STREAMS_ADM
- Oracle Database Reference and Oracle Database Upgrade Guide for more information about the COMPATIBLE initialization parameter

**Syntax**

```sql
MEMBER FUNCTION GET_COMPATIBLE()
RETURN NUMBER;
```
GET_EXTRA_ATTRIBUTE Member Function

Gets the value for the specified extra attribute in the LCR. The returned extra attribute is contained within an ANYDATA instance. You can use the INCLUDE_EXTRA_ATTRIBUTE procedure in the DBMS_CAPTURE_ADM package to instruct a capture process to capture one or more extra attributes.

See Also:
- INCLUDE_EXTRA_ATTRIBUTE Procedure

Syntax

```sql
MEMBER FUNCTION GET_EXTRA_ATTRIBUTE(
    attribute_name  IN  VARCHAR2)
RETURN ANYDATA;
```
Parameters

Table 289-36   GET_EXTRA_ATTRIBUTE Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_name</td>
<td>The name of the extra attribute to return</td>
</tr>
<tr>
<td></td>
<td>Valid names are:</td>
</tr>
<tr>
<td></td>
<td>row_id</td>
</tr>
<tr>
<td></td>
<td>The rowid of the row changed in a row LCR. This attribute is</td>
</tr>
<tr>
<td></td>
<td>not included in DDL LCRs, nor in row LCRs for index-</td>
</tr>
<tr>
<td></td>
<td>organized tables. The type is UROWID.</td>
</tr>
<tr>
<td></td>
<td>serial#</td>
</tr>
<tr>
<td></td>
<td>The serial number of the session that performed the change</td>
</tr>
<tr>
<td></td>
<td>captured in the LCR. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>session#</td>
</tr>
<tr>
<td></td>
<td>The identifier of the session that performed the change</td>
</tr>
<tr>
<td></td>
<td>captured in the LCR. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>thread#</td>
</tr>
<tr>
<td></td>
<td>The thread number of the instance in which the change</td>
</tr>
<tr>
<td></td>
<td>captured in the LCR was performed. Typically, the thread</td>
</tr>
<tr>
<td></td>
<td>number is relevant only in an Oracle Real Application Clusters (Oracle</td>
</tr>
<tr>
<td></td>
<td>RAC) environment. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>tx_name</td>
</tr>
<tr>
<td></td>
<td>The name of the transaction that includes the LCR. The type is VARCHAR2.</td>
</tr>
<tr>
<td></td>
<td>username</td>
</tr>
<tr>
<td></td>
<td>The name of the current user who performed the change</td>
</tr>
<tr>
<td></td>
<td>captured in the LCR. The type is VARCHAR2.</td>
</tr>
</tbody>
</table>

An error is raised if the specified attribute_name is not valid.
If no value exists for the specified extra attribute, then the function returns a NULL.

See Also: Oracle Database PL/SQL Language Reference for more information about the current user

GET_OBJECT_NAME Member Function

Gets the name of the object that is changed by the LCR.

Syntax

MEMBER FUNCTION GET_OBJECT_NAME ()
RETURN VARCHAR2;

GET_OBJECT_OWNER Member Function

Gets the owner of the object that is changed by the LCR.

Syntax

MEMBER FUNCTION GET_OBJECT_OWNER ()
RETURN VARCHAR2;
GET_POSITION Member Function

GET_POSITION Member Function

GET_POSITION Member Function

GET_ROOT_NAME Member Function

GET_ROOT_NAME Member Function

GET_SCN Member Function

GET_SCN Member Function

GET_SCN_FROM_POSITION Static Function

GET_SCN_FROM_POSITION Static Function

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.

GET_SCN_FROM_POSITION Static Function

GET_SCN_FROM_POSITION Static Function

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See Oracle Database XStream Guide.
Parameters

### GET_SCN_FROM_POSITION Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>The position  You can obtain the position by using the GET_POSITION member function or by querying the DBA_XSTREAM_OUTBOUND_PROGRESS data dictionary view.</td>
</tr>
</tbody>
</table>

**Note:**

Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

**GET_SOURCE_DATABASE_NAME** Member Function

Gets the global name of the source database. The source database is the database where the change occurred.

**Syntax**

```sql
MEMBER FUNCTION GET_SOURCE_DATABASE_NAME()
RETURN VARCHAR2;
```

**GET_SOURCE_TIME** Member Function

Gets the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created.

**Syntax**

```sql
MEMBER FUNCTION GET_SOURCE_TIME()
RETURN DATE;
```

**GET_TAG** Member Function

Gets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the DML or DDL change when apply forwarding is used.

**See Also:**

*Oracle Streams Replication Administrator’s Guide* for more information about tags.

---

Oracle Database XStream Guide
Syntax

MEMBER FUNCTION GET_TAG()
RETURN RAW;

GET_THREAD_NUMBER Member Function

Gets the thread number of the database instance that made the change that is encapsulated in the LCR. Typically, the thread number is relevant in an Oracle Real Application Clusters configuration.

See Also:

Oracle Real Application Clusters Administration and Deployment Guide

Syntax

MEMBER FUNCTION GET_THREAD_NUMBER()
RETURN NUMBER;

GET_TRANSACTION_ID Member Function

Gets the transaction identifier of the LCR.

Syntax

MEMBER FUNCTION GET_TRANSACTION_ID()
RETURN VARCHAR2;

IS_NULL_TAG Member Function

Returns Y if the tag for the LCR is NULL, or returns N if the tag for the LCR is not NULL.

See Also:

Oracle Streams Replication Administrator’s Guide for more information about tags

Syntax

MEMBER FUNCTION IS_NULL_TAG()
RETURN VARCHAR2;

SET_COMMAND_TYPE Member Procedure

Sets the command type in the LCR. If the command type specified cannot be interpreted, then this procedure raises an error. For example, changing INSERT to GRANT would raise an error.
See Also:

- The description of the `command_type` parameter in LCR$_DDL_RECORD Constructor Function Parameters
- The description of the `command_type` parameter in LCR$_ROW_RECORD Type
- The “SQL Command Codes” table in the Oracle Call Interface Programmer’s Guide for a complete list of command types

Syntax

```
MEMBER PROCEDURE SET_COMMAND_TYPE(
    command_type  IN  VARCHAR2);
```

Parameter

Table 289-38  SET_COMMAND_TYPE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_type</td>
<td>The command type</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to a non-NULL value.</td>
</tr>
</tbody>
</table>

SET_EXTRA_ATTRIBUTE Member Procedure

Sets the value for the specified extra attribute in the LCR. You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the DBMS_CAPTURE_ADM package to instruct a capture process to capture one or more extra attributes.

See Also:

`INCLUDE_EXTRA_ATTRIBUTE` Procedure

Syntax

```
MEMBER PROCEDURE SET_EXTRA_ATTRIBUTE(
    attribute_name   IN  VARCHAR2,
    attribute_value  IN  ANYDATA);
```
Parameters

Table 289-39  SET_EXTRA_ATTRIBUTE Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute_name</td>
<td>The name of the extra attribute to set</td>
</tr>
<tr>
<td></td>
<td>Valid names are:</td>
</tr>
<tr>
<td></td>
<td>• row_id</td>
</tr>
<tr>
<td></td>
<td>The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, nor in row LCRs for index-organized tables. The type is VARCHAR2.</td>
</tr>
<tr>
<td></td>
<td>• serial#</td>
</tr>
<tr>
<td></td>
<td>The serial number of the session that performed the change captured in the LCR. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>• session#</td>
</tr>
<tr>
<td></td>
<td>The identifier of the session that performed the change captured in the LCR. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>• thread#</td>
</tr>
<tr>
<td></td>
<td>The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment. The type is NUMBER.</td>
</tr>
<tr>
<td></td>
<td>• tx_name</td>
</tr>
<tr>
<td></td>
<td>The name of the transaction that includes the LCR. The type is VARCHAR2.</td>
</tr>
<tr>
<td></td>
<td>• username</td>
</tr>
<tr>
<td></td>
<td>The name of the current user who performed the change captured in the LCR. The type is VARCHAR2.</td>
</tr>
</tbody>
</table>

An error is raised if the specified attribute_name is not valid.

See Also: Oracle Database PL/SQL Language Reference for more information about the current user

<table>
<thead>
<tr>
<th>attribute_value</th>
<th>The value to which the specified extra attribute is set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If set to NULL, then this procedure removes the specified extra attribute from the LCR. To set to NULL, encapsulate the NULL in an ANYDATA instance.</td>
</tr>
</tbody>
</table>

SET_OBJECT_NAME Member Procedure

Sets the name of the object that is changed by the LCR.

Syntax

MEMBER PROCEDURE SET_OBJECT_NAME(
    object_name  IN  VARCHAR2);
Parameter

**Table 289-40  SET_OBJECT_NAME Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_name</td>
<td>The name of the object</td>
</tr>
</tbody>
</table>

**SET_OBJECT_OWNER Member Procedure**
Sets the owner of the object that is changed by the LCR.

**Syntax**

```sql
MEMBER PROCEDURE SET_OBJECT_OWNER(
    object_owner  IN  VARCHAR2);
```

Parameter

**Table 289-41  SET_OBJECT_OWNER Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_owner</td>
<td>The schema that contains the object</td>
</tr>
</tbody>
</table>

**SET_ROOT_NAME Member Procedure**
Sets the global name of the root in a CDB. The setting is the root name for the LCR.

**Syntax**

```sql
MEMBER PROCEDURE SET_ROOT_NAME(
    root_name  IN  VARCHAR2);
```

Parameter

**Table 289-42  SET_ROOT_NAME Procedure Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>root_name</td>
<td>The global name of the root.</td>
</tr>
</tbody>
</table>

**SET_SOURCE_DATABASE_NAME Member Procedure**
Sets the source database name of the object that is changed by the LCR.

**Syntax**

```sql
MEMBER PROCEDURE SET_SOURCE_DATABASE_NAME(
    source_database_name  IN  VARCHAR2);
```
Parameter

Table 289-43  SET_SOURCE_DATABASE_NAME Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_database_name</td>
<td>The source database of the change If you do not include the domain name, then the procedure appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. Set this parameter to a non-NULL value.</td>
</tr>
</tbody>
</table>

SET_TAG Member Procedure

Sets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the change when apply forwarding is used.

See Also:

Oracle Streams Replication Administrator's Guide for more information about tags

Syntax

MEMBER PROCEDURE SET_TAG(
    tag  IN  RAW);

Parameter

Table 289-44  SET_TAG Procedure Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>The binary tag for the LCR The size limit for a tag value is two kilobytes.</td>
</tr>
</tbody>
</table>

289.3.4 LCR$_ROW_LIST Type

This type identifies a list of column values for a row in a table. It uses the LCR$_ROW_UNIT type and is used in the LCR$_ROW_RECORD type.

See Also:

- LCR$_ROW_UNIT Type
- LCR$_ROW_RECORD Type
Syntax

CREATE TYPE SYS.LCR$_ROW_LIST AS TABLE OF SYS.LCR$_ROW_UNIT
/

289.3.5 LCR$_ROW_UNIT Type

This type identifies the value for a column in a row.

It is used in the LCR$_ROW_LIST type.

See Also:

LCR$_ROW_LIST Type

Syntax

CREATE TYPE LCR$_ROW_UNIT AS OBJECT (
  column_name          VARCHAR2(4000),
  data                 ANYDATA,
  lob_information      NUMBER,
  lob_offset           NUMBER,
  lob_operation_size   NUMBER,
  long_information     NUMBER);
/

Attributes

Table 289-45  LCR$_ROW_UNIT Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_name</td>
<td>The name of the column</td>
</tr>
<tr>
<td>data</td>
<td>The data contained in the column</td>
</tr>
<tr>
<td>lob_information</td>
<td>Contains the LOB information for the column and contains one of the following values:</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.NOT_A_LOB CONSTANT NUMBER := 1;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.NULL_LOB CONSTANT NUMBER := 2;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR INLINE_LOB CONSTANT NUMBER := 3;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.EMPTY_LOB CONSTANT NUMBER := 4;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.LOB CHUNK CONSTANT NUMBER := 5;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR LAST LOB CHUNK CONSTANT NUMBER := 6;</td>
</tr>
<tr>
<td>lob_offset</td>
<td>The LOB offset specified in the number of characters for CLOB columns and the number of bytes for BLOB columns</td>
</tr>
<tr>
<td></td>
<td>Valid values are NULL or a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE.</td>
</tr>
</tbody>
</table>
### Table 289-45  (Cont.) LCR$\,\,\,\,\,\,\,\,\,\,\,\,$ROW$\,\,\,\,\,\,\,\,\,\,\,$UNIT Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob_operation_size</td>
<td>If lob_information for the LOB is DBMS_LCR.LAST_LOB_CHUNK, then this parameter can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE. If lob_information is not DBMS_LCR.LAST_LOB_CHUNK and for all other operations, is NULL.</td>
</tr>
<tr>
<td>long_information</td>
<td>Contains the LONG information for the column and contains one of the following values:</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.not_a_long CONSTANT NUMBER := 1;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.null_long CONSTANT NUMBER := 2;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.inline_long CONSTANT NUMBER := 3;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.long_chunk CONSTANT NUMBER := 4;</td>
</tr>
<tr>
<td></td>
<td>DBMS_LCR.last_long_chunk CONSTANT NUMBER := 5;</td>
</tr>
</tbody>
</table>
The Oracle Multimedia ORDAudio object type supports the storage and management of audio data.

**Note:**

Oracle Multimedia ORDAudio is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

Audio data can have different formats, encoding types, compression types, numbers of channels, sampling rates, sample sizes, and playing times (duration) depending upon how the audio data is digitally recorded. Oracle Multimedia ORDAudio can store and retrieve audio data of any data format. Oracle Multimedia ORDAudio can extract metadata from audio data of a variety of popular audio formats. Oracle Multimedia ORDAudio can also extract application attributes and store them in the comments attribute of the object in XML form.

For a complete description of this type within the context of Oracle Multimedia, see ORDAudio in the Oracle Multimedia Reference.
Oracle Multimedia ORDDicom TYPE

The Oracle Multimedia ORDDicom object type supports the storage, management, and manipulation of Digital Imaging and Communications in Medicine (DICOM) data.

**Note:**

Oracle Multimedia ORDDicom was deprecated in Oracle Database 12c Release 2 (12.2). Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The DICOM standard is the dominant standard for radiology imaging and communication, to which all major manufacturers of radiological devices must conform. Oracle Multimedia DICOM provides native support for DICOM format medical images and other objects. These include objects such as single frame and multiframe images, waveforms, slices of 3-D volumes, video segments, and structured reports.

Oracle Multimedia ORDDicom provides methods to extract standard and private DICOM metadata from DICOM content into customizable XML documents, to perform image processing operations such as format conversion and thumbnail image generation, and to create new DICOM objects. Oracle Multimedia ORDDicom also provides methods to check DICOM objects for conformance based on a set of user-specified conformance rules, and to make DICOM objects anonymous based on user-defined rules that specify the set of attributes to be made anonymous and the actions to be taken to make those attributes anonymous.

For a complete description of this type within the context of Oracle Multimedia, see ORDDicom in the Oracle Multimedia DICOM Developer’s Guide.
Oracle Multimedia ORDDoc TYPE

The Oracle Multimedia ORDDoc object type supports the storage and management of heterogeneous media data including image, audio, and video.

**Note:**

Oracle Multimedia ORDDoc is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

Heterogeneous media data can have different formats depending upon the application generating the media data. Oracle Multimedia can store and retrieve media data of any data format. Oracle Multimedia ORDDoc datatype can be used in applications that require you to store different types of heterogeneous media data in the same column so you can build a common metadata index on all the different types of media data. Using this index, you can search across all the different types of heterogeneous media data. However, you cannot use this same search technique if the different types of heterogeneous media data are stored in different types of objects in different columns of relational tables.

For a complete description of this type within the context of Oracle Multimedia, see ORDDoc in the *Oracle Multimedia Reference*. 
Oracle Multimedia ORDImage TYPE

The Oracle Multimedia ORDImage object type supports the storage, management, and manipulation of image data.

**Note:**

Oracle Multimedia ORDImage is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

Digitized images consist of the image data (digitized bits) and attributes that describe and characterize the image data.

The image data (pixels) can have varying depths (bits for each pixel) depending on how the image was captured, and can be organized in various ways. The organization of the image data is known as the data format. Oracle Multimedia ORDImage can store and retrieve image data of any data format. Oracle Multimedia ORDImage can process (cut, scale, and generate thumbnails) of images, convert the format of images, extract properties of images of a variety of popular data formats, and extract and embed application metadata in images.

For a complete description of this type within the context of Oracle Multimedia, see ORDImage in the Oracle Multimedia Reference.
Oracle Multimedia provides support for the SQL/MM Still Image Standard, which supports the storage, retrieval, and modification of images in the database and the ability to locate images using visual predicates.

**Note:**

SQL/MM Still Image was deprecated in Oracle Database 12c Release 2 (12.2). Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

The following object relational types for images and image characteristics are included in this support:

- SI_AverageColor
- SI_Color
- SI_ColorHistogram
- SI_FeatureList
- SI_PositionalColor
- SI_StillImage
- SI_Texture

For a complete description of this type within the context of Oracle Multimedia, see SQL/MM Still Image in the *Oracle Multimedia Reference*. 
Oracle Multimedia ORDVideo TYPE

The Oracle Multimedia ORDVideo object type supports the storage and management of video data.

**Note:**

Oracle Multimedia ORDVideo is deprecated in Oracle Database Release 18c. Oracle recommends that you do not use deprecated features. Support for deprecated features is for backward compatibility only.

Digitized video consists of the video data (digitized bits) and the attributes that describe and characterize the video data. Video applications sometimes associate application-specific information, such as the description of the video training tape, date recorded, instructor's name, producer's name, and so on, within the video data.

The video data can have different formats, compression types, frame rates, frame sizes, frame resolutions, playing times, compression types, number of colors, and bit rates depending upon how the video data was digitally recorded. Oracle Multimedia ORDVideo can store and retrieve video data of any data format. Oracle Multimedia ORDVideo can extract metadata from video data of a variety of popular video formats. Oracle Multimedia ORDVideo can also extract application attributes and store them in the comments attribute of the object in XML form.

For a complete description of this type within the context of Oracle Multimedia, see ORDVideo in the Oracle Multimedia Reference.
MGD_ID Package Types

The **MGD_ID** package provides an extensible framework that supports current radio-frequency ID (RFID) tags with the standard family of EPC bit encodings for the supported encoding types. The **MGD_ID** Package also supports new and evolving tag encodings that are not included in the current EPC standard (EPC v1.1 specification).

The **MGD_ID** package contains several predefined types.

**See Also:**

*Oracle Database Development Guide* for more information.

This chapter contains the following topics:

- Security Model
- Summary of Types
- Summary of MGD_ID Subprograms

The method described in this reference chapter show examples based on the examples shown in the constructor functions.

The examples in this chapter assume that you have run the following set of commands before running the contents of each script:

```
SQL> connect / as sysdba;
Connected.
SQL> create user mgduser identified by mgduser;
SQL> grant connect, resource to mgduser;
SQL> connect mgduser
Enter password: mgduserpassword
Connected.
SQL> set serveroutput on;
```

### 296.1 MGD_ID Package Types Security Model

You must run the *catmgd.sql* script to load the **DBMS_MGD_ID_UTIL** package and create the required Identity Code Package schema objects in the **MGDSYS** schema.

**MGD_ID** is a **MGDSYS**-owned object type. Any **MGD_ID** subprogram called from an anonymous PL/SQL block is run using the privileges of the current user.

A user must be granted connect and resource roles to use the **MGD_ID** object type and its subprograms.

EXECUTE privilege is granted to **PUBLIC** for this ADT: **MGD_ID**.

A public synonym, by the same name, is created for this ADT: **MGD_ID**.
296.2 Summary of Types

This table lists and briefly describes the `MGD_ID` Package object types.

**Table 296-1  MGD_ID Package Object Types**

<table>
<thead>
<tr>
<th>Object Type Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGD_ID_COMPONENT Object Type</td>
<td>Datatype that specifies the name and value pair attributes that define a component</td>
</tr>
<tr>
<td>MGD_ID_COMPONENT_VARRAY Object Type</td>
<td>Datatype that specifies a list of up to 128 components as name-value attribute pairs used in two constructor functions for creating an identity code type object</td>
</tr>
<tr>
<td>MGD_ID Object Type</td>
<td>Represents an <code>MGD_ID</code> object that specifies the category identifier for the code category for this identity code and its list of components</td>
</tr>
</tbody>
</table>

**296.2.1 MGD_ID_COMPONENT Object Type**

The `MGD_ID_COMPONENT` type is a datatype that specifies the name and value pair attributes that define a component.

**Syntax**

```sql
MGD_ID_COMPONENT as object (name VARCHAR2(256),
value VARCHAR2(1024));
```

**Attributes**

**Table 296-2  MGD_ID_COMPONENT Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of component</td>
</tr>
<tr>
<td>value</td>
<td>Value of the component as a character</td>
</tr>
</tbody>
</table>

**Examples**

See the `MGD_ID Constructor Function` for an example.

**296.2.2 MGD_ID_COMPONENT_VARRAY Object Type**

The `MGD_ID_COMPONENT_VARRAY` type is a datatype that specifies a list of up to 128 components as name-value attribute pairs for use in two constructor functions for creating a product code type object with its list of components.

**Syntax**

```sql
MGD_ID_COMPONENT_VARRAY is VARRAY (128) of MGD_ID_COMPONENT;
```
296.2.3 MGD_ID Object Type

The MGD_ID type represents an identity code in an RFID application. This type represents RFID tags with standard EPC bit encoding as well as tag encodings that are not included in the EPC standard.

Syntax

```sql
MGD_ID as object (category_id VARCHAR2(256),
                  components MGD_ID_COMPONENT_VARRAY);
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category identifier for the code category of this code</td>
</tr>
<tr>
<td>components</td>
<td>List of components as name-value attributes</td>
</tr>
</tbody>
</table>

Methods

Table 296-5 describes the methods of the MGD_ID object type.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGD_ID constructor function</td>
<td>Creates an MGD_ID object based on the parameters passed in and returns self as a result</td>
</tr>
<tr>
<td>FORMAT function</td>
<td>Returns the string representation of the MGD_ID in the specified format</td>
</tr>
<tr>
<td>GET_COMPONENT function</td>
<td>Returns the string value of the specified MGD_ID component</td>
</tr>
<tr>
<td>TO_STRING function</td>
<td>Returns the string value of semicolon (;) separated component name value pairs of the MGD_ID object</td>
</tr>
<tr>
<td>TRANSLATE function</td>
<td>Returns the result of the conversion of the identifier from one format to the specified format</td>
</tr>
</tbody>
</table>

Examples

296.3 Summary of MGD_ID Subprograms

This table describes the subprograms in the MGD_ID object type.

All the values and names passed to the procedures defined in the MGD_ID object type are case insensitive unless otherwise mentioned. To preserve the case, enclose the values with double quotation marks.

Table 296-5    MGD_ID Object Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGD_ID Constructor Function</td>
<td>Creates an MGD_ID object based on the parameters passed in and returns self as a result</td>
</tr>
<tr>
<td>FORMAT Function</td>
<td>Returns the string representation of the MGD_ID object in the specified format</td>
</tr>
<tr>
<td>GET_COMPONENT Function</td>
<td>Returns the string value of the specified MGD_ID component</td>
</tr>
<tr>
<td>TO_STRING Function</td>
<td>Returns the string value of semicolon (;) separated component name value pairs of the MGD_ID object</td>
</tr>
<tr>
<td>TRANSLATE Function</td>
<td>Returns the result of the conversion of the identifier from one format to the specified format</td>
</tr>
</tbody>
</table>

296.3.1 MGD_ID Constructor Function

This constructor function constructs an identity code type object, MGD_ID. The constructor function is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

Constructs an MGD_ID object type based on the category ID and a list of components.

```sql
MGD_ID (  
category_id    IN VARCHAR2,  
components      IN MGD_ID_COMPONENT_VARRAY)  
RETURN SELF AS RESULT DETERMINISTIC;
```

Constructs an MGD_ID object type based on the category ID, the identifier string, and the list of additional parameters required to create it.

```sql
MGD_ID (  
category_id    VARCHAR2,  
identifier      VARCHAR2,  
parameter_list  VARCHAR2)  
RETURN SELF AS RESULT DETERMINISTIC;
```

Constructs an MGD_ID object type based on the category name, category version, and a list of components.

```sql
MGD_ID (  
category_name   VARCHAR2,  
category_version VARCHAR2,  
components      IN MGD_ID_COMPONENT_VARRAY)  
RETURN SELF AS RESULT DETERMINISTIC;
```
Constructs an MGD_ID object type based on the category name, category version, the identifier string, and the list of additional parameters required to create it.

MGD_ID(
  category_name VARCHAR2,
  category_version VARCHAR2,
  identifier VARCHAR2,
  parameter_list VARCHAR2)
RETURN SELF AS RESULT DETERMINISTIC;

Parameters

Table 296-6  MGD_ID Constructor Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_id</td>
<td>Category identifier</td>
</tr>
<tr>
<td>components</td>
<td>List of component name value pairs</td>
</tr>
<tr>
<td>category_name</td>
<td>Category name, such as EPC</td>
</tr>
<tr>
<td>category_version</td>
<td>Category version. If NULL, the latest version for the specified category name will be used.</td>
</tr>
<tr>
<td>identifier</td>
<td>Identifier string in any format of an encoding scheme in the specified category. For example, for SGTIN-96 encoding, the identifier can be in the format of BINARY, PURE_IDENTITY, TAG_ENCODING, or LEGACY. Express this identifier as a string according to the appropriate grammar or pattern in the tag data translation (TDT) markup file. For example, a binary string consisting of characters 0 and 1, a URI (either tag-encoding or pure-identity formats), or a serialized legacy code expressed as a string format for input, such as gtin=00037000302414;serial=10419703 for a SGTIN coding scheme.</td>
</tr>
<tr>
<td>parameter_list</td>
<td>List of additional parameters required to create the object in the representation. The list is expressed as a parameter string containing key-value pairs, separated by the semicolon (;) as a delimiter between key-value pairs. For example, for a GTIN code, the parameter string would look as follows: filter=3;companyprefixlength=7;taglength=96</td>
</tr>
</tbody>
</table>

Usage Notes

- Use MGD_ID_UTL.EPC_ENCODINGCATEGORY_ID as category_id.
- If the category is not already registered, an error is raised.
- If the bit_length parameter is NULL, the bit_length is 8* the length of bit_encoding.
- If the component list does not contain all required components, an exception MGD_ID_UTL.e_LackComponent will be thrown.

Examples

The following examples construct identity code type objects.
Construct an **MGD_ID** object (SGTIN-64) passing in the category ID and a list of components.

```sql
-- Contents of constructor11.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category('1');
select MGD_ID('1',
    MGD_ID_COMPONENT_VARRAY(
        MGD_ID_COMPONENT('companyprefix', '0037000'),
        MGD_ID_COMPONENT('itemref', '030241'),
        MGD_ID_COMPONENT('serial', '1041970'),
        MGD_ID_COMPONENT('schemes', 'SGTIN-64'))
) from dual;
call DBMS_MGD_ID_UTL.remove_proxy();

SQL> @constructor11.sql
.
.
MGD_ID('1', MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('companyprefix', '0037000'),
    MGD_ID_COMPONENT('itemref', '030241'),
    MGD_ID_COMPONENT('serial', '1041970'),
    MGD_ID_COMPONENT('schemes', 'SGTIN-64'))
.
.
Constructs an **MGD_ID** object (SGTIN-64) passing in the category ID, the tag identifier, and the list of additional parameters that may be required to create it.

```sql
-- Contents of constructor22.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category('1');
select MGD_ID('1',
    'urn:epc:id:sgtin:0037000.030241.1041970',
    'filter=3; scheme=SGTIN-64') from dual;
call DBMS_MGD_ID_UTL.remove_proxy();

SQL> @constructor22.sql
.
.
MGD_ID('1', MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('filter', '3'),
    MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
    MGD_ID_COMPONENT('companyprefixlength', '7'),
    MGD_ID_COMPONENT('companyprefix', '0037000'),
    MGD_ID_COMPONENT('scheme', 'SGTIN-64'),
    MGD_ID_COMPONENT('serial', '1041970'),
    MGD_ID_COMPONENT('itemref', '030241'))
.
.
Construct an **MGD_ID** object (SGTIN-64) passing in the category name, category version (if NULL, then the latest version will be used), and a list of components.

```sql
-- Contents of constructor33.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
select MGD_ID('EPC', NULL,
    MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('filter', '3'),
    MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
    MGD_ID_COMPONENT('companyprefixlength', '7'),
    MGD_ID_COMPONENT('companyprefix', '0037000'),
    MGD_ID_COMPONENT('scheme', 'SGTIN-64'),
    MGD_ID_COMPONENT('serial', '1041970'),
    MGD_ID_COMPONENT('itemref', '030241'))
) from dual;
call DBMS_MGD_ID_UTL.remove_proxy();

SQL> @constructor33.sql
.
.
MGD_ID('EPC', NULL, MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('filter', '3'),
    MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
    MGD_ID_COMPONENT('companyprefixlength', '7'),
    MGD_ID_COMPONENT('companyprefix', '0037000'),
    MGD_ID_COMPONENT('scheme', 'SGTIN-64'),
    MGD_ID_COMPONENT('serial', '1041970'),
    MGD_ID_COMPONENT('itemref', '030241'))
.
.
```
Constructs an `MGD_ID` object (SGTIN-64) passing in the category name and category version, the tag identifier, and the list of additional parameters that may be required to create it.

```sql
-- Contents of constructor4.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
select MGD_ID('EPC', NULL, 'urn:epc:id:sgtin:0037000.030241.1041970', 'filter=3;scheme=SGTIN-64') from dual;
call DBMS_MGD_ID_UTL.remove_proxy();
```

### 296.3.2 FORMAT Function

This function returns the string representation of the `MGD_ID` object in the specified format.

**Syntax**

```
FORMAT (parameter_list IN VARCHAR2,
        output_format IN VARCHAR2)
RETURN VARCHAR2 DETERMINISTIC;
```
Parameters

Table 296-7  FORMAT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| parameter_list | List of additional parameters required to create the object in the representation. The list is expressed as a parameter string containing key-value pairs, separated by the semicolon (;) as a delimiter between key-value pairs. For example, for a GTIN code, the parameter string would look as follows:  
  filter=3;companyprefixlength=7;taglength=96 |
| output_format | One of the supported output formats into which an MGD_ID component is formatted:  
  • BINARY  
  • LEGACY  
  • TAG_ENCODING  
  • PURE.IDENTITY  
  • ONS_HOSTNAME |

Examples

See the example for the GET_COMPONENT Function.

296.3.3 GET_COMPONENT Function

This function returns the value of the specified MGD_ID component.

Syntax

```
GET_COMPONENT (  
    component_name  IN  VARCHAR2)  
RETURN VARCHAR2 DETERMINISTIC;
```

Parameters

Table 296-8  GET_COMPONENT Function Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component_name</td>
<td>Name of component</td>
</tr>
</tbody>
</table>

Usage Notes

- If the code is an invalid code, meaning its structure is not defined in the metadata table, an error is raised.
- If the code is valid, but it does not contain the required component, NULL is returned.

Examples

The following example returns the general manager, object class, and serial number components for this GID-96 identity component:
--Contents of get_components.sql file

-- Contents of get_components.sql file

call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
DECLARE
  id  MGD_ID;
BEGIN
  DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);
  DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
  ---------------
  --PURE_IDENTITY
  ---------------
  dbms_output.put_line('..Testing constructor with pure identity');
  -----------------------------------------------------------------------------
  -- PURE_IDENTITY representation can be translated to BINARY and
  -- TAG_ENCODING ONLY when BOTH scheme and filer are provided.
  -----------------------------------------------------------------------------
  id := MGD_ID('EPC', NULL, 'urn:epc:id:sgtin:0037000.030241.1041970', 'scheme=SGTIN-64;filter=3');
  dbms_output.put_line(id.to_string);
  dbms_output.put_line('filter         = ' || id.get_component('filter'));
  dbms_output.put_line('company prefix = ' || id.get_component('companyprefix'));
  dbms_output.put_line('itemref        = ' || id.get_component('itemref'));
  dbms_output.put_line('serial         = ' || id.get_component('serial'));
  dbms_output.put_line('BINARY format  = ' || id.format(NULL, 'BINARY'));
  dbms_output.put_line('PURE_IDENTITY format  = ' || id.format(NULL, 'PURE_IDENTITY'));
  dbms_output.put_line('TAG_ENCODING format   = ' || id.format(NULL, 'TAG_ENCODING'));
END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();
SQL> @get_components.sql
.
.
..Testing constructor with pure identity

 category_id =1;filter = 3;schemes = SGTIN-64;companyprefixlength = 7;companyprefix = 0037000;scheme = SGTIN-64;serial = 1041970;itemref = 030241
 filter = 3
 company prefix = 0037000
 itemref = 030241
 serial = 1041970
 BINARY format =10011000000000000010000011101100010000011111110011000110010
 PURE_IDENTITY format = urn:epc:id:sgtin:0037000.030241.1041970
 TAG_ENCODING format = urn:epc:tag:sgtin-64:3.0037000.030241.1041970
Pl/SQL procedure successfully completed.
.
.

296.3.4 TO_STRING Function

This function returns the semicolon (;) separated component name value pairs of the MGD_ID object.

Syntax

TO_STRING
RETURN VARCHAR2;

Examples

The following example converts the MGD_ID object into a string value:
```
-- Contents of tostring3.sql file
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
DECLARE
    id MGD_ID;
BEGIN
    DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
    dbms_output.put_line('Testing to_string');
    id := mgd_id('EPC', NULL, 'urn:epc:id:gid:0037000.30241.1041970', 'scheme=GID-96');
    dbms_output.put_line('mgd_id object as a string');
    dbms_output.put_line(id.to_string);
END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();
connect / as sysdba;
drop user mgduser cascade;
```

```
PL/SQL procedure successfully completed.
```

### 296.3.5 TRANSLATE Function

This static function translates between different representations directly without first constructing an MGD_ID object.

This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

**Syntax**

Converts the identifier in one format to another given the category name, the tag identifier, the parameter list, and the output format.

```
TRANSLATE (category_name IN VARCHAR2,
             identifier IN VARCHAR2,
             parameter_list IN VARCHAR2,
             output_format IN VARCHAR2) RETURN VARCHAR2 DETERMINISTIC;
```

Converts the identifier in one format to another given the category name, category version, the tag identifier, the parameter list, and the output format.

```
TRANSLATE (category_name IN VARCHAR2,
             category_version IN VARCHAR2,
             identifier IN VARCHAR2,
             parameter_list IN VARCHAR2,
             output_format IN VARCHAR2) RETURN VARCHAR2 DETERMINISTIC;
```
Parameters

Table 296-9  TRANSLATE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category_name</td>
<td>Name of category</td>
</tr>
<tr>
<td>category_version</td>
<td>Category version. If NULL, the latest version of the specified category name will be used.</td>
</tr>
<tr>
<td>identifier</td>
<td>EPC identifier, expressed as a string in accordance with one of the grammars or patterns in the TDT markup file. For example, a binary string consisting of characters 0 and 1, a URI (either tag-encoding or pure-identity formats), or a serialized legacy code expressed as a string format for input, such as gtin=00037000302414;serial=10419703 for a SGTIN coding scheme.</td>
</tr>
<tr>
<td>parameter_list</td>
<td>List of additional parameters required to create the object in the representation. The list is expressed as a parameter string containing key-value pairs, separated by the semicolon (;) as a delimiter between key-value pairs. For example, for a GTIN code, the parameter string would look as follows: filter=3;companyprefixlength=7;taglength=96</td>
</tr>
<tr>
<td>output_format</td>
<td>One of the supported output formats into which an MGD_ID component shall be converted:</td>
</tr>
<tr>
<td></td>
<td>· BINARY</td>
</tr>
<tr>
<td></td>
<td>· LEGACY</td>
</tr>
<tr>
<td></td>
<td>· TAG_ENCODING</td>
</tr>
<tr>
<td></td>
<td>· PURE_IDENTITY</td>
</tr>
<tr>
<td></td>
<td>· ONS_HOSTNAME</td>
</tr>
</tbody>
</table>

Usage Notes

When converting from a pure identity representation to a binary representation, the filter value must be supplied as a value using the parameter_list parameter.

Examples

The following examples translates one GID-96 representation into another:

```sql
-- Contents of translate1.sql file
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.example.com', '80');
DECLARE
  id MGD_ID;
BEGIN
  DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
  dbms_output.put_line('Category ID is EPC, Identifier is BINARY, Output format is BINARY');
  dbms_output.put_line(mgd_id.translate('EPC', NULL,'0011010100000000000010010000100010000000011101100010000100000000000000001111111111011001010010', NULL, 'BINARY'));
  dbms_output.put_line('Category ID is EPC, Identifier is BINARY, Output format is PURE_IDENTITY');
  dbms_output.put_line(mgd_id.translate('EPC', NULL,'0011010100000000000010010000100010000000011101100010000100000000000000001111111111011001010010', NULL, 'PURE_IDENTITY'));
```
dbms_output.put_line('Category ID is EPC, Identifier is BINARY, Output format is TAG_ENCODING');
dbms_output.put_line(
    mgd_id.translate('EPC',
    'urn:epc:tag:gid-96:0037000.30241.1041970',
    'urn:epc:id:gid:0037000.30241.1041970',
    NULL, 'BINARY'));
dbms_output.put_line('Category ID is EPC, Identifier is BINARY, Output format is PURE_IDENTITY');
dbms_output.put_line(
    mgd_id.translate('EPC',
    'urn:epc:id:gid:0037000.30241.1041970',
    'urn:epc:id:gid:0037000.30241.1041970',
    NULL, 'PURE_IDENTITY'));

END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();
Category ID is EPC, Identifier is TAG_ENCODING, Output format is TAG_ENCODING
urn:epc:tag:gid-96:0037000.30241.1041970
Category ID is EPC, Identifier is PURE_IDENTITY, Output format is BINARY
00110101000000000000100100001000000000000111011000100010000000000000000000001111110011000110010
Category ID is EPC, Identifier is PURE_IDENTITY, Output format is PURE_IDENTITY
urn:epc:id:gid:0037000.30241.1041970
Category ID is EPC, Identifier is PURE_IDENTITY, Output format is TAG_ENCODING
urn:epc:tag:gid-96:0037000.30241.1041970
PL/SQL procedure successfully completed.
Rule TYPES

This chapter describes the types used with rules, rule sets, and evaluation contexts.

This chapter contains the following topics:

• Overview
• Security Model
• Summary of Rule Types

297.1 Rule TYPES Overview

These types are used in rules and enable clients to evaluate rules with the rules engine.

See Also:
• DBMS_RULE
• DBMS_RULE_ADM
• Oracle Streams Extended Examples for examples that use rule types

297.2 Rule TYPES Security Model

PUBLIC is granted EXECUTE privilege on these RULE types.

See Also:
Oracle Database Security Guide for more information about user group PUBLIC

297.3 Summary of Rule Types

This table lists the Rule Types and briefly describes them.

| Table 297-1   | Rule Types
|---------------|-------------
| Type          | Description                                |
| RE$ATTRIBUTE_VALUE Type | Specifies the value of a variable attribute |
| RE$ATTRIBUTE_VALUE_LIST Type | Identifies a list of attribute values       |
| RE$COLUMN_VALUE Type      | Specifies the value of a table column       |
Table 297-1  (Cont.) Rule Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE$COLUMN_VALUE_LIST Type</td>
<td>Identifies a list of column values</td>
</tr>
<tr>
<td>RE$NAME_ARRAY Type</td>
<td>Identifies a list of names</td>
</tr>
<tr>
<td>RE$NAME_ARRAY Type</td>
<td>Identifies a list of name-value pairs</td>
</tr>
<tr>
<td>RE$NV_LIST Type</td>
<td>Identifies an object containing a list of name-value pairs and methods that operate on this list. This object type is used to represent the event context and the action context for a rule</td>
</tr>
<tr>
<td>RE$NV_NODE Type</td>
<td>Identifies a name-value pair</td>
</tr>
<tr>
<td>RE$RULE_HIT Type</td>
<td>Specifies a rule found because of evaluation</td>
</tr>
<tr>
<td>RE$RULE_HIT_LIST Type</td>
<td>Identifies a list of rules found because of evaluation</td>
</tr>
<tr>
<td>RE$RULE_NAME Type</td>
<td>Allows users to pass a list of rule as argument.</td>
</tr>
<tr>
<td>RE$RULE_NAME_LIST Type</td>
<td>Identifies a list of Rule Name values</td>
</tr>
<tr>
<td>RE$TABLE_ALIAS Type</td>
<td>Provides the table corresponding to an alias used in a rule evaluation context</td>
</tr>
<tr>
<td>RE$TABLE_ALIAS_LIST Type</td>
<td>Identifies a list of table aliases used in a rule evaluation context</td>
</tr>
<tr>
<td>RE$TABLE_VALUE Type</td>
<td>Specifies the value of a table row using a ROWID</td>
</tr>
<tr>
<td>RE$TABLE_VALUE_LIST Type</td>
<td>Identifies a list of table values</td>
</tr>
<tr>
<td>RE$VALUE Type</td>
<td>Represents one datapoint value element</td>
</tr>
<tr>
<td>RE$VALUE_LIST Type</td>
<td>Defines a list of RE$VALUE type</td>
</tr>
<tr>
<td>RE$VARIABLE_TYPE Type</td>
<td>Provides the type of a variable used in a rule evaluation context</td>
</tr>
<tr>
<td>RE$VARIABLE_TYPE_LIST Type</td>
<td>Identifies a list of variables and their types used in a rule evaluation context</td>
</tr>
<tr>
<td>RE$VARIABLE_VALUE Type</td>
<td>Specifies the value of a variable</td>
</tr>
<tr>
<td>RE$VARIABLE_VALUE_LIST Type</td>
<td>Identifies a list of variable values</td>
</tr>
</tbody>
</table>

Rule types are used with the following Oracle-supplied PL/SQL packages:

- DBMS_RULE
- DBMS_RULE_ADM

You can use the DBMS_RULE_ADM package to create and administer rules, rule sets, and evaluation contexts, and you can use the DBMS_RULE package to evaluate rules.

When you use Oracle Streams, rules determine which changes are captured by a capture process, which messages are propagated by a propagation, which messages are applied by an apply process, and which messages are dequeued by a messaging client. The following Oracle Streams packages use rules:

- DBMS_APPLY_ADM
- DBMS_CAPTURE_ADM
- DBMS_PROPAGATION_ADM
- DBMS_STREAMS
297.3.1 RE$ATTRIBUTE_VALUE Type

The RE$ATTRIBUTE_VALUE type specifies the value of a variable attribute.

Note:
Enclose the variable name and attribute name in double quotation marks (") if the name contains special characters.

Syntax

TYPE SYS.RE$ATTRIBUTE_VALUE (  
  variable_name    VARCHAR2(32),  
  attribute_name   VARCHAR2(4000),  
  attribute_value  ANYDATA);  

Attributes

Table 297-2  RE$ATTRIBUTE_VALUE Attributes  

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>Specifies the variable used in a rule</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Specifies the attribute name. The attribute name can be a multi-component name, such as a1.b2.c3.</td>
</tr>
<tr>
<td>attribute_value</td>
<td>Specifies the attribute value</td>
</tr>
</tbody>
</table>

297.3.2 RE$ATTRIBUTE_VALUE_LIST Type

The RE$ATTRIBUTE_VALUE_LIST type identifies a list of attribute values.

Syntax

TYPE SYS.RE$ATTRIBUTE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$ATTRIBUTE_VALUE;
297.3.3 RE$COLUMN_VALUE Type

RE$COLUMN_VALUE type specifies the value of a table column.

Note:
Enclose the column name in double quotation marks (" ") if the name contains special characters.

Syntax

```
TYPE SYS.RE$COLUMN_VALUE (  
  table_alias   VARCHAR2(32), 
  column_name   VARCHAR2(4000), 
  column_value  ANYDATA); 
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_alias</td>
<td>Specifies the alias used for the table in a rule</td>
</tr>
<tr>
<td>column_name</td>
<td>Specifies the column name</td>
</tr>
<tr>
<td>column_value</td>
<td>Specifies the column value</td>
</tr>
</tbody>
</table>

297.3.4 RE$COLUMN_VALUE_LIST Type

RE$COLUMN_VALUE_LIST type identifies a list of column values.

Syntax

```
TYPE SYS.RE$COLUMN_VALUE_LIST AS VARRAY(1024) OF SYS.RE$COLUMN_VALUE; 
```

297.3.5 RE$NAME_ARRAY Type

RE$NAME_ARRAY type identifies a list of names.

Syntax

```
TYPE SYS.RE$NAME_ARRAY AS VARRAY(1024) OF VARCHAR2(30); 
```

297.3.6 RE$NV_ARRAY Type

RE$NV_ARRAY type identifies a list of name-value pairs.

Syntax

```
TYPE SYS.RE$NV_ARRAY AS VARRAY(1024) OF SYS.RE$NV_NODE; 
```
297.3.7 RE$NV_LIST Type

identifies an object containing a list of name-value pairs and methods that operate on this list.

This object type is used to represent the event context for rule set evaluation and the action context for a rule.

Syntax

```
TYPE SYS.RE$NV_LIST AS OBJECT(
  actx_list SYS.RE$NV_ARRAY);
```

Attributes

Table 297-4  RE$NV_LIST Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actx_list</td>
<td>The list of name-value pairs</td>
</tr>
</tbody>
</table>

RE$NV_LIST Subprograms

This section describes the following member procedures and member functions of the SYS.RE$NV_LIST type:

- ADD_PAIR Member Procedure
- GET_ALL_NAMES Member Function
- GET_VALUE Member Function
- REMOVE_PAIR Member Procedure

ADD_PAIR Member Procedure

Adds a name-value pair to the list of name-value pairs.

**Note:**

Enclose the name in double quotation marks ("”) if the name contains special characters.

Syntax

```
MEMBER PROCEDURE ADD_PAIR(
  name   IN  VARCHAR2,
  value  IN  ANYDATA);
```

Parameters
### Table 297-5  ADD_PAIR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name in the name-value pair being added to the list. If the name exists in the list, then this procedure raises an error.</td>
</tr>
<tr>
<td>value</td>
<td>The value in the name-value pair being added to the list</td>
</tr>
</tbody>
</table>

### GET_ALL_NAMES Member Function

Returns a list of all the names in the name-value pair list.

**Syntax**

```sql
MEMBER FUNCTION GET_ALL_NAMES()
RETURN SYS.RE$NAME_ARRAY;
```

### GET_VALUE Member Function

Returns the value for the specified name in a name-value pair list.

**Note:**

Enclose the name in double quotation marks (") if the name contains special characters.

**Syntax**

```sql
MEMBER FUNCTION GET_VALUE(
    name  IN  VARCHAR2)
RETURN ANYDATA;
```

### Parameters

#### Table 297-6  GET_VALUE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name whose value to return</td>
</tr>
</tbody>
</table>

### REMOVEPAIR Member Procedure

Removes the name-value pair with the specified name from the name-value pair list.

**Note:**

Enclose the name in double quotation marks (") if the name contains special characters.
Syntax
MEMBER PROCEDURE REMOVE_PAIR(
   name  IN  VARCHAR2);

Parameters

Table 297-7  REMOVE_PAIR Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the pair to remove</td>
</tr>
</tbody>
</table>

297.3.8 RE$NV_NODE Type

RE$COLUMN_VALUE type identifies a name-value pair.

Note:
Enclose the name in double quotation marks (") if the name contains special characters.

Syntax
TYPE SYS.RE$NV_NODE (  
nvn_name   VARCHAR2(30),
nvn_value  ANYDATA);

Attributes

Table 297-8  RE$NV_NODE Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvn_name</td>
<td>Specifies the name in the name-value pair</td>
</tr>
<tr>
<td>nvn_value</td>
<td>Specifies the value in the name-value pair</td>
</tr>
</tbody>
</table>

297.3.9 RE$RULE_HIT Type

The RE$RULE_HIT type specifies a rule found because of an evaluation.

See Also:

- CREATE_RULE Procedure
- ALTER_RULE Procedure
Syntax

TYPE SYS.RE$RULE_HIT (  
    rule_name            VARCHAR2(65),  
    rule_action_context  RE$NV_LIST);  

Attributes

Table 297-9  RE$RULE_HIT Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The rule name in the form schema_name.rule_name. For example, a rule named employee_rule in the hr schema is returned in the form &quot;hr&quot;.&quot;employee_rule&quot;.</td>
</tr>
<tr>
<td>rule_action_context</td>
<td>The rule action context as specified in the CREATE_RULE or ALTER_RULE procedure of the DBMS_RULE_ADM package</td>
</tr>
</tbody>
</table>

297.3.10 RE$RULE_HIT_LIST Type

The RE$RULE_HIT_LIST type identifies a list of rules found because of an evaluation.

Syntax

CREATE OR REPLACE TYPE SYS.RE$RULE_HIT_LIST AS VARRAY(1024) OF SYS.RE$RULE_HIT;

297.3.11 RE$RULE_NAME Type

The RE$RULE_HIT type is create to allow use to pass a list of rule as argument.

Syntax

CREATE OR REPLACE TYPE SYS.RE$RULE_NAME AS OBJECT(  
    rule_name            IN      varchar2,  
    evaluation_context   IN      varchar2);  

Attributes

Table 297-10  RE$RULE_NAME Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_name</td>
<td>The rule name in the form SCHEMA_NAME.RULE_NAME. For example, a rule named employee_rule in the hr schema is returned in the form &quot;hr&quot;.&quot;employee_rule&quot;.</td>
</tr>
<tr>
<td>Evaluation_context</td>
<td>The evaluation context as specified in the CREATE_RULE or ALTER_RULE procedure of the DBMS_RULE_ADM package</td>
</tr>
</tbody>
</table>
297.3.12 RE$RULE_NAME_LIST Type

**RE$RULE_HIT_LIST** type identifies a list of Rule Name values.

**Syntax**

```sql
CREATE OR REPLACE TYPE SYS.RE$RULE_NAME_LIST AS VARRAY(1024) OF SYS.RE$RULE_NAME;
```

297.3.13 RE$TABLE_ALIAS Type

**RE$TABLE_ALIAS** type provides the table corresponding to an alias used in a rule evaluation context.

A specified table name must satisfy the schema object naming rules.

**Note:**

Enclose the table name in double quotation marks (") if the name contains special characters.

**See Also:**

*Oracle Database SQL Language Reference* for information about schema object naming rules.

**Syntax**

```sql
TYPE SYS.RE$TABLE_ALIAS IS OBJECT(
    table_alias  VARCHAR2(32),
    table_name   VARCHAR2(194));
```

**Attributes**

**Table 297-11  RE$TABLE_ALIAS Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_alias</td>
<td>The alias used for the table in a rule</td>
</tr>
</tbody>
</table>
Table 297-11  (Cont.) RE$TABLE_ALIAS Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The table name referred to by the alias. A synonym can be specified. The table name is resolved in the evaluation context schema. The format is one of the following:</td>
</tr>
<tr>
<td></td>
<td>scheme_name.table_name</td>
</tr>
<tr>
<td></td>
<td>table_name</td>
</tr>
<tr>
<td></td>
<td>For example, if the schema_name is hr and the table_name is employees, then enter the following:</td>
</tr>
<tr>
<td></td>
<td>hr.employees</td>
</tr>
</tbody>
</table>

297.3.14 RE$TABLE_ALIAS_LIST Type

The RE$TABLE_ALIAS_LIST type identifies a list of table aliases used in a rule evaluation context.

Syntax

```sql
TYPE SYS.RE$TABLE_ALIAS_LIST AS VARRAY(1024) OF SYS.RE$TABLE_ALIAS;
```

297.3.15 RE$TABLE_VALUE Type

The RE$TABLE_VALUE type specifies the value of a table row using a ROWID.

Syntax

```sql
TYPE SYS.RE$TABLE_VALUE(
    table_alias  VARCHAR2(32),
    table_rowid  VARCHAR2(18));
```

Attributes

Table 297-12  RE$TABLE_VALUE Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_alias</td>
<td>Specifies the alias used for the table in a rule</td>
</tr>
<tr>
<td>table_rowid</td>
<td>Specifies the rowid for the table row</td>
</tr>
</tbody>
</table>
297.3.16 RE$TABLE_VALUE_LIST Type

The RE$TABLE_VALUE_LIST type identifies a list of table values.

Note:
Each table alias in the list in the list must be unique.

Syntax

```sql
TYPE SYS.RE$TABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$TABLE_VALUE;
```

297.3.17 RE$VALUE Type

The RE$VALUE type represents one datapoint value element.

Syntax

```sql
TYPE SYS.RE$VALUE (  
table_values          IN       SYS.RE$TABLE_VALUE_LIST:= NULL,  
column_values         IN       SYS.RE$COLUMN_VALUE_LIST:=NULL,  
variable_values       IN       SYS.RE$VARIABLE_VALUE_LIST:=NULL,  
attribute_values      IN       SYS.RE$ATTRIBUTE_VALUE_LIST:=NULL)
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_values</td>
<td>ROWID of table row for expression evaluation</td>
</tr>
<tr>
<td>column_values</td>
<td>Values of columns referred in the expression</td>
</tr>
<tr>
<td>variable_values</td>
<td>Values of variables referred in expression</td>
</tr>
<tr>
<td>attribute_values</td>
<td>Values of attributes referred in expression</td>
</tr>
</tbody>
</table>

297.3.18 RE$VALUE_LIST Type

RE$TABLE_VALUE_LIST type is a varray for defining a list of RE$VALUE type. It represents a list of datapoint element.

Syntax

```sql
CREATE OR REPLACE TYPE SYS.RE$VALUE_LIST AS VARRAY(1024) OF SYS.RE$VALUE;
```
### 297.3.19 RE$VARIABLE_TYPE Type

The `RE$VARIABLE_TYPE` type provides the type of a variable used in a rule evaluation context. A specified variable name must satisfy the schema object naming rules.

#### Note:

Enclose the variable name in double quotation marks (""") if the name contains special characters.

#### See Also:

*Oracle Database SQL Language Reference* for information about schema object naming rules

#### Syntax

```sql
TYPE SYS.RE$VARIABLE_TYPE (  
  variable_name     VARCHAR2(32),
  variable_type     VARCHAR2(4000),
  variable_value_function VARCHAR2(228),
  variable_method_function VARCHAR2(228));
```

#### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>The variable name used in a rule</td>
</tr>
<tr>
<td>variable_type</td>
<td>The type that is resolved in the evaluation context schema. Any valid Oracle built-in datatype, user-defined type, or Oracle-supplied type can be specified. See the <em>Oracle Database SQL Language Reference</em> for more information about these types.</td>
</tr>
<tr>
<td>variable_value_function</td>
<td>A value function that can be specified for implicit variables. A synonym can be specified. The function name is resolved in the evaluation context schema. It is executed on behalf of the owner of a rule set using the evaluation context or containing a rule that uses the evaluation context. See the &quot;Usage Notes&quot; for more information.</td>
</tr>
</tbody>
</table>
### Table 297-14 (Cont.) RES$VARIABLE_TYPE Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_method_function</td>
<td>Specifies a value function, which can return the result of a method invocation. Specifying such a function can speed up evaluation, if there are many simple rules that invoke the method on the variable. The function can be a synonym or a remote function. The function name is resolved in the evaluation context schema. It is executed on behalf of the owner of a rule set using the evaluation context or containing a rule that uses the evaluation context. See the &quot;Usage Notes&quot; for more information.</td>
</tr>
</tbody>
</table>

#### Usage Notes

The functions for both the for the `variable_value_function` parameter and `variable_method_function` parameter have the following format:

```
schema_name.package_name.function_name@dblink
```

Any of the following parts of the format can be omitted: `schema_name`, `package_name`, and `@dblink`.

For example, if the `schema_name` is `hr`, the `package_name` is `var_pac`, the `function_name` is `func_value`, and the `dblink` is `dbs1.net`, then enter the following:

```
hr.var_pac.func_value@dbs1.net
```

The following sections describe the signature of the functions.

#### Signature for variable_value_function

The function must have the following signature:

```
FUNCTION variable_value_function_name(
    evaluation_context_schema  IN VARCHAR2,
    evaluation_context_name    IN VARCHAR2,
    variable_name              IN VARCHAR2,
    event_context              IN SYS.RE$NV_LIST )
RETURN SYS.RE$VARIABLE_VALUE;
```

#### Signature for variable_method_function

This function must have the following signature:

```
FUNCTION variable_method_function_name(
    evaluation_context_schema  IN VARCHAR2,
    evaluation_context_name    IN VARCHAR2,
    variable_value             IN SYS.RE$VARIABLE_VALUE,
    method_name                IN VARCHAR2,
    event_context              IN SYS.RE$NV_LIST)
RETURN SYS.RE$ATTRIBUTE_VALUE;
```
297.3.20 RE$VARIABLE_TYPE_LIST Type

The RE$VARIABLE_TYPE_LIST type identifies a list of variables and their types used in a rule evaluation context.

Syntax

```
TYPE SYS.RE$VARIABLE_TYPE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_TYPE;
```

297.3.21 RE$VARIABLE_VALUE Type

The RE$VARIABLE_VALUE type specifies the value of a variable.

- **Note:**
  Enclose the variable name in double quotation marks (""") if the name contains special characters.

Syntax

```
TYPE SYS.RE$VARIABLE_VALUE (  
    variable_name  VARCHAR2(32),  
    variable_data  ANYDATA);
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>Specifies the variable name used in a rule</td>
</tr>
<tr>
<td>variable_data</td>
<td>Specifies the data for the variable value</td>
</tr>
</tbody>
</table>

297.3.22 RE$VARIABLE_VALUE_LIST Type

The RE$VARIABLE_VALUE_LIST type identifies a list of variable values.

Syntax

```
TYPE SYS.RE$VARIABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_VALUE;
```
SODA Types

There are several SODA types: SODA_DOCUMENT_T, SODA_COLLECTION_T, SODA_OPERATION_T, and SODA_CURSOR_T. SODA_DOCUMENT_T and SODA_COLLECTION_T represent two primary abstractions provided by SODA: document and collections. SODA_OPERATION_T is used for specifying condition of operations on the collection. SODA_CURSOR_T is a cursor over results of read operations on the collection.

This chapter contains the following topics:

• SODA Types Overview
• SODA Types Security Model

298.1 SODA Types Overview

There are several SODA types: SODA_DOCUMENT_T, SODA_COLLECTION_T, SODA_OPERATION_T, and SODA_CURSOR_T. SODA_DOCUMENT_T and SODA_COLLECTION_T represent two primary abstractions provided by SODA: document and collections. SODA_OPERATION_T is used for specifying condition of operations on the collection. SODA_CURSOR_T is a cursor over results of read operations on the collection.

See Also:

• Oracle Database SODA for PL/SQL Developer’s Guide
• DBMS_SODA

298.2 SODA Types Security Model

The SODA Types are available to users with the SODA_APP role.

All SODA types are SYS types. PUBLIC is granted EXECUTE privilege on the SODA types described in this chapter:

• TYPE SODA_Collection_T
• TYPE SODA_Document_T
• TYPE SODA_Operation_T
• TYPE SODA_Cursor_T
298.3 Summary of SODA Types

This chapter lists the SODA types and describes them.

Table 298-1  SODA Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SODA_Collection_T Type</td>
<td>This SODA type represents a SODA collection. This type is not persistable.</td>
</tr>
<tr>
<td>SODA_Document_T Type</td>
<td>This SODA type represents a document with content, usually in JSON format. This type is not persistable.</td>
</tr>
<tr>
<td>SODA_Operation_T Type</td>
<td>This SODA type performs read/write operations, such as document finds with filtering and pagination, removes, and replaces on a SODA collection. This type is not persistable.</td>
</tr>
<tr>
<td>SODA_Cursor_T Type</td>
<td>This SODA type represents the result set of documents. This type is not persistable.</td>
</tr>
</tbody>
</table>

298.3.1 SODA_Collection_T Type

This SODA type represents a SODA collection. A reference of SODA collection can only be obtained by either calling DBMS_SODA.CREATE_COLLECTION() or DBMS_SODA.OPEN_COLLECTION().

Table 298-2  SODA_Collection_T Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_INDEX Function</td>
<td>Creates an index using an index specification expressed in JSON. Three types of specifications are supported. Each specifying a different type of index: for B-tree, JSON search with Data Guide, and Spatial.</td>
</tr>
<tr>
<td>DROP_INDEX Function</td>
<td>Drops the named index.</td>
</tr>
<tr>
<td>FIND Function</td>
<td>Returns the SODA_OPERATION_T object. This is the only way to get the reference of SODA_Operation_T as there is no constructor.</td>
</tr>
<tr>
<td>FIND_ONE Function</td>
<td>Fetches the document matching the key.</td>
</tr>
<tr>
<td>GET_DATA_GUIDE Function</td>
<td>Returns the JSON data guide as a CLOB.</td>
</tr>
<tr>
<td>GET_METADATA Function</td>
<td>Returns the metadata of the collection in JSON format.</td>
</tr>
<tr>
<td>GET_NAME Function</td>
<td>Returns the name of the collection.</td>
</tr>
<tr>
<td>INSERT_ONE Function</td>
<td>Inserts a document into the collection.</td>
</tr>
</tbody>
</table>
Table 298-2 (Cont.) SODA_Collection_T Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT_ONE_AND_GET Function</td>
<td>Inserts a document into the collection and returns a result document with all components except for content.</td>
</tr>
<tr>
<td>REMOVE_ONE Function</td>
<td>Removes the document matching the key.</td>
</tr>
<tr>
<td>REPLACE_ONE Function</td>
<td>Replaces the content and (optionally) the media type of the document matching the key.</td>
</tr>
<tr>
<td>REPLACE_ONE_AND_GET Function</td>
<td>Replaces the content and (optionally) the media type of the document matching the key and returns a result document with all components (except content).</td>
</tr>
</tbody>
</table>

298.3.1.1 CREATE_INDEX Function

This function creates an index using an index specification expressed in JSON. Three types of specifications are supported. Each specifying a different type of index: for B-tree, JSON search with Data Guide, and Spatial.

Syntax

```sql
CREATE_INDEX (  
    specification VARCHAR2)  
RETURN NUMBER;
```

Parameters

Table 298-3 CREATE_INDEX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>specification</td>
<td>The index specification.</td>
</tr>
</tbody>
</table>

Example 298-1 Return Values

The function returns:

- 1—if the index was successfully created
- 0—if the index was not created

Exceptions

Error—if an error occurs creating the index.
298.3.1.2 DROP_INDEX Function

This function drops the named index.

Syntax

```sql
DROP_INDEX (
    index_Name IN VARCHAR2,
    force      IN BOOLEAN DEFAULT FALSE)
RETURN NUMBER;
```

Example 298-2 Parameters

Table 298-4 DROP_INDEX Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index_Name</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>force</td>
<td>The force parameter can be TRUE or FALSE. Should only be set to TRUE for dropping a JSON search index or spatial index (not B-tree index). For more information, see DROP INDEX</td>
</tr>
</tbody>
</table>

Return Values

The function returns:

- 1—if the index was successfully dropped
- 0—if the index was not dropped. For example, if there was no existing index with the specified name.

Exceptions

Error—if an error occurs while dropping the index.

298.3.1.3 FIND Function

This function returns the operation type for the collection. The operation type allows building and executing various read/write operations. This is the only way to get the reference of `SODA_Operation_T` as there is no constructor.
Syntax

FIND ()
RETURN SODA_Operation_T;

Return Values
This function returns SODA_OPERATION_T object.

Exceptions
This function does not throw any exception.

298.3.1.4 FIND_ONE Function

This function fetches the document matching the given key.

Syntax

FIND_ONE (key IN VARCHAR2)
RETURN SODA_Document_T;

Parameters

Table 298-5 FIND_ONE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>The key of the document to be fetched.</td>
</tr>
</tbody>
</table>

Return Values
This function returns the document that matches the key. Returns NULL if no match is found.

Exceptions
Error—If an error occurs while finding the document.

298.3.1.5 GET_DATA_GUIDE Function

This function fetches the JSON data guide as a CLOB. The JSON data guide is essentially inferred schema for the JSON documents in the collection. In order to be able to return the JSON data guide, a collection must have a JSON Search Index defined on it, with the data guide enabled.

Syntax

GET_DATA_GUIDE ()
RETURN CLOB;

Return Values
The function returns the JSON data guide as a CLOB.
Exceptions
Error—if an error occurs while fetching the data guide.

298.3.1.6 GET_METADATA Function
This function returns the metadata of the collection in JSON format.

Syntax
GET_METADATA ()
RETURN VARCHAR2;

Return Values
This function returns the metadata of the collection in JSON format.

298.3.1.7 GET_NAME Function
This function returns the name of the collection.

Syntax
GET_NAME ()
RETURN NVARCHAR2;

Return Values
This function returns the name of the collection.

298.3.1.8 INSERT_ONE Function
This function inserts a document into the collection.

Syntax
INSERT_ONE (document IN SODA_Document_T)
RETURN NUMBER;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>The input document.</td>
</tr>
</tbody>
</table>
Return Values
The function returns a number—1 if the doc was inserted successfully, 0 otherwise.

Exceptions
Error—if an error occurs while inserting the document into the collection.

298.3.1.9 INSERT_ONE_AND_GET Function
This function inserts a document into the collection.

Syntax
INSERT_ONE_AND_GET (  
    document      IN SODA_Document_T)  
RETURN SODA_Document_T;

Parameters

Table 298-7  INSERT_ONE_AND_GET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>The input document.</td>
</tr>
</tbody>
</table>

Return Values
The function returns the result document containing all document components supported by the given collection, with the exception of content.

Exceptions
Error—if an error occurs while inserting the document into the collection.

298.3.1.10 REMOVE_ONE Function
This function removes the document matching the given key.

Syntax
REMOVE_ONE (  
    key       IN VARCHAR2)  
RETURN NUMBER;

Parameters

Table 298-8  REMOVE_ONE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>The key of the document.</td>
</tr>
</tbody>
</table>

Return Values
This function returns the following values:
• 1—If the document was successfully removed.
• 0—If the document with the specified key was not found.

Exceptions
Error—if an error occurs while deleting the document from the collection.

298.3.1.11 REPLACE_ONE Function

This function updates the existing document with a new content and media type using the key. Any components set in `document` with the exception of content and media type are not used during the replace. They are ignored.

Syntax

```sql
REPLACE_ONE (
    key            IN VARCHAR2,
    document       IN SODA_Document_T)
RETURN NUMBER;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>The key of the document.</td>
</tr>
<tr>
<td>document</td>
<td>The document with the new content and media type to replace the old one.</td>
</tr>
</tbody>
</table>

Return Values

This function returns a number—1 if the document was replaced, 0 otherwise.

Exceptions

Error—if an error occurs while replacing the document in the collection.

298.3.1.12 REPLACE_ONE_AND_GET Function

This function updates the existing document with a new content and media type using the key. Any components set in `document` with the exception of content and media type are not used during the replace. They are ignored.

Syntax

```sql
REPLACE_ONE_AND_GET (
    key            IN VARCHAR2,
    document       IN SODA_Document_T)
RETURN SODA_Document_T;
```
Parameters

Table 298-10  REPLACE_ONE_AND_GET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>The key of the document.</td>
</tr>
<tr>
<td>document</td>
<td>The document with the new content and media type to replace the old one.</td>
</tr>
</tbody>
</table>

Return Values

The function returns the result document containing all document components supported by the given collection, with the exception of content. Last-modified and version components, if supported by the given collection, will be updated with new values. If no document in the collection had the supplied key, NULL is returned instead of the result document.

Exceptions

Error—if an error occurs while replacing the document in the collection.

298.3.2 SODA_Document_T Type

This SODA type represents a document with content, that is usually in JSON format.

This type is not persistable pl/sql type. However, SODA is a system that basically provides persistence — it has read and write operations. So you do not persist SODA_DOCUMENT_T directly, but you pass it to a write operation (like insert or replace), which is defined on SODA_COLLECTION_T, in order to write the document content and other components to the database.

A document has the following components:

• key
• content
• created-on timestamp
• last-modified timestamp
• version
• media type

Table 298-11  SODA_Document_T Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_BLOB Function</td>
<td>Fetches the BLOB content of a BLOB-based document.</td>
</tr>
<tr>
<td>GET_CLOB Function</td>
<td>Fetches the CLOB content of a CLOB-based document.</td>
</tr>
<tr>
<td>GET_CREATED_ON Function</td>
<td>Fetches the created-on timestamp in VARCHAR2.</td>
</tr>
</tbody>
</table>
Table 298-11  (Cont.) SODA_Document_T Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_DATA_TYPE Function</td>
<td>Fetches the SQL datatype of the document content with which it was created.</td>
</tr>
<tr>
<td>GET_KEY Function</td>
<td>Fetches the document key in VARCHAR2.</td>
</tr>
<tr>
<td>GET_LAST_MODIFIED Function</td>
<td>Fetches the last modified timestamp in VARCHAR2.</td>
</tr>
<tr>
<td>GET_MEDIA_TYPE Function</td>
<td>Fetches the media type of the document content in VARCHAR2.</td>
</tr>
<tr>
<td>GET_VARCHAR2 Function</td>
<td>Fetches the VARCHAR2 content of a VARCHAR2-based document.</td>
</tr>
<tr>
<td>GET_VERSION Function</td>
<td>Fetches the version of the document in VARCHAR2.</td>
</tr>
<tr>
<td>SODA_Document_T Function</td>
<td>There are three different SODA_DOCUMENT_T constructor functions. Each constructor function instantiates a document object using key, content, and media type.</td>
</tr>
</tbody>
</table>

298.3.2.1 GET_BLOB Function

This function fetches the BLOB content of the document. It assumes that the document was constructed with BLOB content, or was returned from a collection with BLOB content. Otherwise, an error is returned.

Syntax

```
GET_BLOB ()
RETURN BLOB;
```

Return Values

This function returns the BLOB content of a document.

Exceptions

SODA Error: If the document was initially not created with BLOB content.

298.3.2.2 GET_CLOB Function

The function fetches CLOB content of the document. It assumes that the document was constructed with CLOB content, or was returned from a collection with CLOB content. Otherwise, an error is returned.

Syntax

```
GET_CLOB ()
RETURN CLOB;
```

Return Values

This function returns the CLOB content of a document.
Exceptions

SODA Error: If the document was initially not created with CLOB content.

298.3.2.3 GET_CREATED_ON Function

This function fetches the created-on timestamp. The timestamp string is in ISO-8601 format, in particular this form: YYYY-MM-DDThh:mm:ss.ssssssZ format. As indicated by the Z at the end, timestamps are returned in UTC (Z indicates zero UTC offset).

Syntax

```sql
GET_CREATED_ON ()
RETURN VARCHAR2;
```

Return Values

This function returns the created-on timestamp.

298.3.2.4 GET_DATA_TYPE Function

This function fetches the SQL datatype of the document content with which it was created.

Syntax

```sql
GET_DATA_TYPE ()
RETURN PLS_INTEGER;
```

Return Values

**Table 298-12  GET_DATA_TYPE Return Values**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC_VARCHAR2 CONSTANT PLS_INTEGER</td>
<td>1</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>DOC_BLOB CONSTANT PLS_INTEGER</td>
<td>2</td>
<td>BLOB</td>
</tr>
<tr>
<td>DOC_CLOB CONSTANT PLS_INTEGER</td>
<td>3</td>
<td>CLOB</td>
</tr>
</tbody>
</table>

298.3.2.5 GET_KEY Function

This function fetches the document key.

Syntax

```sql
GET_KEY ()
RETURN VARCHAR2;
```

Return Values

This function returns the document key.
### 298.3.2.6 GET_LAST_MODIFIED Function

This function fetches the last modified timestamp. The timestamp string is in ISO-8601 format, in particular this form: `YYYY-MM-DDThh:mm:ss.sssssZ` format. As indicated by the `Z` at the end, timestamps are returned in UTC (Z indicates zero UTC offset).

**Syntax**

```sql
GET_LAST_MODIFIED ()
RETURN VARCHAR2;
```

**Return Values**

This function returns the last modified timestamp.

### 298.3.2.7 GET_MEDIA_TYPE Function

This function fetches the media type of the document content.

**Syntax**

```sql
GET_MEDIA_TYPE ()
RETURN VARCHAR2;
```

**Return Values**

This function returns the media type of the document content. `application/JSON` is the media type for JSON documents (default).

### 298.3.2.8 GET_VARCHAR2 Function

This function fetches the VARCHAR2 content of the document. It assumes that the document was constructed with VARCHAR2 content, or was returned from a collection with VARCHAR2 content. Otherwise, an error is returned.

**Syntax**

```sql
GET_VARCHAR2 ()
RETURN VARCHAR2;
```

**Return Values**

This function returns the VARCHAR2 content of a document.

**Exceptions**

SODA Error: If the document was initially not created with VARCHAR2 content.

### 298.3.2.9 GET_VERSION Function

This function fetches the version of the document.

**Syntax**

```sql
GET_VERSION ()
RETURN VARCHAR2;
```
Return Values

This function returns the version of the document.

298.3.2.10 SODA_Document_T Function

This function instantiates a document object using key, content, and media type. There are three different SODA DOCUMENT_T constructor functions. The second parameter (<v|b|c>_Content) is different in each constructor. It is VARCHAR2 in the first variant, BLOB in the second, and CLOB in the third.

Key and media type are optional parameters (will be defaulted to NULL). All three parameters can be set to NULL. If media_Type is set to NULL, it will be defaulted to application/json.

Syntax

SODA_DOCUMENT_T (
  key           IN VARCHAR2 DEFAULT NULL,
  v_Content     IN VARCHAR2,
  media_Type    IN VARCHAR2 DEFAULT NULL)
RETURN SODA_Document_T;

SODA_DOCUMENT_T (
  key           IN VARCHAR2 DEFAULT NULL,
  b_Content     IN BLOB,
  media_Type    IN VARCHAR2 DEFAULT NULL)
RETURN SODA_Document_T;

SODA_DOCUMENT_T (
  key           IN VARCHAR2 DEFAULT NULL,
  c_Content     IN CLOB,
  media_Type    IN VARCHAR2 DEFAULT NULL)
RETURN SODA_Document_T;

Parameters

Table 298-13  SODA_Document_T Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>The key of the document.</td>
</tr>
<tr>
<td>v_Content</td>
<td>The content of the document in VARCHAR2 format.</td>
</tr>
<tr>
<td>b_Content</td>
<td>The content of the document in BLOB format.</td>
</tr>
<tr>
<td>c_Content</td>
<td>The content of the document in CLOB format.</td>
</tr>
<tr>
<td>media_Type</td>
<td>The media type of the document. The media type could be application/json for JSON documents.</td>
</tr>
</tbody>
</table>
Note:

\(v\), \(b\), and \(c\) are not all parameters of a single SODA \_DOCUMENT\_T constructor function. Each one corresponds to a particular variant of the constructor function as shown in the Syntax section.

Return Values

This function returns a document of type SODA\_Document\_T.

298.3.3 SODA\_Operation\_T Type

This SODA type is used to perform read/write operations, such as document finds with filtering and pagination, removes, and replaces on a SODA collection.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT Function</td>
<td>Returns a count of the number of documents in the collection that match the criteria. If skip(...) or limit(...) were chained together with this count(), an exception is raised.</td>
</tr>
<tr>
<td>FILTER Function</td>
<td>Sets the filter (also known as QBE or query-by-example) criteria on the operation. Returns the same SODA_OPERATION_T object so that further criteria can be chained together if required.</td>
</tr>
<tr>
<td>GET_CURSOR Function</td>
<td>Returns a SODA_CURSOR_T object that can be used to iterate over the documents that match the criteria.</td>
</tr>
<tr>
<td>GET_ONE Function</td>
<td>Returns a single SODA_DOCUMENT_T object that matches the criteria. Note that, if multiple documents match the criteria, only the first document is returned.</td>
</tr>
<tr>
<td>KEY Function</td>
<td>Specifies that the document with the specified key should be returned. This causes any previous calls made to this function and KEYS(...), when they appear in the same chain, to be ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.</td>
</tr>
<tr>
<td>KEYS Function</td>
<td>Specifies that documents that match the keys supplied to this function should be returned. This causes any previous calls made to this function and KEY(...), when they appear in the same chain, to be ignored. Returns the same SODA_OPERATION_T object, so that further operation criteria can be chained together, if needed.</td>
</tr>
</tbody>
</table>
### Table 298-14  (Cont.) SODA_Operation_T Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMIT Function</td>
<td>Sets a limit on the specified number of documents the operation should return. This setting is only usable for read operations such as GET_CURSOR. For write operations, any value set using this method is ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.</td>
</tr>
<tr>
<td>REMOVE Function</td>
<td>Removes all of the documents in the collection that match the criteria. Returns the number of documents that was removed.</td>
</tr>
<tr>
<td>REPLACE_ONE Function</td>
<td>Replaces a single document in the collection with the specified document. Returns a number that indicates if the document was replaced or not. Currently, before calling this function, you must call the function KEY(...) to uniquely identify the document being replaced. Any components set in the input document with the exception of content and media type are not used during the replace. They are ignored.</td>
</tr>
<tr>
<td>REPLACE_ONE_AND_GET Function</td>
<td>Replaces a single document in the collection with the specified document. Returns a result document if the document was replaced, NULL otherwise. Currently, before calling this function, you must call the function KEY(...) to uniquely identify the document being replaced. This function is similar to REPLACE_ONE. The only difference is that REPLACE_ONE_AND_GET also returns the result document with updated components, such as version and last-modified timestamp. The result document does not contain the content component. Any components set in the input document with the exception of content and media type are not used during the replace. They are ignored.</td>
</tr>
<tr>
<td>SKIP Function</td>
<td>Sets the number of documents that match the operation criteria that will be skipped from the operation result. This setting is only usable for read operations such as GET_CURSOR. For write operations, any value set using this method is ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.</td>
</tr>
</tbody>
</table>
### Table 298-14  (Cont.) SODA_Operation_T Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION Function</td>
<td>Specifies that only documents with the supplied version should be returned. Typically, this is chained together with KEY(...) to implement optimistic locking for write operations such as REMOVE and REPLACE. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.</td>
</tr>
</tbody>
</table>

#### 298.3.3.1 COUNT Function

This function returns a count of the number of documents in the collection that match the criteria. If skip(...) or limit(...) were chained together with this count(), an exception is raised.

**Syntax**

```sql
COUNT ()
RETURN NUMBER;
```

**Return Values**

This function returns the number of documents matching the criteria specified in the operation.

**Exceptions**

*Error—If an error occurs while finding the count.*

#### 298.3.3.2 FILTER Function

Sets the filter (also known as QBE or query-by-example) criteria on the operation. Returns the same SODA_OPERATION_T object so that further criteria can be attached if needed.

**Syntax**

```sql
FILTER (qbe IN VARCHAR2)
RETURN SODA_Operation_T;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qbe</td>
<td>The string representing the query by example.</td>
</tr>
</tbody>
</table>
Return Values
This function returns the same SODA_OPERATION_T object it was invoked on.

See Also:
• Overview of SODA Filter Specifications (QBEs)
• SODA Filter Specifications (Reference)

298.3.3.3 GET_CURSOR Function

Returns a SODA_CURSOR_T object that can be used to iterate over the documents that match the criteria.

Syntax
(Optional) Enter syntax information here.
GET_CURSOR ()
RETURN SODA_Cursor_T;

Return Values
This function returns a SODA_CURSOR_T object that can be used to iterate over the documents that match the read operation criteria.

Exceptions
SODA Error—If an error occurs while fetching the cursor.

298.3.3.4 GET_ONE Function

Returns a single SODA_DOCUMENT_T object that matches the criteria. Note that, if multiple documents match the criteria, only the first document is returned.

Syntax
GET_ONE ()
RETURN SODA_Document_T;

Return Values
The first matching document.

Exceptions
Error—if an error occurs while fetching the document.

298.3.3.5 KEY Function

Specifies that the document with the specified key should be returned. This causes any previous calls made to this function and KEYS(...), when they appear in the same chain, to be
ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.

Syntax

    KEY (  
        key   IN VARCHAR2)  
    RETURN SODA_Operation_T;

Parameters

Table 298-16  KEY Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>The key to be used for the operations.</td>
</tr>
</tbody>
</table>

Return Values

This function returns the same SODA_OPERATION_T object it was invoked on.

298.3.3.6 KEYS Function

Specifies that documents that match the keys supplied to this function should be returned. This causes any previous calls made to this function and key(...), when they appear in the same chain, to be ignored. Returns the same SODA_OPERATION_T object, so that further operation criteria can be chained together, if needed.

Syntax

    KEYS (  
        key_List  IN SODA_Key_List_T)  
    RETURN SODA_Operation_T;

Parameters

Table 298-17  KEYS Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_List</td>
<td>The parameter is a SODA_Key_List_T which is a list of VARCHAR2 values representing keys.</td>
</tr>
<tr>
<td></td>
<td>Assuming key_list is a variable of type SODA_Key_List_T, it can be initialized as follows:</td>
</tr>
<tr>
<td></td>
<td>key_list := SODA_Key_List_T('key1', 'key2', 'key3', etc);</td>
</tr>
</tbody>
</table>

Return Values

This function returns the same SODA_OPERATION_T object it was invoked on.
298.3.3.7 LIMIT Function

This function sets a limit on the specified number of documents the operation should return. This setting is only usable for read operations such as GET_CURSOR. For write operations, any value set using this method is ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.

Syntax

LIMIT (limit IN NUMBER)
RETURN SODA_Operation_T;

Parameters

Table 298-18 LIMIT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>A limit on the number of results returned by read operations.</td>
</tr>
</tbody>
</table>

Return Values

This function returns the same SODA_OPERATION_T object it was invoked on.

298.3.3.8 REMOVE Function

This function removes all of the documents in the collection that match the criteria. Returns the number of documents that was removed.

Syntax

REMOVE ()
RETURN NUMBER;

Return Values

This function returns the number of matching documents that were removed in the operation.

Exceptions

Error—If an error occurs while removing the documents.

298.3.3.9 REPLACE.ONE Function

This function replaces a single document in the collection with the specified document. Returns a number that indicates if the document was replaced or not. Currently, before calling this function, you must call the function KEY(...) to uniquely identify the document being replaced. Any components set in the input document with the exception of content and media type are not used during the replace. They are ignored.
Syntax

REPLACE_ONE (  
document IN SODA_Document_T)  
RETURN NUMBER;

Parameters

Table 298-19  REPLACE_ONE Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>The document object with the new content and media type to be used for replacement.</td>
</tr>
</tbody>
</table>

Return Values

This function returns a number—1 if the document was replaced, 0 otherwise.

Exceptions

Error—if an error occurs while updating the collection.

298.3.3.10 REPLACE_ONE_AND_GET Function

Replaces a single document in the collection with the specified document. Returns a result document if the document was replaced, NULL otherwise. Currently, before calling this function, you must call the function KEY(...) to uniquely identify the document being replaced. This function is similar to REPLACE_ONE. The only difference is that REPLACE_ONE_AND_GET also returns the result document with updated components, such as version and last-modified timestamp. The result document does not contain the content component. Any components set in the input document with the exception of content and media type are not used during the replace. They are ignored.

Syntax

REPLACE_ONE_AND_GET (  
document         IN SODA_Document_T)  
RETURN SODA_Document_T;

Parameters

Table 298-20  REPLACE_ONE_AND_GET Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>The document object with the new content and media type to be used for replacement.</td>
</tr>
</tbody>
</table>

Return Values

The function returns the result document containing all document components supported by the given collection, with the exception of content. Last-modified and version components, if supported by the given collection, will be updated with new
values. If no document in the collection had the supplied key, NULL is returned instead of the result document.

Exceptions

Error—If an error occurs while updating the collection

298.3.3.11 SKIP Function

This function sets the number of documents that match the operation criteria that will be skipped from the operation result. This setting is only usable for read operations such as GET_CURSOR. For write operations, any value set using this method is ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.

Syntax

```
SKIP (  
    offset         IN NUMBER)
RETURN SODA_Operation_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>The number of documents to skip.</td>
</tr>
</tbody>
</table>

Return Values

This function returns the same SODA_OPERATION_T object it was invoked on

298.3.3.12 VERSION Function

This function specifies that only documents with the supplied version should be returned. Typically, this is chained together with KEY(...) to implement optimistic locking for write operations such as REMOVE and REPLACE. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.

Syntax

```
VERSION (  
    version         IN VARCHAR2)
RETURN SODA_Operation_T;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Document version to be used for the operation.</td>
</tr>
</tbody>
</table>
Return Values
This function returns the same SODA_OPERATION_T object it was invoked on.

298.3.4 SODA_Cursor_T Type
This SODA type is used to represent a result set of documents.

Table 298-23  SODA_Cursor_T Type Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE Function</td>
<td>Closes the cursor.</td>
</tr>
<tr>
<td>HAS_NEXT Function</td>
<td>Returns TRUE, if the next document is available for the cursor. Otherwise, returns FALSE.</td>
</tr>
<tr>
<td>NEXT Function</td>
<td>Returns the next SODA documented pointed by the cursor.</td>
</tr>
</tbody>
</table>

298.3.4.1 CLOSE Function
This function closes the cursor.

Syntax
CLOSE ()
RETURN BOOLEAN;

Example 298-3  Return Values
This function returns a boolean value.

298.3.4.2 HAS_NEXT Function
This function returns TRUE, if the next document is available for the cursor. Otherwise, returns FALSE.

Syntax
HAS_NEXT ()
RETURN BOOLEAN;

Return Values
This function returns a boolean value. TRUE, if the next document is available for the cursor. Otherwise, returns FALSE.

Exceptions
Error—If an error occurs while checking if the next document is available.

298.3.4.3 NEXT Function
This function returns the next SODA documented pointed by the cursor.
Syntax
NEXT ()
RETURN SODA_Document_T;

Return Values
This function returns the next SODA documented pointed by the cursor. Returns NULL when the HAS_NEXT function returns FALSE.

Exceptions
Error—if an error occurs while getting the next document.
UTL Streams Types describe abstract types used with Oracle XML functionality.

Four abstract PL/SQL streams are introduced and defined within the 'SYS' schema. The streams may be referenced by PUBLIC and are described in the following sections.

This chapter contains the following topics:

* Security Model
* Summary of UTL Binary Streams Types

**299.1 UTL Streams Types Security Model**

EXECUTE on UTL Streams Types is granted to PUBLIC.

**299.2 Summary of UTL Binary Streams Types**

This table lists the UTL Binary Streams Types and briefly describes them.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTL_BINARYINPUTSTREAM Type</td>
<td>Reads bytes and closes a stream.</td>
</tr>
<tr>
<td>UTL_BINARYOUTPUTSTREAM Type</td>
<td>Writes bytes and closes a stream.</td>
</tr>
<tr>
<td>UTL_CHARACTERINPUTSTREAM M Type</td>
<td>Reads chars and closes a stream.</td>
</tr>
<tr>
<td>UTL_CHARACTEROUTPUTSTREAM EAM Type</td>
<td>Writes chars and closes a stream.</td>
</tr>
</tbody>
</table>

**299.2.1 UTL_BINARYINPUTSTREAM Type**

This type is similar to java.io.InputStream in that it can only read and close a stream.

**Syntax**

CREATE OR REPLACE TYPE Utl_BinaryInputStream AS OBJECT (
MEMBER FUNCTION available (self IN OUT NOCOPY Utl_BinaryInputStream) RETURN INTEGER,
MEMBER FUNCTION read (self IN OUT NOCOPY Utl_BinaryInputStream, numBytes IN INTEGER DEFAULT 1) RETURN RAW,
MEMBER PROCEDURE read (self IN OUT NOCOPY Utl_BinaryInputStream, numBytes IN OUT INTEGER),
MEMBER PROCEDURE read (self IN OUT NOCOPY Utl_BinaryInputStream, offset IN INTEGER, numBytes IN OUT INTEGER),
MEMBER function close (self In Out Nocopy Utl_BinaryInputStream)
) NOT FINAL;

Attributes

Table 299-2 UTL_BINARYINPUTSTREAM Type Member Subprograms

<table>
<thead>
<tr>
<th>Member Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVAILABLE</td>
<td>Returns the number of bytes available to be read</td>
</tr>
<tr>
<td>READ</td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>#1 - Reads the number of bytes specified by numBytes (default is 1) and returns the bytes as a RAW. If there are no remaining bytes a value of NULL is returned.</td>
</tr>
<tr>
<td>#2</td>
<td>#2 - Reads the number of bytes specified in numBytes into the parameter bytes. Additionally, the actual number of bytes read is returned in parameter numBytes. If this parameter is set to 0 then there are no more bytes to be read.</td>
</tr>
<tr>
<td>#3</td>
<td>#3 - Reads the number of bytes specified in numBytes into the parameter bytes, beginning at the offset specified by parameter offset. The actual number of bytes read is returned in parameter numBytes. If this value is 0, then there are no additional bytes to be read.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Releases all resources held on the node to support the stream</td>
</tr>
</tbody>
</table>

299.2.2 UTL_BINARYOUTPUTSTREAM Type

This type is similar to java.io.OutputStream in that it can only write and close a stream.

Syntax

CREATE OR REPLACE TYPE Utl_BinaryOutputStream AS OBJECT {
MEMBER FUNCTION write (self IN OUT NOCOPY sys.utl_BinaryOutputStream, -- #1

Chapter 299
Summary of UTL Binary Streams Types
299.2.3 UTL_BINARYOUTPUTSTREAM Type

This type is similar to java.io.Reader in that it can only read characters (chars) and close a stream.

Syntax

```
CREATE OR REPLACE TYPE Utl_CharacterInputStream AS OBJECT {

  MEMBER FUNCTION available (self IN OUT NOCOPY utl_CharacterInputStream) RETURN INTEGER,

  MEMBER FUNCTION read (self IN OUT NOCOPY utl_CharacterInputStream, numChars IN NOCOPY INTEGER DEFAULT 1) RETURN INTEGER,

  MEMBER FUNCTION close (self IN OUT NOCOPY utl_CharacterInputStream)
}
```

Table 299-3 UTL_BINARYOUTPUTSTREAM Type Member Subprograms

<table>
<thead>
<tr>
<th>Member Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td>- #1 - Writes the number of bytes specified by numBytes (default is 1) from RAW into the stream. The actual number of bytes written is returned.</td>
</tr>
<tr>
<td></td>
<td>- #2 - Writes the number of bytes specified in parameter numBytes from parameter bytes to the stream. The actual number of bytes written is returned in parameter numBytes.</td>
</tr>
<tr>
<td></td>
<td>- #3 - Writes the number of bytes specified by numBytes to the stream, beginning at the offset specified by parameter offset. The actual number of bytes written is returned in parameter numBytes.</td>
</tr>
<tr>
<td>FLUSH</td>
<td>Insures that any buffered bytes are copied to the node destination</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Frees all resources associated with the stream</td>
</tr>
</tbody>
</table>

299.2.3 UTL_CHARACTERINPUTSTREAM Type

This type is similar to java.io.Reader in that it can only read characters (chars) and close a stream.
The UTL_CHARACTERINPUTSTREAM type has the following member subprograms:

- **AVAILABLE**: Returns the number of bytes available to be read.
- **READ**
  - #1 - Returns the number of characters remaining to be read.
  - #2 - Reads the number of characters specified by `numChars` (default value is 1) and returns the characters as a VARCHAR2. If the value of `lineFeed` is true (default value is FALSE) then the reading stops if a linefeed character is found. If there are no remaining characters a value of NULL is returned.
  - #3 - Reads reads the number of characters specified by parameter `numChars` into the parameter `chars`. Additionally, the actual number of characters read is returned in parameter `numChars`. If this value is 0, then there are no more characters to be read. If the value of `lineFeed` is TRUE (default is FALSE), then reading stops if a linefeed character is encountered.
- **CLOSE**: Releases all resources held by the stream.

---

### 299.2.4 UTL_CHARACTEROUTPUTSTREAM Type

This type is similar to java.io.Reader in that it can only read characters (chars) and close a stream.

**Syntax**

```
CREATE OR REPLACE TYPE utl_CharacterOutputStream AS OBJECT (  
    MEMBER FUNCTION write (  
        self IN OUT NOCOPY utl_CharacterOutputStream,  
        chars IN VARCHAR2,  
        numChars IN INTEGER DEFAULT 1,  
        lineFeed IN BOOLEAN DEFAULT FALSE)  
    RETURN INTEGER,  
) NOT FINAL;
```

---

**Additional Information**

- **ATTRIBUTES**
- **TABLE 299-4**
  - **UTL_CHARACTERINPUTSTREAM Type Member Subprograms**
  - **UTL_CHARACTEROUTPUTSTREAM Type**
MEMBER PROCEDURE write (                                  -- #2
    self      IN OUT NOCOPY   utl_CharacterOutputStream,
    chars     IN OUT NOCOPY   VARCHAR2,
    numChars  IN OUT          INTEGER,
    lineFeed  IN              BOOLEAN DEFAULT FALSE),

member procedure write (                                  -- #3
    self     IN OUT NOCOPY    utl_CharacterOutputStream,
    chars    IN  NOCOPY       varchar2,
    offset   IN               integer,
    numChars IN OUT           integer,
    lineFeed  IN               boolean default false),

MEMBER PROCEDURE flush (                                  -- #1
    self     IN OUT NOCOPY    utl_CharacterOutputStream),

MEMBER PROCEDURE close (                                  -- #1
    self     IN OUT NOCOPY utl_CharacterOutputStream)
)
) NOT FINAL;

Attributes

Table 299-5  UTL_CHARACTEROUTPUTSTREAM Type Member Subprograms

<table>
<thead>
<tr>
<th>Member Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>Writes the number of characters specified by numChars (default is 1) from parameter chars into the stream and returns the actual number of characters written. If the value of lineFeed is TRUE (default is FALSE) a lineFeed character is inserted after the last character.</td>
</tr>
<tr>
<td>#2</td>
<td>Writes the number of characters specified by parameter numChars, from parameter chars into the stream. The actual number of characters written is returned in parameter numChars. If the value of lineFeed is true (default is FALSE) a lineFeed character is inserted after the last character.</td>
</tr>
<tr>
<td>#3</td>
<td>Writes the number of characters specified by parameter numChars, from parameter chars, beginning at offset specified by parameter offset. The actual number of characters written is returned in parameter numChars. If the value of lineFeed is true (default is FALSE) a lineFeed character is inserted after the last character.</td>
</tr>
<tr>
<td>FLUSH</td>
<td>Copies all characters that may be contained within buffers to the node value</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Releases all resources held by the stream</td>
</tr>
</tbody>
</table>
XMLType is a system-defined opaque type for handling XML data. It as predefined member functions on it to extract XML nodes and fragments.

You can create columns of XMLType and insert XML documents into it. You can also generate XML documents as XMLType instances dynamically using the SYS_XMLAGG SQL function.

This chapter contains the following topics:

• Summary of XMLType Subprograms

See Also:

– Oracle XML DB Developer’s Guide

300.1 Summary of XMLType Subprograms

This table summarizes functions and procedures of XMLType.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATENONSCHEMABASEDXM  L</td>
<td>Creates a non schema based XML from the input schema based instance.</td>
</tr>
<tr>
<td>CREATESCHEMABASEDXML</td>
<td>Creates a schema based XMLType instance from the non-schema based instance using the input schema URL.</td>
</tr>
<tr>
<td>CREATEXML</td>
<td>Static function for creating and returning an XMLType instance.</td>
</tr>
<tr>
<td>EXISTSNODE</td>
<td>Takes a XMLType instance and a XPath and returns 1 or 0 indicating if applying the XPath returns a non-empty set of nodes.</td>
</tr>
<tr>
<td>EXTRACT</td>
<td>Takes a XMLType instance and an XPath, applies the XPath expression and returns the results as an XMLType.</td>
</tr>
<tr>
<td>GETBLOBVAL</td>
<td>Returns the value of the XMLType instance as a BLOB</td>
</tr>
<tr>
<td>GETCLOBVAL</td>
<td>Returns the value of the XMLType instance as a CLOB</td>
</tr>
<tr>
<td>GETNAMESPACE</td>
<td>Returns the namespace for the top level element in a schema based document.</td>
</tr>
<tr>
<td>GETNUMBERVAL</td>
<td>Returns the value of the XMLType instance as a NUMBER. This is only valid if the input XMLType instance contains a simple text node and is convertible to a number.</td>
</tr>
<tr>
<td>GETROOTELEMENT</td>
<td>Returns the root element of the input instance. Returns NULL if the instance is a fragment</td>
</tr>
</tbody>
</table>
Table 300-1  (Cont.) XMLTYPE Subprograms

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETSCHEMAURL</td>
<td>Returns the XML schema URL if the input is an XML Schema based.</td>
</tr>
<tr>
<td>GETSTRINGVAL</td>
<td>Returns the value of the XMLType instance as a string.</td>
</tr>
<tr>
<td>ISFRAGMENT</td>
<td>Checks if the input XMLType instance is a fragment or not. A fragment is a XML instance, which has more than one root element.</td>
</tr>
<tr>
<td>ISSCHEMABASED</td>
<td>Returns 1 or 0 indicating if the input XMLType instance is a schema based one or not.</td>
</tr>
<tr>
<td>ISSCHEMAVALID</td>
<td>Checks if the input instance is schema valid according to the given schema URL.</td>
</tr>
<tr>
<td>ISSCHEMAVALIDATED</td>
<td>Checks if the instance has been validated against the schema.</td>
</tr>
<tr>
<td>SCHEMAVALIDATE</td>
<td>Validates the input instance according to the XML Schema. Raises error if the input instance is non-schema based.</td>
</tr>
<tr>
<td>SETSCHEMAVALIDATED</td>
<td>Sets the schema valid flag to avoid costly schema validation.</td>
</tr>
<tr>
<td>TOOBJECT</td>
<td>Converts the XMLType instance to an object type.</td>
</tr>
<tr>
<td>TRANSFORM</td>
<td>Takes an XMLType instance and an associated stylesheet (which is also an XMLType instance), applies the stylesheet and returns the result as XML.</td>
</tr>
<tr>
<td>XMLTYPE</td>
<td>Constructs an instance of the XMLType datatype. The constructor can take in the XML as a CLOB, VARCHAR2 or take in a object type.</td>
</tr>
</tbody>
</table>

300.1.1 CREATENONSCHEMABASEDXML

This member function creates a non-schema based XML document from a schema based instance.

Syntax

```
MEMBER FUNCTION CREATENONSCHEMABASEDXML
return XMLType deterministic;
```

300.1.2 CREATESCHEMABASEDXML

This member function creates a schema based XMLType instance from a non-schema based XMLType value.

It uses either the supplied SCHEMA URL, or the SCHEMALOCATION attribute of the instance.

Syntax

```
MEMBER FUNCTION createSchemaBasedXML(
    schema IN varchar2 := NULL)
return XMLType deterministic;
```
### Table 300-2  CREATESCHEMABASEDXML Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Optional XMLSchema URL used to convert the value to the specified schema.</td>
</tr>
</tbody>
</table>

### 300.1.3 CREATEXML

This static function creates and returns an XMLType instance. The string and CLOB parameters used to pass in the date must contain well-formed and valid XML documents.

The options are described in the following table.

### Table 300-3  CREATEXML Subprograms

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC FUNCTION createXML(xmlData IN varchar2) RETURN XMLType deterministic;</td>
<td>Creates the XMLType instance from a string.</td>
</tr>
<tr>
<td>STATIC FUNCTION createXML(xmlData IN clob) RETURN XMLType deterministic;</td>
<td>Creates the XMLType instance from a CLOB.</td>
</tr>
<tr>
<td>STATIC FUNCTION createXML (xmlData IN clob, schema IN varchar2, validated IN number := 0, wellformed IN number := 0) RETURN XMLType deterministic;</td>
<td>This static function creates a schema-based XMLType instance using the specified schema and xml data parameters.</td>
</tr>
<tr>
<td>STATIC FUNCTION createXML (xmlData IN varchar2, schema IN varchar2, validated IN number := 0, wellformed IN number := 0) RETURN XMLType deterministic;</td>
<td>This static function creates a schema-based XMLType instance using the specified schema and xml data parameters.</td>
</tr>
<tr>
<td>STATIC FUNCTION createXML (xmlData IN &quot;&lt;ADT_1&gt;&quot;, schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN NUMBER := 0) RETURN XMLType deterministic;</td>
<td>Creates an XML instance from an instance of an user-defined type.</td>
</tr>
</tbody>
</table>
Table 300-3  (Cont.) CREATEXML Subprograms

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| STATIC FUNCTION createXML (    
  xmlData IN SYS_REFCURSOR,    
  schema in varchar2 := NULL,    
  element in varchar2 := NULL,    
  validated in number := 0)    
RETURN XMLType deterministic; | Creates an XML instance from a cursor reference. You can pass in any arbitrary SQL query as a CURSOR. |
| STATIC FUNCTION createXML (    
  xmlData IN AnyData,    
  schema in varchar2 := NULL,    
  element in varchar2 := NULL,    
  validated in number := 0)    
RETURN sys.XMLType deterministic parallel_enable | Creates an XML instance from ANYDATA. If the ANYDATA instance contains an ADT, the XMLType returned is the same as would be returned for a call directly on the ADT. If the ANYDATA contains a scalar, the XMLType contains a leaf node with the scalar value. The element name for this node is taken from the optional element string if present, and is “ANYDATA” if it is not. |
| STATIC FUNCTION createXML (    
  xmlData IN blob,    
  csid IN number,    
  schema IN varchar2,    
  validated IN number := 0,    
  wellformed IN number := 0)    
return sys.XMLType deterministic | Creates an XML instance from a BLOB. |
| STATIC FUNCTION createXML (    
  xmlData IN bfile,    
  csid IN number,    
  Schema IN varchar2,    
  validated IN number := 0,    
  wellformed IN number := 0)    
return sys.XMLType deterministic | Creates an XML instance from a BFILE. |

Table 300-4  CREATEXML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlData</td>
<td>The actual data in the form of a BFILE, BLOB, CLOB, REF cursor, VARCHAR2 or object type.</td>
</tr>
<tr>
<td>schema</td>
<td>Optional Schema URL to be used to make the input conform to the given schema. <strong>Caution:</strong> Oracle does not support use of types generated by Schema Registration (see Oracle XML DB Developer’s Guide).</td>
</tr>
<tr>
<td>validated</td>
<td>Flag to indicate that the instance is valid according to the given XML Schema. (Default is 0)</td>
</tr>
</tbody>
</table>
### Table 300-4 (Cont.) CREATEXML Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wellformed</td>
<td>Flag to indicate that the input is well formed. If set, then the database would not do well-formed check on the input instance. (Default is 0)</td>
</tr>
<tr>
<td>element</td>
<td>Optional element name in the case of the ADT_1 or REF CURSOR constructors. (Default is NULL). <strong>Caution:</strong> Oracle does not support use of types generated by Schema Registration (see Oracle XML DB Developer’s Guide).</td>
</tr>
<tr>
<td>csid</td>
<td>The character set id of input XML data.</td>
</tr>
</tbody>
</table>

### 300.1.4 EXISTSNODE

This member function checks if the node exists.

If the XPath string is NULL or the document is empty, then a value of 0 is returned, otherwise returns 1.

The options are described in the following table.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMBER FUNCTION existsNode( xpath IN varchar2) RETURN number deterministic;</td>
<td>Given an XPath expression, checks if the XPath applied over the document can return any valid nodes.</td>
</tr>
<tr>
<td>MEMBER FUNCTION existsNode( xpath in varchar2, nsmap in varchar2) RETURN number deterministic;</td>
<td>This member function uses the XPath expression with the namespace information and checks if applying the XPath returns any nodes or not.</td>
</tr>
</tbody>
</table>

### Table 300-5 EXISTSNODE Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpath</td>
<td>The XPath expression to test.</td>
</tr>
<tr>
<td>nsmap</td>
<td>Optional namespace mapping.</td>
</tr>
</tbody>
</table>

### 300.1.5 EXTRACT

This member function extracts an XMLType fragment and returns an XMLType instance containing the result nodes. If the XPath does not result in any nodes, it then returns NULL.

The options are described in the following table.
Syntax | Description
--- | ---
MEMBER FUNCTION extract(xpath IN varchar2) RETURN XMLType deterministic; | Given an XPath expression, applies the XPath to the document and returns the fragment as an XMLType.
MEMBER FUNCTION extract(xpath IN varchar2, nsmap IN varchar2) RETURN XMLType deterministic; | This member function applies the XPath expression and namespace mapping, over the XML data to return a XMLType instance containing the resultant fragment.

### Table 300-6  EXTRACT Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpath</td>
<td>The XPath expression to apply.</td>
</tr>
<tr>
<td>nsmap</td>
<td>Optional prefix to namespace mapping information.</td>
</tr>
</tbody>
</table>

#### 300.1.6 GETBLOBVAL

This member function returns a BLOB containing the serialized XML representation. If the BLOB returned is temporary, it must be freed after use.

Syntax

MEMBER FUNCTION getBlobVal(csid IN NUMBER) RETURN BLOB DETERMINISTIC;

### Table 300-7  GETBLOBVAL Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csid</td>
<td>The desired character set ID of output BLOB</td>
</tr>
</tbody>
</table>

#### 300.1.7 GETCLOBVAL

This member function returns a CLOB containing the serialized XML representation. If the CLOB returned is temporary, it must be freed after use.

Syntax

MEMBER FUNCTION getClobVal() RETURN clob deterministic;
300.1.8 GETNAMESPACE

GETNAMESPACE is a member function. It returns the namespace of the top level element in the instance. It returns NULL if the input is a fragment or is a non-schema based instance.

Syntax

MEMBER FUNCTION getNamespace
return varchar2 deterministic;

300.1.9 GETNUMBERVAL

This is a member function. It returns a numeric value, formatted from the text value pointed to by the XMLType instance. The XMLType must point to a valid text node that contains a numerical value.

The options are described in the following table.

Syntax

MEMBER FUNCTION getNumberVal()
RETURN number deterministic;

300.1.10 GETROOTELEMENT

This member function gets the root element of the XMLType instance. It returns NULL if the instance is a fragment.

Syntax

MEMBER FUNCTION getRootElement
return varchar2 deterministic;

300.1.11 GETSCHEMAURL

This member function returns the XML Schema URL corresponding to the XMLType instance, if the XMLType instance is a schema-based document. Otherwise, it returns NULL.

Syntax

MEMBER FUNCTION getSchemaURL
return varchar2 deterministic;

300.1.12 GETSTRINGVAL

This member function returns the document as a string. It returns a string containing the serialized XML representation, or in the case of text nodes, the text itself.

If the XML document exceeds the VARCHAR2 maximum size (4000), then an error is raised at run time.

Syntax

MEMBER FUNCTION getStringVal()
RETURN varchar2 deterministic;
300.1.13 ISFRAGMENT

**ISFRAGMENT** determines if the **XMLType** instance corresponds to a well-formed document, or a fragment. It returns 1 or 0 indicating if the **XMLType** instance contains a fragment or a well-formed document.

**Syntax**

MEMBER FUNCTION isFragment()
RETURN number deterministic;

300.1.14 ISSCHEMABASED

This member function determines whether the **XMLType** instance is schema-based or not. It returns 1 or 0 depending on whether the **XMLType** instance is schema-based.

**Syntax**

MEMBER FUNCTION isSchemaBased
return number deterministic;

300.1.15 ISSCHEMAVALID

This member function checks if the input instance conforms to a specified schema. It does not change the validation status of the XML instance.

If an XML Schema URL is not specified and the xml document is schema based, the conformance is checked against the **XMLType** instance’s own schema.

**Syntax**

member function isSchemaValid(
  schurl IN VARCHAR2 := NULL,
  elem IN VARCHAR2 := NULL)
return NUMBER deterministic;

<table>
<thead>
<tr>
<th>Table 300-8</th>
<th>ISSCHEMAVALID Subprogram Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>IN / OUT</td>
</tr>
<tr>
<td>schurl</td>
<td>(IN)</td>
</tr>
<tr>
<td>elem</td>
<td>(IN)</td>
</tr>
</tbody>
</table>
300.1.16 ISSCHEMAVALIDATED

This member function returns the validation status of the XMLType instance to tell if a schema-based instance has been actually validated against its schema. It returns 1 if the instance has been validated against the schema, 0 otherwise.

Syntax

MEMBER FUNCTION isSchemaValidated
return NUMBER deterministic;

300.1.17 SCHEMAVALIDATE

This member procedure validates the XML instance against its schema, if it has not already been done.

For non-schema based documents an error is raised. If validation fails an error is raised; else, the document’s status is changed to validated.

Syntax

MEMBER PROCEDURE schemaValidate(
    self IF OUT NOCOPY XMLType);

Table 300-9 SCHEMAVALIDATE Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>(OUT)</td>
<td>XML instance being validated against the schema.</td>
</tr>
</tbody>
</table>

300.1.18 SETSCHEMAVALIDATED

This member function sets the VALIDATION state of the input XML instance.

Syntax

MEMBER PROCEDURE setSchemaValidated(
    self IF OUT NOCOPY XMLType,
    flag IN BINARY_INTEGER := 1);

Table 300-10 SETSCHEMAVALIDATED Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>(OUT)</td>
<td>XML instance.</td>
</tr>
<tr>
<td>flag</td>
<td>(IN)</td>
<td>0 - NOT VALIDATED; 1 - VALIDATED (Default)</td>
</tr>
</tbody>
</table>
300.1.19 TOOBJECT

This member procedure converts the XML value to an object type using the XMLSCHEMA mapping, if available. If a SCHEMA is not supplied or the input is a non-schema based XML, the procedure uses cannonical mapping between elements and object type attributes.

Syntax

MEMBER PROCEDURE toObject(
SELF in XMLType,
object OUT "<ADT_1>",
schema in varchar2 := NULL,
element in varchar2 := NULL);

Table 300-11  TOOBJECT Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELF</td>
<td>(IN)</td>
<td>Instance to be converted. Implicit if used as a member procedure.</td>
</tr>
<tr>
<td>object</td>
<td>(IN)</td>
<td>Converted object. An object instance of the required type may be passed in to this function</td>
</tr>
<tr>
<td>schema</td>
<td>(IN)</td>
<td>Schema URL. The mapping of the XMLType instance to the converted object instance may be specified using a schema. Caution: Oracle does not support use of types generated by Schema Registration (see Oracle XML DB Developer's Guide).</td>
</tr>
<tr>
<td>element</td>
<td>(IN)</td>
<td>Top-level element name. An XML Schema document does not specify the top-level element for a conforming XML instance document without this parameter. Caution: Oracle does not support use of types generated by Schema Registration (see Oracle XML DB Developer's Guide).</td>
</tr>
</tbody>
</table>

300.1.20 TRANSFORM

This member function transforms the XML data using the XSL stylesheet argument and the top-level parameters passed as a string of name=value pairs

If any of the arguments other than the parammap is NULL, then a NULL is returned.

Syntax

MEMBER FUNCTION transform(
    xsl IN XMLType,
    parammap in varchar2 := NULL)
    RETURN XMLType deterministic;

Table 300-12  TRANSFORM Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN / OUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsl</td>
<td>(IN)</td>
<td>The XSL stylesheet describing the transformation</td>
</tr>
<tr>
<td>parammap</td>
<td>(IN)</td>
<td>Top level parameters to the XSL - string of name=value pairs</td>
</tr>
</tbody>
</table>
300.1.21 XMLTYPE

This is an XMLType constructor.

The options are described in the following table.

Table 300-13  XMLTYPE Member Subprogram Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor function XMLType( xmlData IN clob, schema IN varchar2 := NULL, validated IN number := 0, wellformed IN number := 0) return self as result deterministic;</td>
<td>This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data parameters.</td>
</tr>
<tr>
<td>constructor function XMLType( xmlData IN varchar2, schema IN varchar2 := NULL, validated IN number := 0, wellformed IN number := 0) return self as result deterministic;</td>
<td>This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data parameters.</td>
</tr>
<tr>
<td>constructor function XMLType ( xmlData IN &quot;w&lt;ADT_1&gt;&quot;, schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN number := 0) return self as result deterministic;</td>
<td>This constructor function creates an optionally schema-based XMLType instance from the specified object type parameter.</td>
</tr>
<tr>
<td>constructor function XMLType( xmlData IN SYS_REFCURSOR, schema in varchar2 := NULL, element in varchar2 := NULL, validated in number := 0) return self as result deterministic;</td>
<td>This constructor function creates an optionally schema-based XMLType instance from the specified REF CURSOR parameter.</td>
</tr>
<tr>
<td>constructor function XMLType( xmlData IN AnyData, schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN number := 0) return self as result deterministic parallel_enable</td>
<td>This constructor function creates an optionally schema-based XMLType instance from the specified ANYDATA parameter. If the ANYDATA instance contains an ADT, the XMLType returned is the same as would be returned for a call directly on the ADT. If the ANYDATA contains a scalar, the XMLType contains a leaf node with the scalar value. The element name for this node is taken from the optional element string if present, and is &quot;ANYDATA&quot; if it is not.</td>
</tr>
</tbody>
</table>
Table 300-13  (Cont.) XMLTYPE Member Subprogram Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| constructor function XMLType(  
  xmlData IN blob, csid IN number,  
  schema IN varchar2 := NULL,  
  validated IN number := 0,  
  wellformed IN number := 0)  
return self as result deterministic | This constructor function creates an optionally schema-based XMLType instance from the specified BLOB parameter. |
| constructor function XMLType(  
  xmlData IN bfile,  
  csid IN number,  
  schema IN varchar2 := NULL,  
  validated IN number := 0,  
  wellformed IN number := 0)  
return self as result deterministic | This constructor function creates an optionally schema-based XMLType instance from the specified BFILE parameter. |

Table 300-14  XMLTYPE Constructor Subprogram Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlData</td>
<td>The data in the form of a BFILE, BLOB, CLOB, REFS, VARCHAR2 or object type.</td>
</tr>
<tr>
<td>schema</td>
<td>Optional Schema URL to be used to make the input conform to the given schema. <strong>Caution:</strong> Oracle does not support use of types generated by Schema Registration (see Oracle XML DB Developer’s Guide).</td>
</tr>
<tr>
<td>validated</td>
<td>Indicates that the instance is valid to the given XML Schema.</td>
</tr>
<tr>
<td>wellformed</td>
<td>Indicates that the input is well formed. If set, then the database would not do well formed check on the input instance.</td>
</tr>
<tr>
<td>element</td>
<td>Optional element name in the case of the ADT_1 or REF CURSOR constructors. (Default is NULL). <strong>Caution:</strong> Oracle does not support use of types generated by Schema Registration (see Oracle XML DB Developer’s Guide).</td>
</tr>
<tr>
<td>csid</td>
<td>The character set id of input XML data.</td>
</tr>
</tbody>
</table>
Index

A
ABORT procedure, 137-2
ABORT_GLOBAL_INSTANTIATION procedure, 32-3
ABORT_REDEF_TABLE procedure, 130-5
ABORT_ROLLBACK Procedure, 130-6
ABORT_SCHEMA_INSTANTIATION procedure, 32-4
ABORT_TABLE_INSTANTIATION procedure, 32-6
ABORT_UPDATE Procedure, 130-6
ABORTED_REQUEST_THRESHOLD procedure, 149-2
ACCEPT_SQL_PATCH Procedure, 157-8
ACCEPT_SQL_PLAN_BASELINE Procedure, 153-4
ACCEPT_SQL_PROFILE procedure, 160-15
ACLCHECKPRIVILEGES Function, 194-4
ACTIVE_INSTANCES Procedure, 183-6
ADD procedure, 131-2
ADD_AUTO_CDR procedure, 72-3
ADD_AUTO_CDR_COLUMN_GROUP procedure, 72-4
ADD_AUTO_CDR_DELTA_RES procedure, 72-5
ADD_CAPTURE Procedure, 187-4
ADD_COLORED_SQL Procedure, 188-5
ADD_COLUMN member procedure, 289-11
ADD_COLUMN procedure, 165-24, 212-5
ADD_COOKIES procedure, 265-29
ADD_FILE procedure, 66-4
ADD_FILTER Procedure, 186-3, 187-5
ADD_GLOBAL_PROPAGATION_RULES procedure, 165-27, 212-8
ADD_GLOBAL_RULES procedure, 165-32, 212-12
ADD_GROUPED_POLICY Procedure, 138-4
ADD_MESSAGES_PROPAGATION_RULE procedure, 165-37
ADD_MESSAGE_RULE procedure, 165-40
ADD_OUTBOUND procedure, 212-17
ADD_PAIR member procedure, 297-5
ADD_POLICY Procedure, 65-2, 129-7, 138-6, 179-3
ADD_POLICY_CONTEXT Procedure, 138-10
ADD_PRIVILEGE Procedure, 108-6
ADD_REFERENCE function, 159-6
ADD_RULE procedure, 142-2
ADD_SCHEDULE_ORDERING Function, 187-6
ADD_SCHEMA_PROPAGATION_RULES procedure, 165-43, 212-21
ADD_SCHEMA_RULES procedure, 165-48, 212-26
ADD_SCHEMA procedure, 102-4
ADD_SENSITIVE_COLUMN Procedure, 178-2
ADD_SENSITIVE_TYPE Procedure, 178-4
ADD_SQL_CONNECTION_TEST Procedure, 18-2, 146-2
ADD_SQLSET_REFERENCE function, 160-20
ADD_SQLWKLD_REF Procedure, 16-5
ADD_SQLWKLD_STATEMENT Procedure, 16-6
ADD_STMT_HANDLER procedure, 20-5
ADD_STMT_TO_HANDLER procedure, 168-7
ADD_SUBSCRIBER Procedure, 103-12
ADD_SUBSET_OUTBOUND_RULES procedure, 212-31
ADD_SUBSET_PROPAGATION_RULES procedure, 165-53, 212-33
ADD_SUBSET_RULES procedure, 165-57, 212-37
ADD_TABLE_PROPAGATION_RULES procedure, 165-63, 212-42
ADD_TABLE_RULES procedure, 165-67, 212-46
ADD_TABLE_TO_APPLICATION Procedure, 69-4
ADD_TO_ILM Procedure, 82-3
ADD_VIRTUAL_COLUMNS procedure, 89-2
ADD_WARNING_SETTING_CAT procedure, 184-2
ADD_WARNING_SETTING_NUM procedure, 184-2
ADD2MULTI procedure, 230-2
ADDATTR member procedure of ANYTYPE TYPE, 283-3
ADDDHTTPEXPIREMAPPING Procedure, 192-4
ADDINSTANCE member procedure of ANYDATASET TYPE, 282-2
ADDMIMEMAPPING Procedure, 192-4
ADDREPOSITORYRESCONFIG Procedure, 133-2
ADDRESCONFIG Function, 133-2
ADDRESS function
of HTF package, 215-12
ADDRESS procedure
of HTP package, 216-11
ADDSCEMALLOCMAPPING Procedure, 192-5
ADDSEVRLET Procedure, 192-5
ADDSEVRLETMAPPING Procedure, 192-6
ADDSEVRLETSECROLE Procedure, 192-7
ADDXDBNAMESPACE, 207-3
ADDXMLEXTENSION Procedure, 192-7
ADM_DROP_CHUNKS Procedure, 113-7
ADM_DROP_TASK Procedure, 113-7
ADM_GET_TASK_STATUS Procedure, 113-8
ADM_STOP_TASK Procedure, 113-8
ADM_TABLES procedure, 132-5
ADVISE_COMMIT procedure, 176-2
ADVISE_NOTHING procedure, 176-2
ADVISE_ROLLBACK procedure, 176-3
ADVISOR privilege, 40-2
advisors
Oracle Streams, 166-1
aggregation management, 40-2
aggregation operators (OLAP), 40-19
ALLOCATE_UNIQUE procedure, 94-3
ALLOCATE_UNIQUE_AUTONOMOUS procedure, 94-5
ALTER_AGENT Procedure, 103-15
ALTER_APPLY procedure, 20-7
ALTER_AUTO_CDR procedure, 72-6
ALTER_AUTO_CDR_COLUMN_GROUP procedure, 72-7
ALTER_CAPTURE procedure, 32-6
ALTER_COMPILE procedure, 52-2
ALTER_EVALUATION_CONTEXT procedure, 142-4
ALTER_FILE procedure, 66-6
ALTER_FILE_GROUP procedure, 66-8
ALTER_GROUPED_POLICY Procedure, 138-12
ALTER_INBOUND procedure, 212-52
ALTER_JOB Procedure, 103-17
ALTER_MsgSYSTEM_LINK Procedure for TIB/Rendezvous, 103-19
ALTER_MsgSYSTEM_LINK Procedure for WebSphere MQ, 103-20
ALTER_OUTBOUND procedure, 212-53
ALTER_PARAM Procedure, 35-1
ALTER_PARTITION_INTERVAL procedure, 26-8
ALTER_POLICY Procedure, 129-13, 138-11, 179-4
ALTER_PROPAGATION procedure, 126-2
ALTER_PROPAGATION_SCHEDULE Procedure, 103-21
ALTER_REWRITE_EQUIVALENCE Procedure, 15-2
ALTER_RULE procedure, 142-6
ALTER_SENSITIVE_COLUMN Procedure, 178-3
ALTER_SQL_PATCH Procedure, 157-9
ALTER_SQL_PLAN_BASELINE Function, 153-5
ALTER_SQL_PLAN_DIRECTIVE Procedure, 152-2
ALTER_SQL_PROFILE procedure, 160-21
ALTER_STATS_HISTORY_RETENTION procedure, 162-18
ALTER_SUBSCRIBER Procedure, 103-22
ALTER_SYNC_CAPTURE procedure, 32-11
ALTER_TABLE_NOT_REFERENCEABLE procedure, 52-3
ALTER_TABLE_REFERENCEABLE Procedure, 52-4
ALTER_VERSION procedure, 66-10
AMATCH function, 228-3
analytic views, 77-1
hierarchy validation, 77-6
validation, 77-3
validation log table, 77-2
verify validation, 77-5
ANALYZE Function, 79-1
ANALYZE Procedure, 31-1
ANALYZE_CURRENT_PERFORMANCE procedure, 166-5
ANALYZE_DATABASE Procedure, 183-7
ANALYZE_DB Procedure, 14-2
ANALYZE_INST Procedure, 14-3
ANALYZE_PART_OBJECT Procedure, 183-7
ANALYZE_PARTIAL Procedure, 14-4
ANALYZE_SCHEMA Procedure, 183-8
ANCHOR function
of HTF package, 215-12
ANCHOR procedure
of HTP package, 216-12
ANCHOR2 function
of HTF package, 215-13
ANCHOR2 procedure
of HTP package, 216-13
anonymous PL/SQL blocks
dynamic SQL and, 154-2
AnyData datatype
queues
creating, 165-167, 212-98
removing, 165-147, 212-83
APEX packages, 2-1
APPEND Procedures, 93-14
APPEND_HOST_ACE Procedure, 108-7
APPEND_HOST_ACL Procedure, 108-8
APPEND_WALLET_ACE Procedure, 108-9
APPEND_WALLET_ACL Procedure, 108-10
APPENDCHILD function, 200-22
APPENDDATA procedure, 200-22
APPENDRESCONFIG Procedure, 133-3
APPENDRESOURCENETADATA Procedure, 194-4
APPLETCLOSE function
of HTF package, 215-14
APPLETCLOSE procedure
of HTP package, 216-13
APPLETOPEN function
of HTF package, 215-14
APPLETOPEN procedure
of HTP package, 216-14
apply process
altering, 20-7
apply user, 165-2
change handlers
setting, 20-33
conflict handlers
setting, 20-86
creating, 20-14, 165-32, 165-40, 165-48,
165-57, 165-67, 212-12, 212-26,
212-37, 212-46
DBMS_APPLY_ADM package, 20-1
DDL handler
setting, 20-7, 20-14
dropping, 20-23
enqueuing events, 20-46
error handlers
setting, 20-41
error queue
deleting errors, 20-22
executing errors, 20-26, 20-27
getting error messages, 20-29
instantiation
global SCN, 20-49
schema SCN, 20-80
table SCN, 20-83
message handler
setting, 20-7, 20-14
parameters
setting, 20-55
precommit handler
setting, 20-7, 20-14
procedure DML handlers
setting, 20-41
rules
defining global, 165-32, 212-12
defining message, 165-40
defining schema, 165-48, 212-26
defining subset, 165-57, 212-37
defining table, 165-67, 212-46
for LCRs, 165-7
for user messages, 165-7
removing, 165-148, 212-84
specifying execution, 20-48
starting, 20-90
apply process (continued)
statement DML handlers
setting, 20-5, 20-33
stopping, 20-91
substitute key columns
setting, 20-52
apply user
DBMS_STREAMS_ADM package, 165-2
APPLY_POLICY_EXPR_TO_COL Procedure, 129-19
APPLY$_ENQUEUE, 20-46
APPLY$_EXECUTE, 20-48
AQ$AGENT Type, 284-2
AQ$AGENT_LIST_T Type, 284-2
AQ$DESCRIPTOR Type, 284-3
AQ$NTFN_MSGID_ARRAY Type, 284-4
AQ$POST_INFO Type, 284-4
AQ$POST_INFO_LIST Type, 284-5
AQ$PURGE_OPTIONS_T Type, 284-5
AQ$RECIPIENT_LIST_T Type, 284-6
AQ$REG_INFO Type, 284-6
AQ$REG_INFO_LIST Type, 284-9
AQ$SUBSCRIBER_LIST_T Type, 284-9
ARCHIVESTATENAME Function, 82-4
AREA function
of HTF package, 215-15
AREA procedure
of HTP package, 216-15
arrays
BIND_ARRAY procedure, 154-7
bulk DML using DBMS_SQL, 154-27
ASA_RECO_ROW Record Type, 150-2
ASA_RECO_ROW_TB Table Type, 150-3
ASA_RECOMMENDATIONS Function, 150-4
ASH_GLOBAL_REPORT_HTML Function, 188-5
ASH_GLOBAL_REPORT_TEXT Function, 188-7
ASH_REPORT_ANALYTICS function, 188-10
ASH_REPORT_HTML Function, 188-11
ASH_REPORT_TEXT Function, 188-13
ASSIGN_ACL Procedure, 108-11
ASSIGN_GROUP_TO_INSTANCE Procedure, 187-7
ASSIGN_WALLET_ACL Procedure, 108-12
ASSM_SEGMENT_VERIFY Procedure, 151-4
ASSM_TABLESPACE_VERIFY Procedure, 151-6
ASSOCIATE_POLICY Procedure, 179-5
ATTACH_SESSION procedure, 53-16
ATTACH_SIMPLE_TABLESPACE procedure, 170-4
ATTACH_TABLESPACES procedure, 170-6
AUTH Function and Procedure, 276-8
AUTHORIZE function, 225-1
AUTHORIZE_DAD Procedure, 63-4
AVAILABLE function, 278-5
AWR_BASELINE_METRIC_TYPE Object Type, 188-2
AWR_BASELINE_METRIC_TYPE_TABLE Table Type, 188-3
AWR_DIFF_REPORT_HTML Function, 188-16
AWR_DIFF_REPORT_TEXT Function, 188-16
AWR_GLOBAL_DIFF_REPORT_HTML Functions, 188-17
AWR_GLOBAL_DIFF_REPORT_TEXT Functions, 188-18
AWR_GLOBAL_REPORT_HTML Functions, 188-19
AWR_GLOBAL_REPORT_TEXT Functions, 188-20
AWR_REPORT_HTML function, 188-21
AWR_REPORT_TEXT function, 188-22
AWR_SET_REPORT_THRESHOLDS Procedure, 188-22
AWR_SQL_REPORT_HTML Function, 188-23
AWR_SQL_REPORT_TEXT Function, 188-24

B

BACKTRACE_DEPTH Function, 260-4
BACKTRACE_LINE Function, 260-5
BACKTRACE_UNIT Function, 260-5
BASE function
 of HTF package, 215-15
BASE procedure
 of HTF package, 216-15
BASE64_DECODER function, 263-2
BASE64_ENCODER function, 263-2
BASE_FONT function
 of HTF package, 215-16
BASE_FONT procedure
 of HTF package, 216-16
BEGIN_OPERATION Function, 155-3
BEGIN_PREPARE Procedure, 61-3
BEGIN_REPLAY_SCHEDULE Procedure, 187-7
BEGIN_REQUEST function, 265-30
BEGIN_SQL_BLOCK Procedure, 134-5
BEGIN_UPGRADE Procedure, 61-3
BEGIN_CREATE static procedure
 of ANYDATA TYPE, 281-3
 of ANYDATASET TYPE, 282-3
 of ANYTYPE TYPE, 283-1
BFILE_TABLE Table Type, 154-20
BGSOUND function
 of HTF package, 215-16
BGSOUND procedure
 of HTF package, 216-16
BIG function
 of HTF package, 215-17
BIG procedure
 of HTF package, 216-17

BINARY_DOUBLE_TABLE Table Type, 154-21
BINARY_FLOAT_TABLE Table Type, 154-21
BIND_ARRAY procedures, 154-26
BIND_INOUT_VARIABLE Procedure, 81-2
BIND_INOUT_VARIABLE_RAW Procedure, 81-3
BIND_OUT_VARIABLE Procedure, 81-4
BIND_OUT_VARIABLE_RAW Procedure, 81-5
BIND_VARIABLE Procedure, 81-6, 154-7
BIND_VARIABLE procedures, 154-31
BIND_VARIABLE_PKG Procedure, 154-7, 154-33
BIND_VARIABLE_RAW Procedure, 81-7
BIND_VARIABLES function, 231-3
BIT_AND function, 273-3
BIT_COMPLEMENT function, 273-4
BIT_OR function, 273-4
BIT_XOR function, 273-5
BLACK_LIST Function, 136-2
BLACK_LIST_ADD Procedure, 136-3
BLACK_LIST_REMOVE, 136-3
BLAS Level 1 (Vector-Vector Operations) Subprograms, 272-2
BLAS Level 2 (Matrix-Vector Operations) Subprograms, 272-3
BLAS Level 3 (Matrix-Matrix Operations) Subprograms, 272-4
BLAS_ASUM Functions, 272-14
BLAS_AXPY Procedures, 272-14
BLAS_COPY Procedures, 272-15
BLAS_DOT Functions, 272-16
BLAS_GBMV Procedures, 272-17
BLAS_GEMM Procedures, 272-19
BLAS_GEMV Procedures, 272-21
BLAS_GER Procedures, 272-23
BLAS_GER2 Procedures, 272-27
BLAS_IAMAX Functions, 272-25
BLAS_NRM2 Functions, 272-25
BLAS_NRM2H Functions, 272-25
BLAS_ROT Procedures, 272-26
BLAS_ROTG Procedures, 272-27
BLAS_SBMV Procedures, 272-34
BLAS_SBMVH Procedures, 272-34
BLAS_SBMVH Functions, 272-34
BLAS_SPMV Procedures, 272-29
BLAS_SPMVH Procedures, 272-29
BLAS_SPR Procedures, 272-30
BLAS_SPR2 Procedures, 272-32
BLAS_SWAP Procedure, 272-35
BLAS_SYMM Procedures, 272-36
BLAS_SYMV Procedures, 272-39
BLAS_SYR Procedures, 272-40
BLAS_SYR2 Procedures, 272-42
BLAS_SYR2H Procedures, 272-42
BLAS_SYRK Procedures, 272-46
BLAS_TBMV Procedures, 272-48
BLAS_TBMVH Procedures, 272-50
BLAS_TPSV Procedures, 272-52
BLAS_TRMM Procedures, 272-55
BLAS_TRMV Procedures, 272-57
capture process (continued)
capture user, 165-2
creating, 165-32, 165-48, 165-57, 165-67, 212-12, 212-26, 212-37, 212-46
DBMS_CAPTURE_ADM package, 32-1 instantiation
aborting database preparation, 32-3
aborting schema preparation, 32-4
aborting table preparation, 32-6
preparing a database for, 32-9
preparing a schema for, 32-30
preparing a table for, 32-33
parameters
setting, 32-34
rules, 165-7
defining global, 165-32, 212-12
defining schema, 165-48, 212-26
defining subset, 165-57, 212-37
defining table, 165-67, 212-46
removing, 165-148, 212-84
starting, 32-51
stopping, 32-52
CAPTURE_CURSOR_CACHE procedure, 159-7
CAPTURE_CURSOR_CACHE_SQLSET Procedure, 160-23
CAPTURE_DEPENDENCY_PRIVS Procedure, 123-3
CAST_FROM_BINARY_DOUBLE function, 273-6
CAST_FROM_BINARY_FLOAT function, 273-7
CAST_FROM_BINARY_INTEGER function, 273-8
CAST_FROM_NUMBER function, 273-9
CAST_TO_BINARY_DOUBLE function, 273-9
CAST_TO_BINARY_FLOAT function, 273-10
CAST_TO_BINARY_INTEGER function, 273-11
CAST_TO_NUMBER function, 273-12
CAST_TO_NVARCHAR2 function, 273-12
CAST_TO_RAW function, 273-13
CAST_TO_VARCHAR2 function, 273-14
CB$ table prefix, 40-3
CELLSPRINT procedures, 231-5
CENTER function of HTF package, 215-20
CENTER procedure of HTP package, 216-20
CENTERCLOSE function of HTF package, 215-20
CENTERCLOSE procedure of HTP package, 216-21
CENTEROPEN function of HTF package, 215-21
CENTEROPEN procedure of HTP package, 216-21
CFG_GET Function, 192-8
CFG_REFRESH Procedure, 192-8
CFG_UPDATE Procedure, 192-8
change data capture
   Oracle Streams
      configuring, 165-85
CHANGE functions and procedures, 228-5
CHANGE procedure, 88-4, 131-3
CHANGEOWNER Procedure, 194-5
CHANGEPRIVILEGES Function, 194-5
CHARARR Table Type, 112-5
CHECK_OBJECT procedure, 132-6
CHECK_PATCH_INSTALLED Function, 127-3
CHECK_PLUG_COMPATIBILITY Function, 116-3
CHECK_PRIVILEGE Function, 108-13
CHECK_PRIVILEGE_ACLID Function, 108-14
CHECKACCESS Function, 48-26, 49-4
CHECKIN function, 195-2
CHECKOUT procedure, 195-2
CHECKPRIVILEGES Function, 194-6
CHECKSPI Functions and Procedures, 48-26
CHECKSUM functions, 166-2
CHOOSE_DATE procedure, 231-7
CITE function
   of HTF package, 215-22
CITE procedure
   of HTP package, 216-22
CLEAN_AUDIT_TRAIL procedure, 26-9
CLEANUP_GATEWAY Procedure, 103-24
CLEANUP_GIDX Procedure, 114-2
CLEANUP_GIDX_JOB Procedure, 114-2
CLEANUP_INSTANTIATION_SETUP procedure, 165-74
CLEANUP_ONLINE_OP Procedure, 114-3
CLEANUPUNUSEDBACKUPFILES Procedure, 50-7
CLEAR Procedure, 199-8
CLEAR_ALL_CONTEXT Procedure, 146-3
CLEAR_AUDIT_TRAIL_PROPERTY procedure, 26-11
CLEAR_CONTEXT Procedure, 146-4
CLEAR.Cookies procedure, 265-32
CLEAR.HEAT_MAP.ALL Procedure, 83-5
CLEAR.HEAT_MAP_TABLE Procedure, 83-5
CLEAR_IDENTIFIER Procedure, 146-4
CLEAR_LAST_ARCHIVE_TIMESTAMP procedure, 26-12
CLEAR_PENDING_AREA procedure, 134-8
CLEAR_PLSQL_TRACE procedure, 175-6
CLEAR_USED procedure, 111-2
CLEARKEYCOLUMNLIST procedure, 205-3, 209-2
CLEARUPDATACOLUMNLIST procedure, 205-3, 209-2
CLIENT_ID_STAT_DISABLE procedure, 105-2
CLIENT_ID_STAT_ENABLE procedure, 105-2
CLIENT_ID_TRACING_DISABLE procedure, 105-3
CLIENT_ID_TRACING_ENABLE procedure, 105-3
CLOB_TABLE Table Type, 154-21
CLOB2FILE procedure, 93-16, 211-3
CLONE_SIMPLE_TABLESPACE procedure, 170-10
CLONE_TABLESPACES procedure, 170-12
CLONEDB_RENAMEFILE Procedure, 60-1
CLONENODE function, 200-23
CLOSE Procedure, 93-16
CLOSE_ALL_CONNECTIONS procedure, 278-7
CLOSE_CONNECTION procedure, 278-7
CLOSE_CONNECTION Procedure, 276-10
CLOSE_CURSOR procedure, 154-35
CLOSE_CURSOR Procedure, 81-8
CLOSE_DATA Function and Procedure, 276-10
CLOSE_DATABASE_LINK Procedure, 146-5
CLOSE_ITERATOR procedure, 141-2
CLOSE_PERSISTENT_CONN procedure, 265-32
CLOSE_PERSISTENT_CONNS procedure, 265-33
CLOSECONTEXT procedure, 201-2, 204-4, 205-4, 209-3
CODE function
   of HTF package, 215-22
CODE procedure
   of HTP package, 216-22
COLLECT_STATS procedure, 277-13
collections
table items, 154-26
COLUMN_T Type, 173-24
COLUMN_VALUE procedure, 154-35
COLUMN_VALUE_LONG procedure, 154-38
COLUMNS_NEW_T Type, 173-27
COLUMNS_T Type, 173-27
COLUMNS_WITH_TYPE_T Type, 173-27
COMMA_TO_TABLE Procedures, 183-10
COMMAND function and procedure, 276-11
COMMAND_REPLIES function, 276-12
COMMENT function
   of HTF package, 215-23
COMMENT procedure
   of HTP package, 216-23
COMMIT procedure, 176-3
COMMIT_ACL procedure, 147-3
COMMIT_COMMENT procedure, 176-3
COMMIT_FORCE procedure, 176-3
COMPARE function, 273-15
COMPARE Function, 33-11
COMPARE Functions, 93-17
COMPARE_CAPTURE_REPLAY_REPORT Function, 14-5
COMPARE_DATABASES Function, 14-6
COMPARE_INSTANCES Function, 14-7
COMPARE_OLD_VALUES procedure, 20-12
COMPARE_PERIOD_REPORT Procedure, 187-9
COMPARE_REPLAY_REPLAY_REPORT Function, 14-8
COMPARE_SQLSET_REPORT Function, 187-10
COMPARISON_TYPE type, 33-10
COMPATIBLE_10_1 function, 164-6, 289-35
COMPATIBLE_10_2 function, 164-5, 289-35
COMPATIBLE_11_1 function, 164-4
COMPATIBLE_11_2 function, 164-4
COMPATIBLE_12_1 function, 164-3
COMPATIBLE_12_2 function, 164-3
COMPATIBLE_9_2 function, 164-6, 289-35
COMPILABLE_FROM_REMOTE procedure, 92-2
COMPILABLE_SCHEMA Procedure, 183-11
COMPILABLESCHEMA procedure, 206-5
CONCAT function, 273-15
CONCATENATE_SUBPROGRAM Function, 260-6
CONFIGURE Procedure, 153-6
CONFIGURE_ADVISOR_FILTER function, 162-20
CONFIGURE_ADVISOR_OBJ_FILTER function, 162-26
CONFIGURE_ADVISOR_OPR_FILTER function, 162-27
CONFIGURE_ADVISOR_RULE_FILTER function, 162-30
CONFIGURE_NODE procedure, 182-2
CONFIGURE_POOL Procedure, 35-2
CONFIGURE_AUTOSYNC procedure, 197-4
conflicts detection stopping, 20-12
CONNECT_TCP Procedure, 54-2
constants
DBMS_HIERARCHY package, 77-1
DBMS_ILM_ADMIN package, 83-1
DBMS_MGWMSG package, 104-1
CONTAINS_HOST Function, 109-3
CONTINUE function, 53-16
CONTROL_REstricted_SNAPSHOT procedure, 188-25
CONVERGE Procedure, 33-13
CONVERT function, 94-6, 201-2, 273-16
CONVERT_ANYDATA_TO_LCR_DDL function, 164-7
CONVERT_ANYDATA_TO_LCR_ROW function, 164-7
CONVERT_LCR_TO_XML function, 164-8
CONVERT_LONG_TO_LOB_CHUNK member procedure, 289-11
CONVERT_RAW_VALUE procedures, 162-31
CONVERT_RAW_VALUE_NVARCHAR procedure, 162-32
CONVERT_RAW_VALUE_ROWID procedure, 162-32
CONVERT_XML_TO_LCR function, 164-9
CONVERTTOBLOB procedure, 93-19
CONVERTTOCLOB Procedure, 93-22
COPY_FILE procedure, 67-2
COPY_FROM_DBFS_LINK Procedures, 93-26
CREATe ANY DIMENSION privilege, 40-2
CREATe ANALYSIS_TASK Functions, 158-2
CREATe APPLY procedure, 20-14
CREATe ANALYSIS_TASK Function and Procedure, 188-26
CREATe_BASELINE Function and Procedure, 188-27
CREATe_BASELINE_TEMPLATE Procedures, 44-18
CREATe_BIN_CAT procedure, 44-18
CREATe_BIN_NUM procedure, 44-19
CREATe_CAPTURE procedure capturing, 32-13
CREATE_CAPTURE Procedure, 123-4
CREATE_CATEGORY function, 102-8
CREATE_CATEGORY Procedure, 134-8
CREATE_CDB_PLAN Procedure, 134-8
CREATE_CDB_PLAN_DIRECTIVE Procedure, 134-9
CREATE_CDB_PROFILE_DIRECTIVE Procedure, 134-10
CREATE_CHUNKS_BY_NUMBER_COL Procedure, 113-9
CREATE_CHUNKS_BY_ROWID Procedure, 113-9
CREATE_CHUNKS_BY_SQL Procedure, 113-10
CREATE_CLIP procedure, 44-21
CREATE_COL_REM procedure, 44-23
CREATE_COMPARISON Procedure, 33-14
CREATE_CONSUMER_GROUP procedure, 134-11
CREATE_COVERAGE_TABLES Procedure, 120-2
CREATE_CREDENTIAL Procedure, 37-3
CREATE_DAD Procedure, 63-5
CREATE_DIAGNOSIS_TASK Functions, 157-10
CREATE_ERROR_LOG Procedure, 64-1
CREATE_ERROR_TABLE Procedure, 61-5
CREATE_EVALUATION_CONTEXT procedure, 142-8
CREATE_EVOLVE_TASK Function, 153-9
CREATE_EXTDDL_FOR_HIVE Procedure, 74-2
CREATE_EXTENDED_STATS Function, 162-36
CREATE_FILE Procedure, 16-11
CREATE_FILE_GROUP procedure, 66-11
CREATE_FILTER_SET Procedure, 187-11
CREATE_INDEX_COST procedure, 150-5
CREATE_JOB Procedure, 103-28
CREATE_LINK procedure, 182-2
CREATE_MISS_CAT procedure, 44-24
CREATE_MISS_NUM procedure, 44-25
CREATE_MSGSYSTEM_LINK Procedure for TIB/Rendezvous, 103-31
CREATE_MSGSYSTEM_LINK Procedure for WebSphere MQ, 103-32
CREATE_MVIEW function, 40-5, 40-30
CREATE_NORM_LIN procedure, 44-27
CREATE_OBJECT Procedure, 16-13
CREATE_OBJECT_DEPENDENCY procedure, 20-21
CREATE_OR_REPLACE_VIEW procedure of DBMS_HS_PARALLEL package, 80-1
CREATE_OUTBOUND Procedure, 212-60
CREATE_OUTLINE procedure, 111-3
CREATE_PENDING_AREA procedure, 134-12
CREATE_PIPE function, 119-13
CREATE_PLAN procedure, 134-13
CREATE_PLAN_DIRECTIVE procedure, 134-14
CREATE_POLICY_EXPRESSION Procedure, 129-20
CREATE_POLICY_GROUP Procedure, 138-14
CREATE_PROFILE Procedure, 156-5
CREATE_PROPAGATION procedure, 126-4
CREATE_REMOTE_SNAPSHOT function and procedure, 188-28
CREATE_REQUEST_CONTEXT Function, 265-34
CREATE_RULE procedure, 142-10
CREATE_RULE_SET procedure, 142-11
CREATE_SERVICE procedure, 145-6
CREATE_SIMPLE_PLAN procedure, 134-20
CREATE_SNAPSHOT function and procedure, 188-29
CREATE_SQL_PATCH Function, 157-12
CREATE_SQL_PLAN_BASELINE Procedure, 160-25
CREATE_SQLSET procedure, 159-9, 160-26
CREATE_SQLWKLKLD Procedure, 16-14
CREATE_STAT_TABLE Procedure, 162-38
CREATE_STMT_HANDLER procedure, 168-7
CREATE_TABLE_COST procedures, 150-6
CREATE_TABLE_COST_COLINFO Object Type, 150-1
CREATE_TABLE_TEMPLAGE procedure of DBMS_HS_PARALLEL package, 80-3
CREATE_TABLES Function, 79-3
CREATE_TASK Procedure, 113-8
CREATE_TASK Procedures, 16-15
CREATE_TEMP_HISTORY_TABLE Procedure, 69-4
CREATE_TOPOLOGY procedure, 182-3
CREATE_TRIGGER_TABLE Procedure, 61-5
CREATE_TUNING_TASK functions, 160-29
CREATE_VALIDATE_LOG_TABLE procedure of DBMS_HIERARCHY package, 77-2
CREATE_VERSION procedure, 66-12
CREATE_VIEW procedure, 89-4
CREATE_VIEW_ON_PATH procedure, 89-4
CREATE_WRAPPED Procedure, 52-5
CREATEATTRIBUTE function, 200-23
CREATEBUCKET Procedure, 50-7
CREATEDATASECTION function, 200-24
CREATECOMMENT function, 200-25
CREATEDATSTOREPREF procedure, 197-5
CREATEDATEINDEX Procedure, 202-2
CREATEDIRECTORY Procedures, 48-27, 49-5
CREATEDOCUMENT function, 200-25
CREATEDOCUMENTFRAGMENT function, 200-26
CREATEELEMENT function, 200-26
CREATEENTITYREFERENCE function, 200-27
CREATEFILE Procedures, 48-28, 49-6
CREATEFILTERPREF procedure, 197-5
CREATEFOLDER Function, 194-6
CREATEINDEX procedure, 197-5
CREATELEXERPREF procedure, 197-6
CREATEREFERENCE Procedures, 48-30, 49-7
CREATERESOURCE Functions, 194-7
CREATESCHEMABASEDXML function, 300-2
CREATENONCEKEY Procedure, 191-1, 198-2
CREATESCHEMABASEDXML function, 300-2
CREATESTOPLISTPREF procedure, 197-6
CREATESTORAGEPREF procedure, 197-6
CREATEINTERFACE Procedure, 50-8, 51-4
CREATETEMPORARY Procedures, 93-28
CREATETEXTnode function, 200-28
CREATEWORLDLISTPREF procedure, 197-7
CREATETEXTXML function, 300-3
creating
packages, 1-2
CSTORE_EXISTS function, 173-30
CSTORE_GET procedure, 173-31
CTX_ADM package, 3-1
CTX_ANL package, 4-1
CTX_CLS package, 5-1
CTX_DDL package, 6-1
CTX_DOC package, 7-1
CTX_ENTITY package, 8-1
CTX_OUTPUT package, 9-1
CTX_QUERY package, 10-1
CTX_REPORT package, 11-1
CTX_THES package, 12-1
CTX_ULEXER package, 13-1
cube aggregation, 40-14
Cube Build log, 42-4
Cube Dimension Compile log, 42-6
cube log verbosity levels, 42-3
cube logging targets, 42-2
cube logging types, 42-2
cube maintenance, 40-1
cube materialized views, 40-2
optimizing, 41-1
Cube Operations log, 42-6
cube refresh, 40-14
Cube Rejected Records Log, 42-7
CUBE_BUILD_LATEST view, 40-22
CUBE_UPGRADE_INFO table, 40-45
cubes
optimizer statistics, 31-1
CURRENT_EDITION Function, 260-6
CURRENT_INSTANCE Function, 183-12
CURRENT_SESSION_ID Function, 54-4
cursors
DBMS_SQL package, 154-5
CUSTOMIZE_ILM Procedure, 83-6
data dictionary
removing Streams information, 165-144
removing XStream information, 212-80
DATA function and procedure, 276-13
data mining, 43-97
algorithms, 43-2
anomaly detection, 43-2
association rules, 43-2
attribute importance, 43-2
automated, 121-1
classification, 43-2
clustering, 43-2
confusion matrix, 43-59
confusion matrix part, 43-65
cost matrix, 43-48
data transformation, 43-91, 43-140, 43-142
feature extraction, 43-2
lift, 43-72
lift part, 43-76
mining functions, 43-3
PMML, 43-138
regression, 43-2
ROC, 43-82, 43-86
scoring, 43-55, 43-155
supervised, 43-1
time series, 43-2
transformations, 44-1
unsupervised, 43-1
DATA_BLOCK_ADDRESS_BLOCK Function, 183-12
DATA_BLOCK_ADDRESS_FILE Function, 183-13
database
  locking
    OWA_OPT_LOCK package, 227-1
database tables
  creating for DBMS_TRACE, 175-3
DATABASE_TRACE_DISABLE Procedure, 105-4
DATABASE_TRACE_ENABLE Procedure, 105-4
datatypes
  DBMS_DESCRIBE, 56-7
  PL/SQL numeric codes for, 56-4
DATE_TABLE Table Type, 154-21
DB_CONNECT_INFO Procedure, 103-33
DB_VERSION Procedure, 183-13
DBFS_LINK_GENERATE_PATH Functions, 93-28
DBLINK_ARRAY Table Type, 183-3
DBMS, 200-21
DBMS_ALERT package, 17-1
DBMS_APPLY_ADM package, 20-1
DBMS_AQADM Constants, 22-2
DBMS_AQELM package, 23-1
DBMS_AQIN package, 24-1
DBMS_AUDIT_MGMT package
  ALTER_PARTITION_INTERVAL procedure, 26-8
  CLEAN_AUDIT_TRAIL procedure, 26-9
  CLEAR_AUDIT_TRAIL_PROPERTY procedure, 26-11
  CLEAR_LAST_ARCHIVE_TIMESTAMP procedure, 26-12
  CREATE_PURGE_JOB procedure, 26-14
  DEINIT_CLEANUP procedure, 26-15
  DROP_PURGE_JOB procedure, 26-16
  GET_AUDIT_COMMIT_DELAY function, 26-18
  INIT_CLEANUP procedure, 26-20
  IS_CLEANUP_INITIALIZED function, 26-22
  SET_AUDIT_TRAIL_LOCATION procedure, 26-24
  SET_AUDIT_TRAIL_PROPERTY procedure, 26-25
  SET_LAST_ARCHIVE_TIMESTAMP procedure, 26-29
  SET_PURGE_JOB_INTERVAL procedure, 26-30
  SET_PURGE_JOB_STATUS procedure, 26-31
DBMS_AUDIT_UTIL package
  DECODE_RLS_INFO_ATRAIL_FGA function, 27-2
DBMS_AUDIT_UTIL package (continued)
  DECODE_RLS_INFO_ATRAIL_STD function, 27-3
  DECODE_RLS_INFO_ATRAIL_UNI function, 27-4
  DECODE_RLS_INFO_ATRAIL_XML function, 27-5
  overview, 27-1
  security model, 27-1
  views used by, 27-2
DBMS_AUTO_TASK_ADMIN package, 30-1
DBMS_CAPTURE package, 284-1
DBMS_CAPTURE_ADM package, 32-1
DBMS_CHANGE_NOTIFICATION package, 36-1
DBMS_COMPARISON package, 33-1
  constants, 33-4
  data structures, 33-10
  index columns, 33-6
  lead index column, 33-6
  requirements, 33-6
  subprograms, 33-11
  views, 33-5
DBMS_CONNECTION_POOL package, 35-1
DBMS_CONTENT_CONTEXT_T Object Type, 285-2
DBMS_CSX_ADMIN package, 39-1
DBMS_CUBE package, 40-1
DBMS_CUBE_ADVISE package, 41-1
DBMS_CUBE_LOG package, 42-1
DBMS_DATA_MINING
  ESA, 43-22
  ESM, 43-23
DBMS_DATA_MINING package, 43-1, 43-97
  ADD_COST_MATRIX procedure, 43-48
  algorithms, 43-1
  ALTER_REVERSE_EXPRESSION procedure, 43-52
  APPLY procedure, 43-55
  Automatic Data Preparation, 43-6
  COMPUTE_CONFUSION_MATRIX procedure, 43-59
  COMPUTE_CONFUSION_MATRIX_PART procedure, 43-65
  COMPUTE_LIFT procedure, 43-72
  COMPUTE_LIFT_PART procedure, 43-76
  COMPUTE_ROC procedure, 43-82
  COMPUTE_ROC_PART procedure, 43-86
  Create Model Using Registration Information, 43-97
  CREATE_MODEL procedure, 43-91
  CREATE_MODEL2 procedure, 43-95
  data transformation, 43-91, 43-142
  datatypes, 43-40
  deprecated types, 43-41
  DROP_ALGORITHM procedure, 43-97
DBMS_DATA_MINING package (continued)
DROP_MODEL procedure, 43-98
DROP_PARTITION procedure, 43-98
EXPORT_MODEL procedure, 43-99
EXPORT_SERMODEL procedure, 43-102
FETCH_JSON_SCHEMA procedure, 43-103
GET_ASSOCIATION_RULES function, 43-103
GET_FREQUENT_ITEMSETS function, 43-108
GET_MODEL_COST_MATRIX function, 43-110
GET_MODEL_DETAILS_AI function, 43-112
GET_MODEL_DETAILS_EM function, 43-113
GET_MODEL_DETAILS_EM_COMP function, 43-115
GET_MODEL_DETAILS_EM_PROJ function, 43-117
GET_MODEL_DETAILS_GLM function, 43-118
GET_MODEL_DETAILS_GLOBAL function, 43-121
GET_MODEL_DETAILS_KM function, 43-123
GET_MODEL_DETAILS_NB function, 43-125
GET_MODEL_DETAILS_NMF function, 43-127
GET_MODEL_DETAILS_OC function, 43-128
GET_MODEL_DETAILS_SVM function, 43-129
GET_MODELSETTINGS function, 43-130
GET_MODEL_SIGNATURE function, 43-131
GET_MODEL_TRANSFORMATIONS function, 43-140
GET_TRANSFORM_LIST procedure, 43-142
IMPORT_MODEL procedure, 43-145
IMPORT_SERMODEL procedure, 43-150
JSON Schema for R Extensible Algorithm, 43-151
overview, 43-1
PMML, 43-138
RANK_APPLY procedure, 43-156
REGISTER_ALGORITHM procedure, 43-155
REMOVE_COST_MATRIX procedure, 43-159
RENAME_MODEL procedure, 43-160
scoring, 43-55, 43-64, 43-71, 43-74, 43-79, 43-155, 43-159
settings
ADMM, 43-39

DBMS_DATA_MINING package (continued)
settings (continued)
algorithm names, 43-5
CUR Matrix Decomposition, 43-17
decision tree, 43-17
GLM, 43-27
global, 43-11
k-Means, 43-29
LBFGS, 43-34
mining functions, 43-7
mining models, 43-4
Naive Bayes, 43-31
neural network, 43-31
NMF, 43-34
O-Cluster, 43-34
R Model Extensibility, 43-15
random forest, 43-35
solvers, 43-39
SVM, 43-38
subprograms, 43-46
transactional data, 43-12
DBMS_DATA_MINING Package
ADD_PARTITION procedure, 43-51
DBMS_DATA_MINING_TRANSFORM
datatypes, 44-14
package, 44-1
subprograms, 44-16
DBMS_DATAPUMP package, 45-1
ADD_FILE procedure, 45-13
ATTACH function, 45-17
DATA_FILTER procedure, 45-18
DETACH procedure, 45-22
GET_DUMPFILE_INFO procedure, 45-23
GET_STATUS procedure, 45-26
LOGENTRY procedure, 45-29
METADATA_FILTER procedure, 45-29
METADATA_REMAP procedure, 45-33
METADATA_TRANSFORM procedure, 45-35
OPEN function, 45-39
roles used by, 45-2
SET_PARALLEL procedure, 45-42
SET_PARAMETER procedure, 45-43
STARTJOB procedure, 45-52
STOPJOB procedure, 45-54
types used by, 45-3
WAITFOR_JOB procedure, 45-55
DBMS_DB_VERSION package, 46-1
DBMS_DBFS_CONTENT package, 48-1
DBMS_DBFS_CONTENT_LIST_ITEM_T Object Type, 285-2
DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type, 285-3
DBMS_DBFS_CONTENT_PROPERTIES_T Table Type, 285-4
DBMS_DBFS_CONTENT_PROPERTY_T Object Type, 285-3
DBMS_DBFS_CONTENT_RAW_T Table Type, 285-4
DBMS_DBFS_CONTENT_SPI package, 49-1
DBMS_DBFS_HS package, 50-1
DBMS_DBFS_SFS package, 51-1
DBMS_DDL package, 52-1
DBMS_DEBUG_JDWP package, 54-1
DBMS_DESCRIBE package, 56-1
DBMS_DG package, 57-1
using, 57-1
DBMS_DG Package
INITIATE_FS_FAILOVER procedure, 57-2
DBMS_DIMENSION package, 58-1
DBMS_DISTRIBUTED_TRUST_ADMIN package, 59-1
DBMS_EDITIONS_UTILS package, 62-1
CLEAN_UNUSABLE_EDITIONS Procedure, 62-2
DBMS_EPG package, 63-1
DBMS_FILE_GROUP package, 66-1
constants, 66-2
DBMS_FILE_TRANSFER package, 67-1
types, 68-2
DBMS_FS
UNMOUNT_ORACLE_FS, 71-6
DBMS_FS package
overview, 71-1
DBMS_FS package, 71-1
DESTROY_ORACLE_FS procedure, 71-2
MAKE_ORACLE_FS procedure, 71-3
MOUNT_ORACLE_FS procedure, 71-4
security model, 71-1
DBMS_GOLDENGATE_ADM package, 72-1
DBMS_GOLDENGATE_AUTH package, 73-1
DBMS_HADOOP package, 74-1
DBMS_HIERARCHY package, 77-1
overview, 77-1
security model, 77-1
DBMS_HPROF package, 79-1
DBMS_HS_PARALLEL package, 80-1
CREATE_OR_REPLACE_VIEW procedure, 80-1
CREATE_TABLE_TEMPLATE procedure, 80-3
DROP_VIEW procedure, 80-3
LOAD_TABLE procedure, 80-4
DBMS_HS_PASSTHROUGH package, 81-1
DBMS_ILM_ADMIN package
constants, 83-1
DBMS_INMEMORY package
POPULATE procedure, 84-3
REPOPULATE procedure, 84-4
DBMS_INMEMORY package (continued)
security model, 84-2
SEGMENT_DEALLOCATE_VERSIONS procedure, 84-4
DBMS_INMEMORY_ADMIN package, 85-1
FASTSTART_ENABLE procedure, 85-7
FASTSTART_MIGRATE_STORAGE procedure, 85-9
GET_FASTSTART_TABLESPACE function, 85-10
IME_CAPTURE_EXPRESSIONS procedure, 85-11
IME_CLOSE_CAPTURE_WINDOW procedure, 85-13
IME_DROP_ALL_EXPRESSIONS procedure, 85-14
IME_OPEN_CAPTURE_WINDOW procedure, 85-15
IME_POPULATE_EXPRESSIONS procedure, 85-16
operation notes, 85-3
overview, 85-1
security, 85-2
DBMS_IOT package, 86-1
DBMS_JAVA package, 87-1
DBMS_JSON constants, 89-1
DBMS_JSON package, 89-1, 89-2
DBMS_LDAP_UTL package, 91-1
DBMS_LOB package, 93-1
CLOB2FILE procedure, 93-16
deprecated subprograms, 93-1
DBMS_LOCK package, 94-1
DBMS_LOGMNR package
ADD_LOGFILE procedure, 95-5
COLUMN_PRESENT function, 95-7
END_LOGMNR procedure, 95-8
MINE_VALUE function, 95-9
REMOVE_LOGFILE procedure, 95-10
START_LOGMNR procedure, 95-11
DBMS_LOGMNR_D package
BUILD procedure, 96-2
SET_TABLESPACE procedure, 96-5
DBMS_LOGSTDBY package, 97-1
APPLY_SET procedure, 97-5
APPLY_UNSET procedure, 97-8
BUILD procedure, 97-8
INSTANTIATE_TABLE procedure, 97-15
MAP_PRIMARY_SCN function, 97-17
overview of managing SQL Apply, 97-1
PREPARE_FOR_NEW_PRIMARY procedure, 97-17
PURGE_SESSION procedure, 97-18
SET_TABLESPACE procedure, 97-20
SKIP procedure, 97-21
SKIP_ERROR procedure, 97-30
DBMS_LOGSTDBY package (continued)
  SKIP_TRANSACTION procedure, 97-34
  UNSKIP procedure, 97-35
  UNSKIP_ERROR procedure, 97-36
  UNSKIP_TRANSACTION procedure, 97-38
DBMS_MEMOPTIMIZE package, 99-1
DBMS_METADATA package
  ADD_TRANSFORM function, 100-6
  CLOSE procedure, 100-10
  CONVERT functions and procedures, 100-11
  GET_DDL function, 100-16
  GET_QUERY function, 100-19
  GET_XML function, 100-16
  OPEN function, 100-20
  OPENW function, 100-27
  PUT function, 100-28
  security, 100-2
  SET_COUNT procedure, 100-29
  SET_FILTER procedure, 100-30
  SET_PARSE_ITEM procedure, 100-42
  SET_REMAP_PARAM procedure, 100-45
  SET_TRANSFORM_PARAM procedure, 100-45
DBMS_METADATA_DIFF package
  ADD_DOCUMENT procedure, 101-5
  FETCH_CLOB functions and procedures, 101-5
  OPENC function, 101-4
DBMS_MGD_ID_UTL package, 102-1
  ADD_SCHEME procedure, 102-4
  CREATE_CATEGORY function, 102-8
  EPC_TO_ORACLE_SCHEME function, 102-8
  GET_CATEGORY_ID function, 102-11
  GET_COMPONENTS function, 102-12
  GET_ENCODINGS function, 102-13
  GET_JAVA_LOGGING_LEVEL function, 102-14
  GET_PLSQL_LOGGING_LEVEL function, 102-15
  GET_SCHEMENAMES function, 102-15
  GET_TDT_XML function, 102-16
  GET_VALIDATOR function, 102-17
  REFRESH_CATEGORY function, 102-21
  REMOVE_CATEGORY procedure, 102-22
  REMOVE_PROXY procedure, 102-23
  REMOVE_SCHEMA procedure, 102-23
  SET_JAVA_LOGGING_LEVEL procedure, 102-24
  SET_PLSQL_LOGGING_LEVEL procedure, 102-24
  SET_PROXY procedure, 102-25
  VALIDATE_SCHEMA function, 102-26
DBMS_MGWADM package
  summary of subprograms, 103-10

DBMS_MGWMSG package
  constants, 104-1
  summary of subprograms, 104-18
DBMS_MONITOR package
  statistics tracing and gathering
  DBMS_MONITOR package, 105-1
DBMS_MVIEW package
  BEGIN_TABLE_REORGANIZATION procedure, 106-3
  END_TABLE_REORGANIZATION procedure, 106-3
  EXPLAIN_MVIEW procedure, 106-4
  EXPLAIN_REWRITE procedure, 106-5
  I_AM_A_REFRESH function, 106-6
  PMARKER function, 106-7
  PURGE_DIRECT_LOAD_LOG procedure, 106-7
  PURGE_LOG procedure, 106-7
  PURGE_MVIEW_FROM_LOG procedure, 106-8
  REFRESH procedure, 106-9
  REFRESH_ALL_MVIEWS procedure, 106-11
  REFRESH_DEPENDENT procedure, 106-12
  REGISTER_MVIEW procedure, 106-15
  UNREGISTER_MVIEW procedure, 106-16
DBMS_MVIEW_STATS Overview, 107-1
DBMS_MVIEW_STATS package, 107-1
DBMS_MVIEW_STATS security model, 107-1
DBMS_ODCI package, 110-1
  ESTIMATE_CPU_UNITS function, 110-1
  methods, 110-1
DBMS_OUTPUT package, 112-1
DBMS_PARALLEL_EXECUTE package, 113-1
DBMS_PDB_ALTER_SHARING package, 117-1
DBMS_PIPE package, 119-1
DBMS_PLSQL_CODE_COVERAGE package, 120-1
DBMS_PREDICTIVE_ANALYTICS package, 121-1
  EXPLAIN procedure, 121-2
  PREDICT procedure, 121-4
  PROFILE Procedure, 121-6
DBMS_PREPROCESSOR package, 122-1
DBMS_PROCESS package
  overview, 124-1
  security model, 124-1
  DBMS_PROCESS procedure
    CONFIGURE_POOL procedure, 124-2
    START_POOL procedure, 124-3
    STOP_POOL procedure, 124-4
    DBMS_PROCESS package, 124-1
    overview, 124-1
    security model, 124-1
    DBMS_PROCESS procedure
      CONFIGURE_POOL procedure, 124-2
      DBMS_PROPAGATION_ADM package, 126-1
      DBMS_RANDOM package, 128-1
      DBMS_RESCONFIG package, 133-1
      DBMSRESOURCE_MANAGER package, 134-1
DBMS_RESOURCE_MANAGER_PRIVS package, 135-1
DBMS_RESULT_CACHE package, 136-1
DBMS_RESUMABLE package, 137-1
DBMS_RLS package
  fine-grained access control
  DBMS_RLS package, 138-1
DBMS_ROWID package, 140-1
DBMS_RULE package, 141-1
DBMS_RULE_ADM package, 142-1
DBMS_SCHEDULER package, 143-1
  ADD_EVENT_QUEUE_SUBSCRIBER Procedure, 143-26
  ADD_GROUP_MEMBER Procedure, 143-27
  ADD_JOB_EMAIL_NOTIFICATION Procedure, 143-28
  ADD_TO_INCOMPATIBILITY Procedure, 143-30
  ALTER_CHAIN Procedure, 143-31
  ALTER_RUNNING_CHAIN Procedure, 143-33
  CLOSE_WINDOW Procedure, 143-35
  COPY_JOB Procedure, 143-36
  CREATE_CHAIN Procedure, 143-37
  CREATE_CREDENTIAL Procedure, 143-38
  CREATE_DATABASE_DESTINATION Procedure, 143-39
  CREATE_EVENT_SCHEDULE Procedure, 143-40
  CREATE_FILE_WATCHER Procedure, 143-41
  CREATE_GROUP Procedure, 143-42
  CREATE_INCOMPATIBILITY Procedure, 143-44
  CREATE_JOB Procedure, 143-45
  CREATE_JOB_CLASS Procedure, 143-53
  CREATE_JOBS Procedure, 143-55
  CREATE_PROGRAM Procedure, 143-56
  CREATE_Resource Procedure, 143-60
  CREATE_SCHEDULE Procedure, 143-61
  CREATE_WINDOW Procedure, 143-62
  DEFINE_ANYDATA_ARGUMENT Procedure, 143-64
  DEFINE_CHAIN_EVENT_STEP Procedure, 143-65
  DEFINE_CHAIN_RULE Procedure, 143-66
  DEFINE_CHAIN_STEP Procedure, 143-69
  DEFINE_METADATA_ARGUMENT Procedure, 143-70
  DEFINE_PROGRAM_ARGUMENT Procedure, 143-71
  DISABLE Procedure, 143-73
  DROP_AGENT_DESTINATION Procedure, 143-75
  DROP_CHAIN Procedure, 143-76
DBMS_SCHEDULER package (continued)
  DROP_CHAIN_RULE Procedure, 143-77
  DROP_CHAIN_STEP Procedure, 143-78
  DROP_CREDENTIAL Procedure, 143-78
  DROP_DATABASE_DESTINATION Procedure, 143-79
  DROP_FILE_WATCHER Procedure, 143-80
  DROP_GROUP Procedure, 143-81
  DROP_INCOMPATIBILITY Procedure, 143-81
  DROP_JOB Procedure, 143-82
  DROP_JOB_CLASS Procedure, 143-83
  DROP_PROGRAM Procedure, 143-84
  DROP_PROGRAM_ARGUMENT Procedure, 143-84
  DROP_RESOURCE Procedure, 143-85
  DROP_SCHEDULE Procedure, 143-86
  DROP_WINDOW Procedure, 143-86
  ENABLE Procedure, 143-87
  END_DETACHED_JOB_RUN Procedure, 143-88
  EVALUATE CALENDAR STRING Procedure, 143-89
  EVALUATE_RUNNING_CHAIN Procedure, 143-91
  GENERATE_JOB_NAME Function, 143-92
  GET_AGENT_INFO Function, 143-92
  GET_AGENT_VERSION Function, 143-93
  GET_ATTRIBUTE Procedure, 143-94
  GET_FILE Procedure, 143-94
  GET_SCHEDULER_ATTRIBUTE Procedure, 143-96
  JOB_DEFINITION constructor function, 143-14
  JOB_DEFINITION object type, 143-14
  JOB_DEFINITION_ARRAY table type, 143-17
  JOBARG constructor function, 143-12
  JOBARG object type, 143-12
  JOBARG_ARRAY table type, 143-12, 143-14
  JOBATTR constructor function, 143-18
  JOBATTR object type, 143-18
  JOBATTR_ARRAY table type, 143-18, 143-19
  OPEN_WINDOW Procedure, 143-97
  PURGE_LOG Procedure, 143-98
  PUT_FILE Procedure, 143-99
  REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure, 143-100
  REMOVE_FROM_INCOMPATIBILITY Procedure, 143-101
  REMOVE_GROUP_MEMBER Procedure, 143-101
  REMOVE_JOB_EMAIL_NOTIFICATION Procedure, 143-102
DBMS_SCHEDULER package (continued)
RESCHEDULE_JOB_ARGUMENT_VALUE Procedure, 143-103
RUN_CHAIN Procedure, 143-104
RUN_JOB Procedure, 143-105
Scheduler Chain Condition Syntax, 143-66
SCHEDULER_FILEWATCHER_REQUEST Object Type, 143-22
SCHEDULER_FILEWATCHER_RESULT Object Type, 143-21
SCHEDULER$_EVENT_INFO Object Type, 143-20
SCHEDULER$_STEP_TYPE object type, 143-19
SCHEDULER$_STEP_TYPE_LIST table type, 143-20
SET_AGENT_REGISTRATION_PASS Procedure, 143-107
SET_ATTRIBUTE Procedure, 143-108
SET_ATTRIBUTE_NULL Procedure, 143-124
SET_JOB_ANYDATA_VALUE Procedure, 143-124
SET_JOB_ARGUMENT_VALUE Procedure, 143-125
SET_JOB_ATTRIBUTES Procedure, 143-127
SET_RESOURCE_CONSTRAINT Procedure, 143-127
SET_SCHEDULER_ATTRIBUTE Procedure, 143-128
STOP_JOB Procedure, 143-130
DBMS_SERVER_ALERT package, 144-1
DBMS_SERVICE package, 145-1
DBMS_SESSION package, 146-1
DBMS_SODA subprograms, 148-1
DBMS_SODA package, 148-1
ADD_REFERENCE function, 159-6
COLUMN_DATA_T, 173-22
COLUMN_METADATA_T Type, 173-23
COLUMN_T, 173-24
COLUMN_TYPE_NAME, 173-28
COLUMNS pseudo-operator, 173-11
COLUMNS_APPEND_T, 173-27
COLUMNS_APPEND_T, 173-27
COLUMNS_APPEND_T, 173-27
DATA structures, 173-2
DELETE_SQLSET procedure, 159-11
DBMS_SQLSET package (continued)
DROP_SQLSET procedure, 159-11
LOAD_SQLSET procedure, 159-12
overview, 159-1
PACK_STGTAB procedure, 159-16
REMAP_STGTAB procedure, 159-18
REMOVE_REFERENCE procedure, 159-19
security model, 159-2
SELECT_CURSOR_CACHE function, 159-20
SELECT_SQL_TRACE function, 159-25
SELECT_SQLPA_TASK function, 159-28
SELECT_SQLSET function, 159-29
SELECT_WORKLOAD_REPOSITORY function, 159-32
DBMS_TDB package, 172-1
DBMS_TDB.CHECK_DB Function, 172-3
DBMS_TDB.CHECK_DB procedure, 172-5
DBMS_TF package, 173-1
client interface, 173-3
COLUMN_DATA_T, 173-22
COLUMN_METADATA_T Type, 173-23
COLUMN_T, 173-24
COLUMN_TYPE_NAME, 173-28
COLUMNS pseudo-operator, 173-11
COLUMNS_APPEND_T, 173-27
COLUMNS_APPEND_T, 173-27
COLUMNS_APPEND_T, 173-27
DATA structures, 173-2
DELETE_SQLSET procedure, 159-11
DBMS_TF package (continued)
CSTORE compilation state management, 173-7
CSTORE_GET, 173-31
data structures, 173-20
DESCRIBE_T, 173-24
ENV_T, 173-25
equivalents, 173-16
execution flow, 173-13
GET_COL, 173-32
GET_ENV, 173-33
GET_ROW_SET, 173-34
GET_XID, 173-38
method name overrides, 173-10
parallel execution, 173-12
PARALLEL_ENV_T, 173-25
pass-through columns, 173-7
PUT_COL, 173-38
PUT_ROW_SET, 173-42
query transformations, 173-11
read columns, 173-6
restrictions, 173-15
ROW_REPLICATION, 173-45
ROW_SET_T, 173-27
ROW_TO_CHAR, 173-47
server side interface, 173-5
SUPPORTED_TYPE, 173-48
TAB_BINARY_DOUBLE_T, 173-21
TAB_BINARY_FLOAT_T, 173-21
TAB_BLOB_T, 173-21
TAB_BOOLEAN_T, 173-21
TAB_CHAR_T, 173-21
TAB_CLOB_T, 173-21
TAB_DATE_T, 173-21
TAB_INTERVAL_DS_T, 173-21
TAB_INTERVAL_YM_T, 173-21
TAB_NATURALN_T, 173-21
TAB_NUMBER_T, 173-21
TAB_RAW_T, 173-21
TAB_ROWID_T, 173-21
TAB_TIMESTAMP_LTZ_T, 173-21
TAB_TIMESTAMP_T, 173-21
TAB_TIMESTAMP_TZ_T, 173-21
TAB_VARCHAR2_T, 173-21
TABLE_COLUMNS_T, 173-27
TABLE_T, 173-26
TRACE, 173-48
XID_T, 173-27
XSTORE execution state management, 173-9
XSTORE_CLEAR, 173-50
XSTORE_EXISTS, 173-50
XSTORE_GET, 173-50
XSTORE_REMOVE, 173-54
XSTORE_SET, 173-54

DBMS_TF_ROW_REPLICATION procedure, 173-45
DBMS_TF_TRACE procedure, 173-48
DBMS_TNS package, 173-48
overview, 174-1
security model, 174-1
DBMS_TNS package RESOLVE_TNSNAME function, 174-2
DBMS_TRACE package, 175-1
DBMS_TRANSACTION package, 176-1
DBMS_TRANSFORM package, 177-1
DBMS_TTS package, 180-1
DBMS_UMF package, 182-1
DBMSUTILITY package, 183-1
deprecated subprograms, 183-1
DBMS_WM package, 185-1
DBMS_WORKLOAD_RECORD package, 186-1
DBMS_WORKLOAD_REPLAY package, 187-1
DBMS_XA_XID Object Type, 189-4
DBMS_XA_XID ARRAY Table Type, 189-5
DBMS_XDB_VERSION package, 195-1
DBMS_XDB Constants, 190-2
DBMS_XDB Overview, 190-1
DBMS_XDB package, 190-1

CONFIGUREAUTOSYNC procedure, 197-4
CREATEDATASTOREPREF procedure, 197-5
CREATEFILTERPREF procedure, 197-5
CREATEINDEX procedure, 197-5
CREATELEXERPREF procedure, 197-6
CREATEPREFERENCES procedure, 197-6
CREATESECTIONGROUPPREF procedure, 197-6
CREATESTOPLISTPREF procedure, 197-6
CREATEGROUPPREFERENCES procedure, 197-7
CREATEWORLDLISTPREF procedure, 197-7
DROPREFERENCES procedure, 197-7
methods, 190-3, 197-3
DBMS_XDB_ADMIN package, 191-1
DBMS_XDB_VERSION package
CHECKIN function, 195-2
CHECKOUT procedure, 195-2
GETCONTENTSBLOBBYRESID function, 195-3
GETCONTENTSCLOBBYRESID function, 195-3
GETCONTENTSEXMLBYRESID function, 195-4
GETPREDECESSORS function, 195-4
GETPREDRESBYRESID function, 195-4
GETRESOURCESBYRESID function, 195-5
GETSUCCESRESID function, 195-5
GETSUCCSBYRESID function, 195-6
MAKEVERSIONED function, 195-6
DBMS_XDB_VERSION package (continued)
  UNCHECKOUT function, 195-7
DBMS_XDBRESOURCE package, 196-1
DBMS_XDBT package
  CONTEXT synchronization settings, 197-2
  filtering settings, 197-2
  general indexing settings, 197-2
  miscellaneous settings, 197-2
  other index preference settings, 197-2
  sectioning and section group settings, 197-2
  stoplist settings, 197-2
  SYNC settings, 197-2
DBMS_XDBZ package
  DISABLE_HIERARCHY procedure, 198-3
  ENABLE_HIERARCHY procedure, 198-3
  GET_ACOloid function, 198-4
  GET_USERID function, 198-4
  IS_HIERARCHY_ENABLED function, 198-5
  PURGELDAPCACHE function, 198-6
DBMS_XMLDOM package
  APPENDDATA procedure, 200-22
  CREATEATTRIBUTE function, 200-23
  CREATECDATASECTION function, 200-24
  CREATECOMMENT function, 200-25
  CREATEDOCUMENT function, 200-25
  CREATEDOCUMENTFRAGMENT function, 200-26
  CREATEELEMENT function, 200-26
  CREATEENTITYREFERENCE function, 200-27
  CREATEPROCESSINGINSTRUCTION function, 200-27
  CREATETEXTNODE function, 200-28
  DELETEDATA procedure, 200-29
  description, 200-1
  exceptions, 200-4
  FINDENTITY function, 200-29
  FINDNOTATION function, 200-30
  FREEDOCFRAG procedure, 200-30
  FREEDOCUMENT procedure, 200-31
  GETATTRIBUTE function, 200-33
  GETATTRIBUTENODE function, 200-34
  GETCHILDRENBYTAGNAME function, 200-36
  GETDATA function, 200-37
  GETDOCTYPE function, 200-37
  GETDOCUMENTELEMENT function, 200-38
  GETELEMENTSBYTAGNAME function, 200-38
  GETENTITIES function, 200-39
  GETIMPLEMENTATION function, 200-41
  GETLENGTH function, 200-42
  GETNAME function, 200-43, 200-44
  GETNAMEDIITEM function, 200-44
  GETNAMESPACE procedure, 200-44
DBMS_XMLDOM package (continued)
  GETNEXTSIBLING function, 200-45
  GETNODENAME function, 200-46
  GETNODETYPE function, 200-46
  GETNODEVALUE function, 200-47
  GETNOTATIONNAME function, 200-49
  GETNOTATIONS function, 200-49
  GETOWNERDOCUMENT function, 200-50
  GETOWNERELEMENT function, 200-51
  GETPARENTNODE function, 200-51
  GETPREFIX function, 200-52
  GETPREVIOUSSIBLING function, 200-52
  GETPUBLICID function, 200-53
  GETQUALIFIEDNAME function, 200-53
  GETSCHEMANODE function, 200-54
  GETSPECIFIED function, 200-54
  GETSTANDALONE function, 200-55
  GETSYSTEMID function, 200-55
  GETTAGNAME function, 200-56
  GETTARGET function, 200-50
  GETVALUE function, 200-56
  GETVERSION function, 200-57
  GETXMLTYPE function, 200-57
  IMPORTNODE function, 200-60
  INSERTBETWEEN function, 200-61
  INSERTDATA procedure, 200-62
  ISNULL function, 200-62
  MAKEATTR function, 200-66
  MAKECDATASECTION function, 200-66
  MAKECHARACTERDATA function, 200-67
  MAKECOMMENT function, 200-67
  MAKEDOCUMENT function, 200-68
  MAKEDOCUMENTFRAGMENT function, 200-68
  MAKEDOCUMENTTYPE function, 200-69
  MAKEELEMENT function, 200-69
  MAKEENTITY function, 200-70
  MAKEENTITYREFERENCE function, 200-70
  MAKENODE function, 200-71
  MAKENOTATION function, 200-73
  MAKEPROCESSINGINSTRUCTION function, 200-73
  MAKETEXT function, 200-74
  methods
    APPENDCHILD function, 200-22
    APPENDDATA procedure, 200-22
    CLONENODE function, 200-23
    CREATEATTRIBUTE function, 200-23
    CREATEDOCUMENTDATASECTION function, 200-24
    CREATEELEMENT function, 200-25
DBMS_XMLDOM package (continued)
methods (continued)
CREATEDOCUMENT function, 200-25
CREATEDOCUMENTFRAGMENT function, 200-26
CREATEELEMENT function, 200-26
CREATEENTITYREFERENCE function, 200-27
CREATEPROCESSINGINSTRUCTION function, 200-27
CREATETEXTNODE function, 200-28
DELETEDATA procedure, 200-29
DOMAttr interface, 200-7
DOMCDataSection interface, 200-7
DOMCharacterData interface, 200-8
DOMComment interface, 200-8
DOMDocument interface, 200-8
DOMDocumentFragment interface, 200-9, 200-21
DOMDocumentType interface, 200-10, 200-21
DOMElemnet interface, 200-10, 200-21
DOMEntity interface, 200-11, 200-21
DOMEntityReference interface, 200-11
DOMImplementation interface, 200-12
DOMNode interface, 200-12
DOMNodeAPPENDCHILD function, 200-22
DOMNodeCLONENODE function, 200-23
DOMNodeGETATTRIBUTES function, 200-34
DOMNodeGETCHILDNODES function, 200-35
DOMNodeGETEXPANDEDNAME procedure, 200-39
DOMNodeGETFIRSTCHILD function, 200-40
DOMNodeGETIMPLEMENTATION function, 200-41
DOMNodeGETLASTCHILD function, 200-41
DOMNodeGETLENGTH function, 200-42
DOMNodeGETOWNERDOCUMENT function, 200-50
DOMNodeGETOWNERELEMENT function, 200-51
DOMNodeGETPARENTNODE function, 200-51
DOMNodeGETPREFIX function, 200-52
DOMNodeGETPREVIOUSSiblingFunction, 200-52
DOMNodeGETPUBLICID function, 200-53
DOMNodeGETQUALIFIEDNAME function, 200-53
DOMNodeGETSCHEMAnODE function, 200-54
DOMNodeGETSPECIFIED function, 200-54
DOMNodeGETSTANDALONE function, 200-55
DOMNodeGETTARGET function, 200-50
DOMNodeGETVALUE function, 200-56
DOMNodeGETVERSION function, 200-57
DOMNodeGETXMLTYPE function, 200-57
DOMHasATTRIBUTE function, 200-58
DOMHasATTRIBUTES function, 200-59
DOMHasCHILDREN function, 200-59
DOMHasFEATURE function, 200-60
DOMIMPORTNODE function, 200-60
DOMINSERTBEFORE function, 200-61
DOMISNULL function, 200-62
DOMMAKEATTR function, 200-66
DOMMAKECDATASECTION function, 200-66
DBMS_XMLDOM package (continued)
methods (continued)
MAKECHARACTERDATA function, 200-67
MAKECOMMENT function, 200-67
MAKEDOCUMENT function, 200-68
MAKEDOCUMENTFRAGMENT function, 200-68
MAKEDOCUMENTTYPE function, 200-69
MAKEELEMENT function, 200-69
MAKEENTITY function, 200-70
MAKEENTITYREFERENCE function, 200-70
MAKENODE function, 200-71
MAKENOTATION function, 200-73
MAKEPROCESSINGINSTRUCTION function, 200-73
MAKETEXT function, 200-74
NEWDOMDOCUMENT function, 200-74
NORMALIZE procedure, 200-75
REMOVEATTRIBUTE procedure, 200-75
REMOVEATTRIBUTENODE function, 200-76
REMOVENAMEDITEM function, 200-77
REPLACECHILD function, 200-78
REPLACEDATA procedure, 200-79
RESOLVENAMESPACEPREFIX function, 200-79
SETATTRIBUTE procedure, 200-80
SETATTRIBUTENODE function, 200-81
SETDATA procedure, 200-82
SETNAMEDITEM function, 200-83
SETNODEVALUE procedure, 200-84
SETPREFIX procedure, 200-86
SETSTANDALONE procedure, 200-86
SETVALUE procedure, 200-87
SETVERSION procedure, 200-87
SPLITTEXT function, 200-88
SUBSTRINGDATA function, 200-88
types, 200-3
WRITETOBUFFER procedure, 200-89
WRITETOCLOB procedure, 200-90
WRITETOFILE procedure, 200-91
DBMS_XMLGEN package, 201-1
CLOSECONTEXT procedure, 201-2
CONVERT function, 201-2
GETNUMROWSPROCESSED Function, 201-3
GETXML function, 201-4
GETXMLTYPE function, 201-5
NEWCONTEXT function, 201-5
RESTARTQUERY procedure, 201-6
SETCONVETSPESIALCHARS procedure, 201-7
SETMAXROWS procedure, 201-7
SETROWSETTAG procedure, 201-8
SETROWTAG procedure, 201-9
SETSKIPROWS procedure, 201-10
USEITEMTAGSFORCOLL procedure, 201-10
USENULLATTRIBUTEINDICATOR procedure, 201-11
DBMS_XMLINDEX package, 202-1
DBMS_XMLPARSER package, 203-1
FREEPARSER procedure, 203-2
GETDOCTYPE function, 203-3
GETDOCUMENT function, 203-3
GETRELEASEVERSION function, 203-3
GETVALIDATIONMODE function, 203-4
NEWPARSER function, 203-4
PARSE function, 203-4
PARSE procedure, 203-4
PARSEBUFFER procedure, 203-5
PARSECLOB procedure, 203-5
PARSEDTD procedure, 203-6
PARSEDTDBUFFER procedure, 203-6
PARSEDTCLOB procedure, 203-7
SETBASEDIR procedure, 203-7
SETDOCTYPE procedure, 203-8
SETERRORLOG procedure, 203-8
SETPRESEVWHITESPACE procedure, 203-9
SETVALIDATIONMODE procedure, 203-9
SHOWWARNINGS procedure, 203-9
DBMS_XMLQUERY package, 204-1
CLOSECONTEXT procedure, 204-4
constants, 204-1
GETDTD function, 204-4
GETDTD procedure, 204-4
GETEXCEPTIONCONTENT procedure, 204-5
GETNUMROWSPROCESSED procedure, 204-5
GETVERSION procedure, 204-6
GETXML function, 204-6
GETXML procedure, 204-6
NEWCONTEXT function, 204-7
PROPAGATEORIGINALEXCEPTION procedure, 204-7
REMOVEXSLTPARAM procedure, 204-8
SETBINDVALUE procedure, 204-8
SETCOLLIDATTRNAME procedure, 204-9
SETDATEHEADER procedure, 204-9
SETDATEFORMAT procedure, 204-9
SETENCODINGTAG procedure, 204-10
SETERRORTAG procedure, 204-10
SETMAXROWS procedure, 204-11
SETMETAHEADER procedure, 204-11
SETRAISEEXCEPTION procedure, 204-11
SETSENOROWSEXCEPTION procedure, 204-12
SETROWIDATTRVALUE procedure, 204-12
SETROWIDATTNAME procedure, 204-12
SETROWTAG procedure, 204-13
SETSKIPROWS procedure, 204-14
SETSQLTOXMLNAMEESCAPING procedure, 204-14
SETSTYLESDATEHEADER procedure, 204-14
SETTAGCASE procedure, 204-15
SETXSLT procedure, 204-15
SETXSLTPARAM procedure, 204-16
USENULLATTRIBUTEINDICATOR procedure, 204-16
USETYPEFORCOLLELEM TAG procedure, 204-17
DBMS_XMLSAVE package, 205-1
CLEARKEYCOLUMNLIST procedure, 205-3
CLEARUPDATECOLUMNLIST procedure, 205-3
CLOSECONTEXT procedure, 205-4
constants, 205-1
DELETEXML function, 205-4
GETEXCEPTIONCONTENT procedure, 205-5
INSERTXML function, 205-6
NEWCONTEXT function, 205-6
DBMS_XMLSAVE package (continued)
PROPAGATEORIGINALEXCEPTION procedure, 205-7
REMOVEXSLTPARAM procedure, 205-7
SETBATCHSIZE procedure, 205-7
SETCOMMITBATCH procedure, 205-8
SETDATEFORMAT procedure, 205-8
SETIGNORECASE procedure, 205-9
SETKEYCOLUMN procedure, 205-9
SETPRESERVEWHITESPACE procedure, 205-10
SETROWTAG procedure, 205-10
SETSQLTOXMLNAMEESCAPING procedure, 205-11
SETUPDATECOLUMN procedure, 205-11
SETXSLT procedure, 205-11
SETXSLTPARAM procedure, 205-12
UPDATEXML function, 205-13
DBMS_XMLSchema Constants, 206-2
DBMS_XMLSchema Operational Notes, 206-4
DBMS_XMLSchema package, 206-1
DBMS_XMLSchema Views, 206-3
DBMS_XMLSTORE package, 209-1
CLEARKEYCOLUMNLIST procedure, 209-2
CLEARUPDATECOLUMNLIST procedure, 209-2
CLOSECONTEXT procedure, 209-3
DELETEXML function, 209-3
INSERTXML function, 209-4
NEWCONTEXT function, 209-5
SETKEYCOLUMN procedure, 209-5
SETROWTAG procedure, 209-5
SETUPDATECOLUMN procedure, 209-6
types, 209-1
UPDATEXML function, 209-6
DBMS_XSLPROCESSOR package
CLOB2FILE function, 211-3
deprecated subprograms, 211-2
FREEPROCESSOR function, 211-4
FREESTYLESHEET function, 211-4
NEWPROCESSOR function, 211-4
NEWSTYLESHEET function, 211-4
PROCESSXSL function, 211-5
READ2CLOB function, 211-7
REMOVEPARAM procedure, 211-8
RESETPARAMS procedure, 211-8
SELECTNOD ES function, 211-9
SELECTSINGLENO DE function, 211-9
SETERRORLOG procedure, 211-10
SETPARAM procedure, 211-10
SHOWWARNINGS procedure, 211-11
TRANSFORMNODE function, 211-11
VALUEOF procedure, 211-11
DBMS_XSLPROCESSOR Package, 211-1
DBMS_XSTREAM_ADM package, 212-1
security, 212-2
DBMS_OUTPUT_LINESARRAY Object Type, 112-5
DBSFWUSER.DBMS_SFW_ACL_ADMIN package, 147-1
DBUriType, 286-8
DBURIType function, 286-9
DBUriType subtype, 286-8
CREATEURI function, 286-9
GETBLOB function, 286-9
GETCLOB function, 286-10
GETCONTENTTYPE function, 286-11
GETEXTERNALURL function, 286-11
GETURL function, 286-11
GETXML function, 286-11
methods, 286-8
DEAUTHORIZE_DAD Procedure, 63-5
DEBUG_EXPTOC package, 214-1
DEBUG_ON procedure, 53-17
DECLARE_REWRITE_EQUIVALENCE Procedures, 15-3
DECODEFEATURES Function, 48-31
DECRYPT Function, 38-8
DECRYPT Procedures, 38-9
DECRYPT_CAPTURE Procedure, 186-4
DEFAULT_NAME function, 42-9
DEFINE_ARRAY procedure, 154-38
DEFINE_COLUMN procedure, 154-43
DEFINE_COLUMN_CHAR Procedure, 154-44
DEFINE_COLUMN_LONG procedure, 154-45
DEFINE_COLUMN_RAW Procedure, 154-45
DEFINE_COLUMN_ROWID Procedure, 154-46
DEINIT_CLEANUP procedure, 26-15
DELETE Procedure, 14-8
DELETE_ALL_ERRORS procedure, 20-22
DELETE_BREAKPOINT function, 53-18
DELETE_CAPTURE_INFO Procedure, 186-5
DELETE_CATEGORY Procedure, 134-21
DELETE_CDB_PLAN Procedure, 134-21
DELETE_CDB_PLAN_DIRECTIVE Procedure, 134-22
DELETE_CDB_PROFILE_DIRECTIVE Procedure, 134-22
DELETE_COLUMN member procedure, 289-11
DELETE_COLUMN procedure, 289-11
DELETE_COLUMN_STATS procedure, 162-38
DELETE_CONSUMER_GROUP procedure, 162-40
DELETE_DAD_ATTRIBUTE Procedure, 63-6, 63-7
DELETE_DATABASE_PREFS procedure, 162-40
DELETE_DATABASE_STATS procedure, 162-45
DELETE_DICTIONARY_STATS procedure, 162-46
DELETE_ERROR procedure, 20-22
DELETE_FILTER Procedure, 186-5, 187-11
DELETE_FINDING_DIRECTIVE Procedure, 14-9
DELETE_FIXED_OBJECTS_STATS procedure, 162-47
DELETE_INDEX_STATS procedure, 162-48
DELETE_OBJECT procedure, 275-3
DELETE_OER_BREAKPOINT function, 53-18
DELETE_PARAMETER_DIRECTIVE Procedure, 14-9
DELETE_PENDING_STATS Procedure, 162-50
DELETE_PLAN procedure, 134-23
DELETE_PLAN_CASCADE procedure, 134-24
DELETE_PLAN_DIRECTIVE procedure, 134-24
DELETE_POLICY_GROUP Procedure, 138-14
DELETE_PRIVILEGE Procedure, 108-16
DELETE_PROCESSING_RATE Procedure, 162-50
DELETE_PROCREP_EXCLUSION_OBJ procedure, 72-8
DELETE_REPLAY_INFO Procedure, 187-12
DELETE_RUN Procedure, 123-5
DELETE_SCHEMA_PREFS procedure, 162-51
DELETE_SCHEMA_STATS Procedure, 162-57
DELETE_SEGMENT_DIRECTIVE Procedure, 14-10
DELETE_SERVICE procedure, 145-10
DELETE_SQL_CONNECTION_TEST Procedure, 18-3, 146-5
DELETE_SQL_DIRECTIVE Procedure, 14-10
DELETE_SQLSET procedure, 159-11, 160-35
DELETE_SQLWKLD Procedure, 16-17
DELETE_SQLWKLD_REF Procedure, 16-18
DELETE_SQLWKLD_STATEMENT Procedure, 16-19
DELETE_SYSTEM_STATS procedure, 162-58
DELETE_TABLE_PREFS procedure, 162-59
DELETE_TABLE_STATS procedure, 162-64
DELETE_TASK Procedure, 16-21
DELETECONTENT Procedure, 48-31, 49-8
DELETEDATA procedure, 200-29
DELETEDIRECTORY Procedure, 48-32, 49-8
DELETETFILE Procedure, 48-32, 49-9
DELETEHTTPEXPIREMAPPING Procedure, 192-9
DELETEIMEMEMAPPING Procedure, 192-9
DELETE容器务REPOSITORYRESCONFIG Procedure, 133-4
DELETERESCONFIGN Procedure, 133-5
DELETERESOURCE Procedure, 194-9
DELETERESOURCEREMETADATA Procedures, 194-10
DELETESCEHMA procedure, 206-8
DELETESCHEMALOCMAPPING Procedure, 192-9
DELETESERVLET Procedure, 192-10
DELETESERVLETMAPPING Procedure, 192-10
DELETESERVLETSECROLE Procedure, 192-10
DELETESXML function, 205-4, 209-3
DELETEXMLEXTENSION Procedure, 192-11
DEQUEUE procedure, 169-2
DEQUEUE_ARRAY Function, 21-10
DEQUEUE_OPTIONS_T Type, 284-9
DEQUEUE_PARALLEL_STATEMENT Procedure, 134-24
DEREGISTER Procedure, 36-18
DEREGISTER_ERROR_TRANSLATION Procedure, 156-7
DEREGISTER_SQL_TRANSATION Procedure, 156-6
DEREGSTORECOMMAND Function, 50-9
DERIVE_FROM_MVIEW function, 40-35
DESC_REC2 Record Type, 154-17
DESC_REC3 Record Type, 154-18
DESC_RESCT Record Type, 154-16
DESC_TAB Table Type, 154-21
DESC_TAB2 Table Type, 154-22
DESC_TAB3 Table Type, 154-22
DESC_TAB4 Table Type, 154-22
DESCRIBE Procedure, 116-3
DESCRIBE_COLUMNS procedure, 154-47
DESCRIBE_COLUMNS2 procedure, 154-48
DESCRIBE_COLUMNS3 Procedure, 154-49
DESCRIBE_DIMENSION procedure, 58-2
DESCRIBE_PROCEDURE procedure, 56-5
DESCRIBE_STACK procedure, 44-28
DESCRIBE_T Type, 173-24
DESTROY procedure, 131-4
DESTROY_REQUEST_CONTEXT Procedure, 265-35
DETACH_SESSION procedure, 53-19
DETACH_SIMPLE_TABLESPACE procedure, 170-16
DETACH_TABLESPACES procedure, 170-18
DFN function
  of HTF package, 215-23
  of HTP package, 216-23
  of DBMS_FLASHBACK package, 68-6
  of DBMS_OUTPUT package, 112-6
  of OWA_CACHE package, 223-2
DISABLE Procedures, 30-2
DISABLE_APPLICATION Procedure, 69-5
DISABLE_ASOF_VALID_TIME Procedure, 68-5
DISABLE_BREAKPOINT function, 53-19
DISABLE_CAPTURE Procedure, 123-5
DISABLE_CONNECTION_TEST Procedure, 18-3, 146-6
DISABLE_CREDENTIAL Procedure, 37-5
DISABLE_GROUPED_POLICY Procedure, 138-15
DISABLE_HIERARCHY procedure, 198-3
DISABLE_ILM Procedure, 83-6
DISABLE_JOB Procedure, 103-34
DISABLE_POLICY Procedure, 129-20
DISABLE_PROPAGATION_SCHEDULE Procedure, 103-35
DISABLE_PROTECTION_COLUMN Procedure, 179-6
DISABLE_PROTECTION_SOURCE Procedure, 179-7
DISABLE_PROTECTION_TYPE Procedure, 179-7
DISABLEDEFAULTTABLECREATION Procedure, 207-3
DISABLEINDEXESANDCONSTRAINTS Procedure, 208-2
DISABLEMAINTAINDOM Procedure, 207-4
DISASSOCIATE_FBA Procedure, 69-5
DISCONNECT Procedure, 54-4
DISCONNECT_SESSION procedure, 145-10
DISPLAY function, 210-6
DISPLAY_AWR function, 210-10
DISPLAY_CURSOR function, 210-12
DISPLAY_PLAN Function, 210-15
DISPLAY_SQL_PLAN_BASELINE Function, 210-18
DISPLAY_SQLSET Function, 210-19
DIST_TXN_SYNC Procedure, 189-6
DIV function
  of HTF package, 215-25
  of HTP package, 216-24
Index-23

DLISTCLOSE function
  of HTF package, 215-25
DLISTCLOSE procedure
  of HTP package, 216-25
DLISTDEF function
  of HTF package, 215-26
DLISTDEF procedure
  of HTP package, 216-25
DLISTOPEN function
  of HTF package, 215-26
DLISTOPEN procedure
  of HTP package, 216-26
DLISTTERM function
  of HTF package, 215-27
DLISTTERM procedure
  of HTP package, 216-27
DOMAIN_LEVEL Function, 109-4
DOMAINS Function, 109-5
DOMAttr methods, 200-7
DOMCDATASection methods, 200-7
DOMCharacterData methods, 200-8
DOMComment methods, 200-8
DOMDocument methods, 200-8
DOMDocumentType methods, 200-10, 200-21
DOMEntity methods, 200-21
DOMNamedNodeMap methods, 200-12
DOMText methods, 200-5
DOWNGRADE procedure, 180-2
DOWNLOAD_FILE procedures, 280-2
DROFILESYSTEM Procedures, 51-5
DROP_ACL Procedure, 108-17
DROP_ADVISOR_TASK procedure, 162-68
DROP_ALL function, 163-3
DROP_ANALYSIS_TASK Procedure, 158-5
DROP_APPLICATION Procedure, 69-6
DROP_APPLY procedure, 20-23
DROP_BASELINE procedure, 188-30
DROP_BY_CAT procedure, 111-3
DROP_CAPTURE procedure
capture process
dropping, 32-26
DROP_CAPTURE Procedure, 123-6
DROP_CHUNKS Procedure, 113-11
DROP_COMPARISON Procedure, 33-18
DROP_CREDENTIAL Procedure, 37-5
DROP_DAD Procedure, 63-7
DROP_DIAGNOSIS_TASK Procedure, 157-14
DROP_EMPTY_SEGMENTS Procedure, 151-7
DROP_EVALUATION_CONTEXT procedure,
  142-12
DROP_EVOLVE_TASK Procedure, 153-11
DROP_EXTENDED_STATS Procedure, 162-69
DROP_FILE function, 163-3
DROP_FILE_GROUP procedure, 166-14
DROP_GROUPED_POLICY Procedure, 138-15
DROP_INBOUND procedure, 212-68
DROP_LINK procedure, 182-4
DROP_MVIEW procedure, 40-7, 40-36
DROP_OBJECT procedure, 99-1
DROP_OBJECT_DEPENDENCY procedure,
  20-25
DROP_OLD_UNIFIED_AUDIT_TABLES
  Procedure, 26-16
DROP_OUTBOUND procedure, 212-68
DROP_POLICY Procedure, 129-21, 138-16,
  179-7
DROP_POLICY_CONTEXT Procedure, 138-17
DROP_POLICY_EXPRESSION Procedure,
  129-22
DROP_PROFILE Procedure, 156-8
DROP_PROPAGATION procedure, 126-7
DROP_PURGE_JOB procedure, 26-16
DROP_REWRITE_EQUIVALENCE Procedure,
  15-6
DROP_RULE procedure, 142-13
DROP_RULE_SET procedure, 142-14
DROP_SENSITIVE_COLUMN Procedure, 178-4
DROP_SENSITIVE_TYPE Procedure, 178-5
DROP_SENSITIVE_TYPE_SOURCE Procedure,
  178-5
DROP_SNAPSHOT_RANGE procedure, 188-31
DROP_SQL_PATCH Function ((amp))
  Procedure, 157-14
DROP_SQL_PLAN_BASELINE, 153-11
DROP_SQL_PLAN_DIRECTIVE Procedure,
  152-4
DROP_SQL_PROFILE procedure, 160-36
DROP_SQLSET procedure, 159-11, 160-37
DROP_STAT_TABLE procedure, 162-70
DROP_STMT_HANDLER procedure, 168-8
DROP_TASK Procedure, 113-11
DROP_TOPOLOGY procedure, 182-4
DROP_TRANSFORMATION procedure, 177-2
DROP_TUNING_TASK procedure, 160-37
DROP_UNUSED procedure, 111-4
DROP_VERSION procedure, 66-14
DROP_VIEW procedure
  of DBMS_HS_PARALLEL package, 80-3
DROP_VIRTUAL_COLUMNS procedure, 89-5
DROPPARAMETER Procedure, 202-4
DROPPREFERENCES procedure, 197-7
DROPSTORE Procedure, 50-10
DUMP_ORPHAN_KEYS procedure, 132-7
dynamic SQL
  anonymous blocks and, 154-2
evaluation flow in, 154-5
DYNAMIC_DEPTH Function, 260-6
E-mail from PL/SQL (email), 278-4
EDIT_DISTANCE Function, 271-2
EDIT_DISTANCE_SSIMILARITY Function, 271-2
EHLO function and procedure, 276-14
EM function
  of HTF package, 215-27
EM procedure
  of HTP package, 216-27
EMPHASIS function
  of HTF package, 215-28
EMPHASIS procedure
  of HTP package, 216-28
ENABLE procedure, 42-10, 112-6
ENABLE Procedures, 30-3
ENABLE_APPLICATION Procedure, 69-6
ENABLE_AT_SYSTEM_CHANGE_NUMBER procedure, 68-6
ENABLE_AT_TIME procedure, 68-6
ENABLE_AT_VALID_TIME Procedure, 69-7
ENABLE_BREAKPOINT function, 53-19
ENABLE_CAPTURE Procedure, 123-6
ENABLE_CONNECTION_TEST Procedure, 18-4, 146-6
ENABLE_CREDENTIAL Procedure, 37-6
ENABLE_ERROR_TRANSLATION Procedure, 156-9
ENABLE_GG_XSTREAM_FOR_STREAMS procedure, 212-69
ENABLE_GROUPED_POLICY Procedure, 138-17
ENABLE_HIERARCHY procedure, 198-3
ENABLE_ILM Procedure, 83-6
ENABLE_JOB Procedure, 103-35
ENABLE_POLICY Procedure, 129-22, 138-18
ENABLE_PROPAGATION_SCHEDULE Procedure, 103-36
ENABLE_PROTECTION_COLUMN Procedure, 179-9
ENABLE_PROTECTION_SOURCE Procedure, 179-9
ENABLE_PROTECTION_TYPE Procedure, 179-10
ENABLE_REG Procedure, 36-19
ENABLE_SERVICE procedure, 182-5
ENABLE_SQL_TRANSLATION Procedure, 156-9
ENABLEDEFAULTTABLECREATION Procedure, 207-5
ENABLEDIGESTAUTHENTICATION Procedure, 192-11
ENABLEINDEXESANDCONSTRAINTS Procedure, 208-4
ENABLEMAINTAINDOM Procedure, 207-5
ENCODING_DEFAULT Function, 193-2
ENCODING_ISOLATIN1 Function, 193-3
ENCODING_UTF8 Function, 193-3
ENCODING_WIN1252 Function, 193-3
ENCRYPT Function, 38-10
ENCRYPT_CAPTURE Procedure, 186-6
END_OPERATION Procedure, 155-3
END_PREPARE Procedure, 61-6
END_REPLAY_SCHEDULE Procedure, 187-12
END_REQUEST procedure, 265-36
END_RESPONSE procedure, 265-37
END_SQL_BLOCK Procedure, 134-25
END_UPGRADE Procedure, 61-6
ENDCREATE member procedure
  of ANYDATA TYPE, 281-4
  of ANYDATASET TYPE, 282-3
  of ANYTYPE TYPE, 283-4
ENQUEUE procedure, 169-4
ENQUEUE_ARRAY Function, 21-12
ENQUEUE_OPTIONS_T Type, 284-12
ENQUOTE_LITERAL Function, 25-2
ENQUOTE_NAME Function, 25-2
ENV_T Type, 173-25
EPC_TO_ORACLE_SCHEME function, 102-8
EQUALS_HOST Function, 109-5
ERASE Procedures, 93-29
error queue
  deleting errors, 20-22
  executing errors, 20-26, 20-27
  getting error messages, 20-29
ERROR_DEPTH Function, 260-7
ERROR_MSG Function, 260-7
ERROR_NUMBER Function, 260-7
ESCAPE function, 279-3
ESCAPE_SC function
  of HTF package, 215-28
ESCAPE_SC procedure
  of HTP package, 216-28
ESCAPE_URL function
  of HTF package, 215-29
ESCAPEURI function, 286-16
ESTIMATE_CPU_UNITS function, 110-1
ESTIMATE_MVIEW_SIZE Procedure, 106-3
ETINSTANCE member function
  of ANYDATASET TYPE, 282-7
EVALUATE procedure, 141-3
EVOLVE_SQL_PLAN_BASELINE Function, 153-12
EXACT_TEXT_SIGNATURES procedure, 111-4
EXCHANGEPOSTPROC Procedure, 208-5
EXCHANGEPREPROC Procedure, 208-6
EXEC_DDL_STATEMENT Procedure, 183-14
EXECUTE function, 154-50
EXECUTE member procedure, 289-2, 289-11
EXECUTE procedure, 53-20
EXECUTE_ADVISOR_TASK function, 162-70
EXECUTE_ALL_ERRORS procedure, 20-26
EXECUTE_ANALYSIS_TASK, 158-5
EXECUTE_AND_FETCH function, 154-51
EXECUTE_DIAGNOSIS_TASK Procedure, 157-14
EXECUTE_ERROR procedure, 20-27
EXECUTE_EVOLVE_TASK Function, 153-14
EXECUTE_ILM Procedure, 82-4
EXECUTE_ILM_TASK Procedure, 82-5
EXECUTE_NON_QUERY Function, 81-10
EXECUTE_TASK Procedure, 16-22
EXECUTE_TUNING_TASK Function (amp), 160-38
EXECUTE_TUNING_TASK Function and Procedure, 29-2
EXECUTE_UPDATE Procedure, 130-9
execution flow in dynamic SQL, 154-5
EXISTSNODE function, 300-5
EXISTSRESOURCE Function, 194-11
EXPAND_MESSAGE function, 144-7
EXPAND_SQL_TEXT Procedure, 183-14
EXPLAIN_SQL_TESTCASE Function, 157-15
EXPONENTIAL_DIST_FIT procedure, 161-1
EXPORT_AWR Procedure, 186-7, 187-13
EXPORT_COLUMN_STATS procedure, 162-72
EXPORT_DATABASE_PREFS Procedure, 162-73
EXPORT_DATABASE_STATS procedure, 162-74
EXPORT_DICTIONARY_STATS procedure, 162-75
EXPORT_FIXED_OBJECTS_STATS procedure, 162-76
EXPORT_INDEX_STATS procedure, 162-76
EXPORT_PENDING_STATS Procedure, 162-77
EXPORT_PROFILE Procedure, 156-10
EXPORT_SCHEMA_PREFS Procedure, 162-78
EXPORT_SCHEMA_STATS procedure, 162-79
EXPORT_SQL_TESTCASE Procedures, 157-15
EXPORT_SQL_TESTCASE_DIR_BY_INC Function, 157-19
EXPORT_SQL_TESTCASE_DIR_BY_TXT Function, 157-20
EXPORT_SYSTEM_STATS procedure, 162-80
EXPORT_TABLE_PREFS Procedure, 162-81
EXPORT_TABLE_STATS procedure, 162-81
EXPORT_XML procedure, 40-38, 40-40
EXPORTXMLTOFILE procedure, 40-12
EXPORTTRMANBACKUP Procedure, 116-4
EXTEND_MAPPINGS Procedure, 69-7
EXTENT_HEAT_MAP Function, 76-7
EXTRACT function, 300-5

F

FASTSTART_DISABLE procedure, 85-6
FASTSTART_ENABLE procedure, 85-7
FASTSTART_MIGRATE_STORAGE procedure, 85-9
FCLOSE procedure, 264-7
FCLOSE_ALL procedure, 264-8
FCOPY procedure, 264-8
FEATURE_OPTIONS Table Type, 179-2
FEATURE_T Record Type, 48-17
FEATURENAME Function, 48-33
FEATURES_T Table Type, 48-22
FETCH_ROW Function, 81-10
FETCH_ROWS function, 154-51
FFLUSH procedure, 264-9
FGETATTR procedure, 264-10
FGETPOS function, 264-10
FI_HORIZONTAL function, 70-1
FI_TRANSACTIONAL function, 70-3
file groups, 66-1
adding files, 66-4
altering, 66-8
altering files, 66-6
altering versions, 66-10
creating, 66-11
creating versions, 66-12
dropping, 66-14
dropping versions, 66-14
granting object privileges, 66-15
granting system privileges, 66-16
purging, 66-17
removing files, 66-17
revoking object privileges, 66-18
revoking system privileges, 66-19
FILE type, 170-2, 170-3
FILE_SET type, 170-2, 170-3
FILECLOSE Procedure, 93-31
FILECLOSEALL Procedure, 93-32
FILEEXISTS Function, 93-32
FILEGETNAME Procedure, 93-33
FILEISOPEN Function, 93-34
FILEOPEN Procedure, 93-35
FILETYPE Record Type, 264-5
FIND_AFFECTED_TABLE Procedure, 61-6
FIDENTITY function, 200-29
FINDNOTATION function, 200-30
FINISH_CAPTURE Procedure, 186-7
FINISH_REDEF_TABLE procedure, 130-9
FINISH_REPORT_CAPTURE Procedure, 28-2
FIX_CORRUPT_BLOCKS procedure, 132-8
FLUSH Function (amp) Procedure, 136-8
FLUSH procedure, 42-12, 278-7
FLUSH_DATA function and procedure, 125-6
FLUSH_DATABASE_MONITORING_INFO procedure, 162-83
FLUSH_SQL_PLAN_DIRECTIVE Procedure, 152-4
FLUSH_UNIFIED_AUDIT_TRAIL procedure, 26-17
FLUSHCACHE Procedure, 50-11
FONTCLOSE function of HTF package, 215-30
FONTCLOSE procedure of HTP package, 216-29
FONTOPEN function of HTF package, 215-30
FONTOPEN procedure of HTP package, 216-29
FOPEN function, 264-11
FOPEN_NCHAR function, 264-13
FORCE parameter and job-to-instance affinity, 88-2
FORMAT function, 296-7
FORMAT_CALL_STACK Function, 183-15
FORMAT_CELL function of HTF package, 215-31
FORMAT_ERROR_BACKTRACE Function, 183-15
FORMAT_ERROR_STACK Function, 183-18
FORMCHECKBOX function of HTF package, 215-31
FORMCHECKBOX procedure of HTP package, 216-30
FORMCLOSE function of HTF package, 215-32
FORMCLOSE procedure of HTP package, 216-30
FORMFILE function of HTF package, 215-32
FORMFILE procedure of HTP package, 216-31
FORMHIDDEN function of HTF package, 215-33
FORMHIDDEN procedure of HTP package, 216-32
FORMIMAGE function of HTF package, 215-33
FORMIMAGE procedure of HTP package, 216-32
FORMOPEN function of HTF package, 215-34
FORMOPEN procedure of HTP package, 216-31
FORMPASSWORD function of HTF package, 215-35
FORMPASSWORD procedure of HTP package, 216-33
FRAME procedure  
of HTP package, 216-41
FRAMESETCLOSE function  
of HTF package, 215-44
FRAMESETCLOSE procedure  
of HTP package, 216-42
FRAMESETOPEN function  
of HTF package, 215-45
FRAMESETOPEN procedure  
of HTP package, 216-42
FREE_BLOCKS procedure, 150-7
FREE_UNUSED_USER_MEMORY Procedure, 146-7
FREEDOCUMENT procedure, 200-31
FREENODE procedure, 200-32
FREEPROCESSOR procedure, 211-4
FREERESOURCES Procedure, 196-4
FREESTYLESHEET procedure, 211-4
FREETEMPORARY Procedures, 93-40
FREMOVE procedure, 264-14
FRENAMe procedure, 264-14
FSEEK procedure, 264-15
G
GATHER_DATABASE_STATS procedures, 162-83
GATHER_DICTIONARY_STATS procedure, 162-88
GATHER_FIXED_OBJECTS_STATS procedure, 162-92
GATHER_INDEX_STATS Procedure, 162-95
GATHER_SCHEMA_STATS procedures, 162-95
GATHER_SYSTEM_STATS procedure, 162-100
GATHER_TABLE_STATS procedure, 162-103
GENERATE_CAPTURE_SUBSET Procedure, 187-13
GENERATE_RESULT Procedure, 123-7
GENERATE_STATS procedure, 162-110
GET function  
of OWA_COOKIE package, 224-2
GET_AUDIT_TRAIL_PROPERTY_VALUE Function, 26-18
GET_AVC_OID function, 198-4
GET_Advisory_OPR_FILTER procedure, 162-108
GET_ADVISOR_RECS function, 162-109
GET_ALL procedure, 224-2
GET_ALL_DAD_ATTRIBUTES Procedure, 63-8
GET_ALL_DAD_MAPPINGS Procedure, 63-8
GET_ALL_GLOBAL_ATTRIBUTES Procedure, 63-9
GET_ALL_NAMES member function, 297-5
GET_ASH_QUERY Function, 14-11
GET_AUDIT_COMMIT_DELAY function, 26-18
GET_AUTHENTICATION procedure, 265-37
GET_BASE_TABLE_NAME member function, 289-2
GET_BASE_TABLE_OWNER member function, 289-2
GET_BODY_CHARSET procedure, 265-38
GET_CAPTURE_INFO Function, 186-8
GET_CATEGORY function, 184-3
GET_CATEGORY_ID function, 102-11
GET_CGI_ENV function, 231-8
GET_CLIENT_ATTRIBUTES Procedure, 30-4
GET_CLIENT_HOSTNAME function, 229-2
GET_CLIENT_IP function, 229-2
GET_COLO procedure, 173-32
GET_COLUMN_STATS procedures, 162-110
GET_COMMAND_TYPE member function, 289-35
GET_COMMIT_SCN member function, 289-35
GET_COMMIT_SCN_FROM_POSITION static function, 289-35
GET_COMPATIBLE member function, 289-35
GET_COMPONENT function, 296-8
GET_COMPONENTS function, 102-12
GET_COMPRESSION_RATIO Procedure, 34-5
GET_COMPRESSION_TYPE Function, 34-6
GET_COOKIE_COUNT function, 265-39
GET_COOKIE_SUPPORT procedure, 265-39
GET_COOKIE function, 265-40
GET_CPU_TIME Function, 183-18
GET_CURRENT_SCHEMA member function, 289-2
GET_DAD_ATTRIBUTE Function, 63-10
GET_DAD_LIST Procedure, 63-10
GET_DBFS_LINK Functions, 93-41
GET_DBFS_LINK_STATE Procedure, 93-42
GET_DEPENDENCY Procedure, 183-19
GET_DETAILED_EXCP_SUPPORT procedure, 265-40
GET_DETAILED_SQLCODE function, 265-41
GET_DETAILED_SQLERRM function, 265-41
GET_DIVerging_STATEMENT Function, 187-14
GET_EDITION_NAME member function, 289-2
GET_ENCODINGS function, 102-13
GET_EngINness Function, 183-19
GET_ERROR_MESSAGE function, 20-29
GET_ETAG function, 223-2
GET_EXPRESSION function, 44-30
GET_EXTRA_ATTRIBUTE member function, 289-35
GET_FASTSTART_TABLESPACE function, 85-10
GET_FILE procedure, 67-4
GET_FIX_CONTROL Function, 157-21
GET_FOLLOW_REDIRECT procedure, 265-41
GET_GLOBAL_ATTRIBUTE Function, 63-11
GET_HASH_VALUE Function, 183-20
GET_HEADER procedure, 265-42
GET_HEADER_BY_NAME procedure, 265-43
GET_HEADER_COUNT function, 265-43
GET_HOST_ADDRESS function, 268-2
GET_HOST_NAME function, 268-3
GET_INDEX_DATAGUIDE Function, 89-6
GET_INDEX_STATS procedures, 162-112
GET_INDEXES function, 53-22
GET_INFORMATION function, 164-9
GET_JAVA_LOGGING_LEVEL function, 102-14
GET_LAST_ARCHIVE_TIMESTAMP Function, 26-19
GET_LEVEL function, 223-2
GET_LINE function, 278-8
GET_LINE procedure, 112-7, 264-16
GET_LINE_MAP function, 53-23
GET_LINE_NCHAR Function, 278-9
GET_LINES procedure, 112-8
GET_LOB_INFORMATION member function, 289-11
GET_LOB_OPERATION_SIZE member function, 289-11
GET_LOG procedure, 42-12
GET_LOG_SPEC function, 42-14
GET_LOGON_USER member function, 289-2
GET_LONG_INFORMATION member function, 289-11
GET_LTIXID_OUTCOME Procedure, 18-5
GET_MAX_STREAMS_POOL Procedure, 22-30
GET_MESSAGE_TRACKING function, 165-80, 212-70
GET_MIN_STREAMS_POOL Procedure, 22-30
GET_MORE_SOURCE procedure, 53-23
GET_NEXT_HIT function, 141-10, 141-11
GET_NEXT_RESULT Procedures, 154-52
GET_NLS_PARAMETER Function, 54-5
GET_NODE_ID_LOCAL function, 182-5
GET_NODE_NAME_LOCAL function, 182-6
GET_NUMBER_COL_CHUNK Procedure, 113-12
GET_OBJECT_NAME member function, 289-35
GET_OBJECT_OWNER member function, 289-35
GET_OBJECT_TYPE member function, 289-35
GET_OPATCH_BUGS Function, 127-3
GET_OPATCH_COUNT Function, 127-4
GET_OPATCH_DATA Function, 127-4
GET_OPATCH_FILES Function, 127-5
GET_OPATCH_INSTALL_INFO Function, 127-5
GET_OPATCH_LIST Function, 127-5
GET_OPATCH_LSINVENTORY, 127-5
GET_OPATCH_OLAYS Function, 127-6
GET_OPATCH_PREQS Function, 127-6
GET_OPATCH_XSLT, 127-6
GET_OWA_SERVICE_PATH function, 231-8
GET_P1_RESOURCES Procedure, 30-4
GET_PACKAGE_MEMORY_UTILIZATION Procedure, 146-9
GET_PARAM function, 162-114
GET_PARAMETER function, 42-14
GET_PARAMETER_VALUE Function, 183-20
GET_PASSWORD function, 229-2
GET_PATCHDETAILS Function, 127-6
GET_PERSISTENT_CONN_COUNT function, 265-44
GET_PERSISTENT_CONN_SUPPORT Procedure, 265-45
GET_PERSISTENT_CONNS procedure, 265-45
GET_PLSQL_LOGGING_LEVEL function, 102-15
GET_PLSQL_TRACE_LEVEL function, 175-6
GET_PERSISTENT_CONN_COUNT function, 265-44
GET_PERSISTENT_CONN_SUPPORT Procedure, 265-45
GET_PERSISTENT_CONNS procedure, 265-45
GET_PROXY procedure, 265-46
GET_RAW function, 264-18, 278-9
GET_REC_ATTRIBUTES Procedure, 16-23
GET_REPLAY_DIRECTORY Function, 187-15
GET_REPLAY_INFO Function, 187-16
GET_REPLAY_TIMEOUT Procedure, 187-16
GET_REPORT Function, 14-11
GET_RESPONSE function, 265-46
GET_RESPONSE_ERROR_CHECK procedure, 265-48
GET_ROW_SET procedure, 173-34
GET_ROW_TEXT member procedure, 289-11
GET_SESSION_TIMEOUT function, 289-35
GET_SESSION_TIMEOUT function, 137-2
GET_SIAD_API function, 151-7
GET_SIADATTRIB Procedure, 151-7
GET_SIADATTRIB Procedure, 151-7
GET_SIADATTRIB Procedure, 151-7
GET_SIADATTRIB Procedure, 151-7
GET_SIADATTRIB Procedure, 151-7
GET_SIADATTRIB Procedure, 151-7
GET_SIADATTRIB Procedure, 151-7
GETSTANDALONE function, 200-55
GETSTATS Procedure, 48-43
GETSTOREBEMY_MOUNT Function, 48-41
GETSTOREBEMY_NAME Function, 48-42
GETSTOREBEMY_PATH Function, 48-42
GETSTOREID Function, 49-13
GETSTOREPROPERTY Function, 50-43
GETSTRINGVAL function, 300-7
GETSUCCESSORS function, 195-5
GETSUCCSBYRESID function, 195-6
GETSYSTEMID Function, 200-55
GETTAGNAME function, 200-56
GETTARGET function, 200-50
GETTOKENTABLEINFO Procedure ((amp)) Function, 39-2
GETTOKENTABLEINFOBYTABLESPACE Procedure, 39-3
GETTRACE Function, 48-43
GETTYPE member function
of ANYDATA TYPE, 281-7
of ANYDATASET TYPE, 282-7
GETTYPENAME member function
of ANYDATA TYPE, 281-7
of ANYDATASET TYPE, 282-8
GETUPDATEBYTECOUNT Function, 199-22
GETUPDATEBYTEOFFSET Function, 199-23
GETURL function, 286-4, 286-11, 286-14, 286-16
GETVALIDATIONMODE function, 203-4
GETVALUE function, 200-56
GETVERSION function, 200-57
GETVERSION Function, 48-43, 49-13
GETVERSION procedure, 204-6
GETVERSIONID Function, 196-11
GETXDB_TABLESPACE Function, 194-15
GETXDBEVENT Function, 199-23
GETXML function, 201-4, 204-6, 286-4, 286-7, 286-11, 286-14
GETXML procedure, 204-6
GETXMLTYPE function, 200-57, 201-5
GG_PROCEDURE_REPLICATION_ON function, 72-9
global ports, 192-13
GoldenGate
privileges, 73-1
GoldenGate administrator, 73-1
GRANT_ADMIN_PRIVILEGE procedure, 73-2, 167-2, 213-2
GRANT_OBJECT_PRIVILEGE procedure, 142-14
GRANT_REMOTE_ADMIN_ACCESS procedure, 167-5, 213-6
GRANT_SWITCH_CONSUMER_GROUP procedure, 135-1
GRANT_SYSTEM_PRIVILEGE procedure, 66-16, 135-2, 142-16
GRANTING_OBJECT_PRIVILEGE procedure, 66-15

H
HANDLE_COLLISIONS procedure, 20-31
HASACLCHANGED Function, 196-11
HASATTRIBUTE function, 200-58
HASAUTHORCHANGED Function, 196-12
HASBLOBCONTENT Function, 194-15
HASCHANGED Function, 196-12
HASCHARACTERSETCHANGED Function, 196-12
HASCHARCONTENT Function, 194-16
HASCHILDNODES function, 200-59
HASCOMMENTCHANGED Function, 196-13
HASCONTENTCHANGED Function, 196-13
HASCONTENTTYPECHANGED Function, 196-14
HASCREATIONDATECHANGED Function, 196-14
HASCREATORDATECHANGED Function, 196-14
HASCUSTOMMETADATA_CHANGED Function, 196-15
HASDISPLAYNAMECHANGED Function, 196-15
HASFEATURE function, 200-60
HASH function, 38-12
HASLANGUAGECHANGED Function, 196-16
HASLASTMODIFIERCHANGED Function, 196-16
HASMODIFICATIONDATECHANGED Function, 196-16
HASOWNERCHANGED Function, 196-17
HASREFCOUNTCHANGED Function, 196-17
HASVERSIONIDCHANGED Function, 196-18
HASXMLCONTENT Function, 194-16
HASXMLREFERENCE Function, 194-16
HEADCLOSE function
of HTF package, 215-45
HEADCLOSE procedure
of HTP package, 216-43
HEADER function
of HTF package, 215-46
HEADER procedure
of HTP package, 216-43
HEADOPEN function
of HTF package, 215-46
HEADOPEN procedure
of HTP package, 216-44
HELO function and procedure, 276-15
HELP function, 276-15
hierarchies
validation log table, 77-2
verify validation, 77-5
HR function
of HTF package, 215-47
HR procedure
of HTP package, 216-44
HTML tags
applet tags
functions, 215-2
procedures, 216-2
atags tags
procedures, 216-2
character formatting tags
functions, 215-2
procedures, 216-2
form tags
functions, 215-2
procedures, 216-2
frame tags
functions, 215-2
procedures, 216-2
list tags
functions, 215-2
procedures, 216-2
paragraph formatting tags
functions, 215-2
procedures, 216-2
table tags
functions, 215-2
HTMLCLOSE function
of HTF package, 215-47
HTMLCLOSE procedure
of HTP package, 216-45
HTMLOPEN function
of HTF package, 215-47
HTMLOPEN procedure
of HTP package, 216-45
HTP procedures, 216-1
HTTP_HEADER_CLOSE procedure, 231-9
HttpUriType, 286-4
HTTPURITYPE function, 286-8
HttpUriType subtype, 286-4
CREATEURI function, 286-5
GETBLOB function, 286-5
GETCLOB function, 286-6
GETCONTENTTYPE function, 286-7
GETEXTERNALURL function, 286-7
GETRUL function, 286-7
GETXML function, 286-7
HTTPURITYPE function, 286-8
methods, 286-4

IM FastStart (continued)
security, 85-2
IME_CAPTURE_EXPRESSIONS procedure, 85-11
IME_CLOSE_CAPTURE_WINDOW procedure, 85-1
IME_DROP_ALL_EXPRESSIONS procedure, 85-13
IME_DROP_EXPRESSIONS procedure, 85-14
IME_OPEN_CAPTURE_WINDOW procedure, 85-15
IME_POPULATE_EXPRESSIONS procedure, 85-16
IMG function
of HTF package, 215-48
IMG procedure
of HTP package, 216-45
IMG2 procedure
of HTP package, 216-46
IMPLEMENT_ADVISOR_TASK function, 162-128
IMPLEMENT_EVOLVE_TASK Procedure, 153-14
IMPLEMENT_TASK Procedure, 16-27
IMPLEMENT_TUNING_TASK procedure, 160-39
IMPORT_AWR Function, 187-16
IMPORT_AWR Procedure, 186-8
IMPORT_COLUMN_STATS procedure, 162-130
IMPORT_DATABASE_PREFS Procedure, 162-131
IMPORT_DATABASE_STATS procedure, 162-132
IMPORT_DICTIONARY_STATS procedure, 162-133
IMPORT_DISCOVERY_RESULT Procedure, 178-6
IMPORT_FIXED_OBJECTS_STATS procedure, 162-135
IMPORT_HISTORY Procedure, 69-8
IMPORT_INDEX_STATS procedure, 162-136
IMPORT_PROFILE Procedure, 156-12
IMPORT_SCHEMA_PREFS Procedure, 162-137
IMPORT_SCHEMA_STATS procedure, 162-138
IMPORT_SENSITIVE_TYPES Procedure, 178-7
IMPORT_SQL_TESTCASE Procedures, 157-22
IMPORT_SQLWKLD_SCHEMA Procedure, 16-27
IMPORT_SQLWKLD_SQLCACHE Procedure, 16-29
IMPORT_SQLWKLD_STS Procedure, 16-30
IMPORT_SQLWKLD_SUMADV Procedure, 16-32
IMPORT_SQLWKLD_USER Procedure, 16-34
IMPORT_SYSTEM_STATS procedure, 162-139
IMPORT_TABLE_PREFS Procedure, 162-140
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPORT_TABLE_STATS procedure</td>
<td>162-140</td>
</tr>
<tr>
<td>IMPORT_XML procedure</td>
<td>40-12, 40-42</td>
</tr>
<tr>
<td>IMPORTNODE function</td>
<td>200-60</td>
</tr>
<tr>
<td>INCDBUS_STREAMS_ADM package</td>
<td>165-2</td>
</tr>
<tr>
<td>INCDBUS_STREAMS_ADM rules</td>
<td>165-7</td>
</tr>
<tr>
<td>INCIDENTID_2_SQL Procedure</td>
<td>157-24</td>
</tr>
<tr>
<td>INCLUDE_EXTRA_ATTRIBUTES procedure</td>
<td>32-28</td>
</tr>
<tr>
<td>INDEX_TABLE_TYPE Table Type</td>
<td>183-3</td>
</tr>
<tr>
<td>INDEXXMLREFERENCES Procedure</td>
<td>208-6</td>
</tr>
<tr>
<td>INIT_CLEANUP procedure</td>
<td>26-20</td>
</tr>
<tr>
<td>INITFS Procedure</td>
<td>51-6</td>
</tr>
<tr>
<td>INITIALIZE function</td>
<td>53-27</td>
</tr>
<tr>
<td>INITIALIZE procedure</td>
<td>128-2</td>
</tr>
<tr>
<td>INITIALIZE_CONSOLIDATED_REPLAY Procedure</td>
<td>187-17</td>
</tr>
<tr>
<td>INITIALIZE_CUBE_UPGRADE procedure</td>
<td>40-11, 40-44</td>
</tr>
<tr>
<td>INITIALIZE_REPLAY Procedure</td>
<td>187-18</td>
</tr>
<tr>
<td>INPLACE_EVOLVE Procedure</td>
<td>206-9</td>
</tr>
<tr>
<td>INSERT_AUTOBIN_NUM_EQWIDTH procedure</td>
<td>44-31</td>
</tr>
<tr>
<td>INSERT_BIN_CAT_FREQ procedure</td>
<td>44-35</td>
</tr>
<tr>
<td>INSERT_BIN_NUM_EQWIDTH procedure</td>
<td>44-39</td>
</tr>
<tr>
<td>INSERT_BIN_NUM_QTILE procedure</td>
<td>44-43</td>
</tr>
<tr>
<td>INSERT_BIN_SUPER procedure</td>
<td>44-45</td>
</tr>
<tr>
<td>INSERT_CLIP_TRIM_TAIL procedure</td>
<td>44-49</td>
</tr>
<tr>
<td>INSERT_CLIP_WINSOR_TAIL procedure</td>
<td>44-52</td>
</tr>
<tr>
<td>INSERT_FINDING_DIRECTIVE Procedure</td>
<td>14-12</td>
</tr>
<tr>
<td>INSERT_MISS_CAT_MODE procedure</td>
<td>44-55</td>
</tr>
<tr>
<td>INSERT_MISS_NUM_MEAN procedure</td>
<td>44-57</td>
</tr>
<tr>
<td>INSERT_NORM_LIN_MINMAX procedure</td>
<td>44-61</td>
</tr>
<tr>
<td>INSERT_NORM_LIN_SCALE procedure</td>
<td>44-64</td>
</tr>
<tr>
<td>INSERT_NORM_LIN_ZSCORE procedure</td>
<td>44-64</td>
</tr>
<tr>
<td>INSERT_PARAMETER_DIRECTIVE Procedure</td>
<td>14-13</td>
</tr>
<tr>
<td>INSERT_PROCESS_EXCLUSION_OBJ procedure</td>
<td>72-9</td>
</tr>
<tr>
<td>INSERT_SEGMENT_DIRECTIVE Procedure</td>
<td>14-13</td>
</tr>
<tr>
<td>INSERT_SQL_DIRECTIVE Procedure</td>
<td>14-15</td>
</tr>
<tr>
<td>INSERTBEFORE function</td>
<td>200-61</td>
</tr>
<tr>
<td>INSERTDATA procedure</td>
<td>200-62</td>
</tr>
<tr>
<td>INSERTXML function</td>
<td>205-6, 209-4</td>
</tr>
<tr>
<td>INSTALL_DEFAULTWALLET Function</td>
<td>191-2</td>
</tr>
<tr>
<td>INSTANCE procedure</td>
<td>88-5</td>
</tr>
<tr>
<td>INSTANCE_RECORD Record Type</td>
<td>183-3</td>
</tr>
<tr>
<td>INSTANCE_TABLE Table Type</td>
<td>183-4</td>
</tr>
<tr>
<td>INVALIDATE Functions ((amp)) Procedures</td>
<td>136-6</td>
</tr>
<tr>
<td>INVALIDATE Procedure</td>
<td>183-23</td>
</tr>
<tr>
<td>INVALIDATE_OBJECT Functions ((amp)) Procedures</td>
<td>136-6</td>
</tr>
<tr>
<td>IP_ADD_ACE procedure</td>
<td>147-3</td>
</tr>
<tr>
<td>IP_ADD_PDB_ACE procedure</td>
<td>147-4</td>
</tr>
<tr>
<td>IP_REMOVE_ACE procedure</td>
<td>147-4</td>
</tr>
<tr>
<td>IP_REMOVE_ACL procedure</td>
<td>147-5</td>
</tr>
<tr>
<td>IP_REMOVE_PDB_ACE procedure</td>
<td>147-5</td>
</tr>
<tr>
<td>IP_REMOVE_PDB_ACL procedure</td>
<td>147-6</td>
</tr>
<tr>
<td>IS_BIT_SET Function</td>
<td>183-26</td>
</tr>
<tr>
<td>IS_CLEANUP_INITIALIZED function</td>
<td>26-22</td>
</tr>
<tr>
<td>IS_CLUSTER_DATABASE Function</td>
<td>183-26</td>
</tr>
<tr>
<td>IS_FAST Procedure</td>
<td>141-11</td>
</tr>
<tr>
<td>IS_GG_XSTREAM FOR_STREAMS function</td>
<td>212-71</td>
</tr>
<tr>
<td>IS_HIERARCHY_ENABLED function</td>
<td>198-5</td>
</tr>
<tr>
<td>IS_LOCATOR function</td>
<td>261-1</td>
</tr>
<tr>
<td>IS_NULL_TAG member function</td>
<td>289-35</td>
</tr>
<tr>
<td>IS_OPEN function</td>
<td>154-53, 264-19</td>
</tr>
<tr>
<td>IS_PATCH_INSTALLED Function</td>
<td>127-7</td>
</tr>
<tr>
<td>IS_REPLAY_PAUSED Procedure</td>
<td>187-20</td>
</tr>
<tr>
<td>IS_ROLE_ENABLED Function</td>
<td>146-10</td>
</tr>
<tr>
<td>IS_SESSION_ALIVE Function</td>
<td>146-10</td>
</tr>
<tr>
<td>IS_TRIGGER_FIRE_ONCE function</td>
<td>52-6</td>
</tr>
<tr>
<td>IS_FOLDER Function</td>
<td>194-17, 196-18</td>
</tr>
<tr>
<td>IS_GLOBALPORTENABLED Function</td>
<td>192-13</td>
</tr>
</tbody>
</table>
ISINDEX function
  of HTF package, 215-49
ISINDEX procedure
  of HTP package, 216-47
ISNULL function, 200-62
ISNULL Function, 199-24
ISOPEN function, 262-2
ISOPEN Functions, 93-49
ISPATHLOCKED Procedure, 48-44, 49-14
ISREMOTE Function, 93-50
ISSCHEMABASED function, 300-8
ISSCHEMAVALID function, 300-8
ISSCHEMAVALIDATED function, 300-9
ISTEMPORARY Functions, 93-52
ITALIC function
  of HTF package, 215-50
ITALIC procedure
  of HTP package, 216-47
ITEM Functions, 200-65

L
LAPACK Driver Routines (Linear Equations)
  Subprograms, 272-5
LAPACK Driver Routines (LLS and Eigenvalue Problems), 272-6
LAPACK_GBSV Procedures, 272-62
LAPACK_GEES Procedures, 272-64
LAPACK_GEEV Procedures, 272-74
LAPACK_GELS Procedures, 272-66
LAPACK_GESDD Procedures, 272-68
LAPACK_GESV Procedures, 272-70
LAPACK_GESVD Procedures, 272-71
LAPACK_GTSV Procedures, 272-76
LAPACK_PBSV Procedures, 272-78
LAPACK_POSV Procedures, 272-79
LAPACK_PPSV Procedures, 272-81
LAPACK_PTSV Procedures, 272-83
LAPACK_SBEV Procedures, 272-84
LAPACK_SBEVD Procedures, 272-86
LAPACK_SPEV Procedures, 272-88
LAPACK_SPEVD Procedures, 272-89
LAPACK_SPSV Procedures, 272-91
LAPACK_STEV Procedures, 272-94
LAPACK_STEVD Procedures, 272-95
LAPACK_SYEV Procedures, 272-96
LAPACK_SYEVD Procedures, 272-98
LAPACK_SYSV Procedures, 272-99
LAST_ERROR_POSITION function, 154-54
LAST_ROW_COUNT function, 154-54
LAST_ROW_ID function, 154-55
LAST_SQL_FUNCTION_CODE function, 154-55
LCR_TO_XML Function, 104-19
LCR$_DDL_RECORD type, 289-2
LCR$_ROW_LIST type, 289-47
LCR$_ROW_RECORD type, 289-11
LCR$_ROW_UNIT type, 289-48
GET_LOB_INFORMATION member function, 289-11
GET_LOB_OPERATION_SIZE member procedure, 289-11
GET_LONG_INFORMATION member function, 289-11
GET_POSITION member function, 289-35
GET_ROOT_NAME member function, 289-35
GET_ROW_TEXT member procedure, 289-11
GET_WHERE_CLAUSE member procedure, 289-11
SET_LOB_INFORMATION member procedure, 289-11
SET_LOB_OPERATION_SIZE member procedure, 289-11
LENGTH function, 273-18
LEXICAL_DEPTH Function, 260-8
LINE function
  of HTF package, 215-51
LINE procedure
  of HTP package, 216-49
LINK Procedures, 194-17
LINKREL function
  of HTF package, 215-52
log apply services
managing initialization parameters for logical
standby databases, 97-1

logical change records (LCRs)
DDL LCRs, 289-2
determining if tag is NULL, 289-35
executing, 289-2, 289-11
extra attributes
excluding, 32-28
including, 32-28
GET_THREAD_NUMBER member function, 289-35
getting command type, 289-35
getting commit SCN, 289-35
getting commit SCN from position, 289-35
getting commit time, 289-35
getting compatibility information, 289-35
getting extra attributes, 289-35
getting LCR creation time, 289-35
getting object name, 289-35
getting object owner, 289-35
getting SCN, 289-35
getting SCN from position, 289-35
getting source database name, 289-35
getting tag, 289-35
getting transaction identifier, 289-35
LCR$_DDL_RECORD type, 289-2
LCR$_ROW_LIST type, 289-47
LCR$_ROW_RECORD type, 289-11
LCR$_ROW_UNIT type, 289-48
row LCRs, 289-11
setting command type, 289-35
setting extra attributes, 289-35
setting object name, 289-35
setting object owner, 289-35
setting source database name, 289-35
setting tag, 289-35
tracking, 165-80, 165-163, 212-70
types, 289-1

logical standby databases
managing with DBMS_LOGSTDBY package, 97-1

logs
Cube Build, 42-4
Cube Dimension Compile, 42-6
Cube Operations, 42-6
Cube Rejected Records, 42-7
LZ_COMPRESS functions and procedures, 262-3
LZ_COMPRESS_ADD procedure, 262-4
LZ_COMPRESS_CLOSE procedure, 262-5
LZ_COMPRESS_OPEN function, 262-5
LZ_UNCOMPRESS functions and procedures, 262-6
LZ_UNCOMPRESS_CLOSE procedure, 262-8
LZ_UNCOMPRESS_EXTRACT procedure, 262-7
LZ_UNCOMPRESS_OPEN function, 262-7

machine learning
  transactional data, 43-12
MAIL function and procedure, 276-16
MAILTO function
  of HTF package, 215-55
MAILTO procedure
  of HTP package, 216-52
MAINTAIN_CHANGE_TABLE procedure, 165-85
MAINTAIN_GLOBAL procedure, 165-93
MAINTAIN_SCHEMAS procedure, 165-98
MAINTAIN_SIMPLE_TABLESPACE procedure, 165-104
MAINTAIN_SIMPLE_TTS procedure, 165-109
MAINTAIN_TABLES procedure, 165-125
MAINTAIN_TABLESPACES procedure, 165-118
MAINTAIN_TTS procedure, 165-125
MAKE procedure, 131-4
MAKE_DATA_BLOCK_ADDRESS Function, 183-26
MAKEATTR function, 200-66
MAKECDATASECTION function, 200-66
MAKECHARACTERDATA function, 200-67
MAKECOMMENT function, 200-67
MAKEDOCUMENT function, 200-68
MAKEDOCUMENT Function, 196-19
MAKEDOCUMENTFRAGMENT function, 200-68
MAKEDOCUMENTTYPE function, 200-69
MAKEELEMENT function, 200-69
MAKEENTITY function, 200-70
MAKEENTITYREFERENCE function, 200-70
MAKENODE function, 200-71
MAKENOTATION function, 200-73
MAKEPROCESSINGINSTRUCTION function, 200-73
MAKETEXT function, 200-74
MAKEVERSIONED function, 195-6
MAP_ALL function, 163-4
MAP_DAD Procedure, 63-11
MAP_ELEMENT function, 163-5
MAP_FILE function, 163-5
MAP_OBJECT function, 163-6
MAPCLOSE function
  of HTF package, 215-56
MAPCLOSE procedure
  of HTP package, 216-53
MAPOPEN function
  of HTF package, 215-56
MAPOpen procedure
  of HTF package, 216-53
MARK_RECOMMANDATION Procedure, 16-36
MARKHOT Procedure, 149-4
MATCH function, 228-7
MATERIALIZE_DEFERRED_SEGMENTS Procedure, 151-8
materialized view logs
  master table purging, 106-7, 106-8
materialized views
  converting, 33-1
  optimizing from table- to cube-organized, 40-30
  refreshing, 106-9, 106-11, 106-12
materialized views (cube organized), 40-2
MEMORY_REPORT Procedure, 136-7
MENULISTCLOSE function
  of HTF package, 215-57
MENULISTCLOSE procedure
  of HTP package, 216-54
MENULISTOPEN function
  of HTF package, 215-57
MENULISTOPEN procedure
  of HTP package, 216-54
merge streams, 165-129, 165-132, 212-74, 212-77
MERGE_COL_USAGE Procedure, 162-146
MERGE_STREAMS procedure, 165-129, 212-74
MERGE_STREAMS_JOB procedure, 165-132, 212-77
MESSAGE_PROPERTIES_ARRAY_T Type, 284-19
MESSAGE_PROPERTIES_T Type, 284-16
messaging client
  messaging client user, 165-2
  rules
    for LCRs, 165-2
    for user messages, 165-7
META function
  of HTF package, 215-57
META procedure
  of HTP package, 216-55
MG2 function
  of HTF package, 215-49
MGD_ID constructor function, 296-4
MGD_ID object type, 296-3
  FORMAT function, 296-7
  GET_COMPONENT function, 296-8
  MGD_ID constructor function, 296-4
  TO_STRING function, 296-9
  TRANSLATE function, 296-10
MGD_ID object types, 296-1
MGD_ID_COMPONENT object type, 296-2
MGD_ID_COMPONENT_VARRAY object type, 296-2
MIGRATE_STORED_OUTLINE Function, 153-19
MIME_HEADER procedure, 231-10
MIMEHEADER_Decode function, 263-3
MIMEHEADER_ENCODE function, 263-4
model transparency, 44-7
MODIFY_BASELINE_WINDOW_SIZE Procedure, 188-32
MODIFY_PACKAGE_STATE Procedure, 146-12
MODIFY_SERVICE Procedure, 145-11
MODIFY_SNAPSHOT_SETTINGS procedure, 188-33
MODIFY_TRANSFORMATION procedure, 177-3
MODIFYPARAMETER Procedure, 202-4
MOUNT_T Record Type, 48-17
MOUNTS_T Table Type, 48-22
MOUNTSTORE Procedure, 48-47
MOVE_TO_DBFS_LINK Procedures, 93-60
MOVEXDB_TABLESPACE Procedure, 191-2
MSGID_ARRAY_T Type, 284-20
MV_CUBE_ADVICE function, 41-1

NAME_ARRAY Table Type, 183-4
NAME_resolve Procedure, 183-27
NAME_TOKENIZE Procedure, 183-28
NAMELIST Table Type, 153-3
NAMESPACE function, 300-7
NAMESPACE_ACL Function, 193-3
NAMESPACE_METADATA Function, 193-4
NAMESPACE_ORACLE Function, 193-4
NAMESPACE_RESOURCE Function, 193-4
NAMESPACE_RESOURCE_CONFIG Function, 193-5
NAMESPACE_RESOURCE_EVENT Function, 193-5
NAMESPACE_XDBSCHEMA Function, 193-5
NAMESPACE_XMILDIFF Function, 193-5
NAMESPACE_XMLINSTANCE Function, 193-6
NAMESPACE_XMLSCHEMA Function, 193-6
NAMESPACEIDTABLE Procedure, 39-4
NEW LINE procedure, 112-9, 264-19
NEW_ROW_LIST function and procedure, 230-3
NEWCONTEXT function, 201-5, 204-7, 205-6, 209-5
NEWCONTEXTFROMHIERARCHY Function, 201-6
NEWDOMDOCUMENT function, 200-74
NEWPARSER function, 203-4
NEWPROCESSOR function, 211-4
NEWSTYLESHEET function, 211-4
NEXT_DATE procedure, 88-7
NEXT_ITEM_TYPE function, 119-14

NL function
of HTF package, 215-58
NL procedure
of HTP package, 216-55
NOBR function
of HTF package, 215-59
NOBR procedure
of HTP package, 216-56
NOFRAMESCLOSE function
of HTF package, 215-59
NOFRAMESCLOSE procedure
of HTP package, 216-56
NOFRAMESOPEN function
of HTF package, 215-59
NOFRAMESOPEN procedure
of HTP package, 216-57
NOOP function and procedure, 276-17
NOOP Functions, 25-4
NORMAL function, 128-3
NORMAL_DIST_FIT procedure, 161-2
NORMALIZE procedure, 200-75
NORMALIZEPATH Functions, 48-47
NSPREFIX_ACL_ACL Function, 193-6
NSPREFIX_RESCONFIG_RC Function, 193-6
NSPREFIX_RESOURCE_R Function, 193-7
NSPREFIX_XDB_XDB Function, 193-7
NSPREFIX_XMILDIFF_XD Function, 193-7
NSPREFIX_XMLINSTANCE_XSI Function, 193-7
NSPREFIX_XMSCSHEMA_XSD Function, 193-8
NUMBER_ARRAY Table Type, 183-4
NUMBER_TABLE Table Type, 154-23
NVARRAY_ADD Procedure, 104-20
NVARRAY_FIND_NAME Function, 104-20
NVARRAY_FIND_NAME_TYPE Function, 104-21

O

OBJECT_DEPENDENT_SEGMENTS function, 150-9
OBJECT_GROWTH_TREND function, 150-10
OBJECT_HEAT_MAP Procedure, 76-4
OLAP aggregation operators, 40-19
OLAP logs
  Cube Build, 42-4
  Cube Dimension Compile, 42-6
  Cube Operations, 42-6
  Cube Rejected Records, 42-7
OLAP metadata
  upgrading OLAP 10g to OLAP 12c, 40-8
OLAP PL/SQL packages
  DBMS_AW_STATS, 31-1
  DBMS_CUBE, 40-1
  DBMS_CUBE_ADVISE, 41-1
  DBMS_CUBE_LOG, 42-1
OLAP templates
  validation, 40-49
  writing to XML files, 40-40
OLAP_DBA role, 40-2
OLAP_USER role, 40-2
OLD_CURRENT_SCHEMA Function, 183-29
OLD_CURRENT_USER Function, 183-29
OLISTCLOSE function
  of HTF package, 215-60
OLISTCLOSE procedure
  of HTF package, 216-57
OLISTOPEN function
  of HTF package, 215-60
OLISTOPEN procedure
  of HTP package, 215-60
ONLINE_INDEX_CLEAN Function, 132-9
OPATCH_COMPARE_CURRENT Function, 127-8
OPATCH_COMPARE_NODES Function, 127-8
OPEN Procedures, 93-61
OPEN_CONNECTION function, 278-12
OPEN_CONNECTION functions, 276-18
OPEN_CURSOR function, 154-55
OPEN_CURSOR Function, 81-13
OPEN_DATA function and procedure, 276-19
OPEN_WINDOW Procedure, 143-97
optimizer statistics, 31-1
OR REPLACE clause
  for creating packages, 1-3
ORA-28104 error, 138-6
Oracle Data Mining, 43-1
Oracle Streams
  administrator
    granting privileges, 167-2
    revoking privileges, 167-5
  advisors, 166-1
  change data capture
    configuring, 165-85
    compatibility, 164-3-164-6, 289-35
  creating queues, 165-167
  data dictionary
    removing information, 165-144
Oracle Streams (continued)
  messaging
    notification, 165-158
    privileges, 167-1
  replication
Oracle XStream
  creating queues, 212-98
  data dictionary
    removing information, 212-80
Oracle-supplied types
  logical change record (LCR) types, 289-1
  rule types, 297-1
ORD_AUDIO package, 217-1
ORD_DICOM package, 218-1
ORD_DICOM_ADMIN package, 219-1
ORD_DOC package, 220-1
ORD_IMAGE package, 221-1
ORD_VIDEO package, 222-1
outbound servers
  rules, 165-7
OVERLAY function, 273-19
OVERRIDE_PRIORITY Procedures, 30-5
OWA_COOKIE package, 224-1
OWA_PATTERN package, 228-1
OWA_SEC package, 229-1
OWNER Function, 260-8

P

PACK_MESSAGE procedures, 119-15
PACK_STGTAB procedure, 159-16
PACK_STGTAB_BASELINE Function, 153-21
PACK_STGTAB_DIRCETIVE Function, 152-6
PACK_STGTAB_SQLPATCH Procedure, 157-26
PACK_STGTAB_SQLPROF Procedure, 160-45
PACK_STGTAB_SQLSET Procedure, 160-47
package
  DBMS_ODCI, 110-1
  DBMS_XDB, 190-1
  DBMS_XMLODOM, 200-1
  DBMS_XMLENENC, 201-1
  DBMS_XMLPARSER, 203-1
  DBMS_XMQUERY, 204-1
  DBMS_XMLSAVE, 205-1
  DBMS_XMSchema, 206-1
  UriFactory, 286-16
Package - UriFactory, 286-16
package overview, 1-2
package variables
  i_am_a_refresh, 106-6
packages
  creating, 1-2
packages (continued)
   DBMS_MGD_ID_UTL, 102-1
   referencing, 1-5
   where documented, 1-5
PARA function
   of HTF package, 215-61
PARA procedure
   of HTP package, 216-58
PARAGRAPh function
   of HTF package, 215-61
PARAGRAPh procedure
   of HTP package, 216-58
PARAGRAPh function
   of HTF package, 215-61
PARAGRAPh procedure
   of HTP package, 216-58
PARAM function
   of HTF package, 215-62
PARAM procedure
   of HTP package, 216-59
PARSE procedure, 154-57, 203-4
PARSE Procedure, 81-14
PARSEBUFFER procedure, 203-5
PARSECLOB procedure, 203-5
PARSEDTO procedure, 203-6
PARSEDTDBUFFER procedure, 203-6
PARSEDTDCLB procedure, 203-7
PATCHREPOSITORYRESCONFIGLIST
   Procedure, 133-8
PATH_ITEM_T Record Type, 48-18
PATH_ITEMS_T Table Type, 48-22
PATHIDTABLE Function, 39-4
PAUSE_PROFILER function and procedure, 125-7
PAUSE_REPLAY Procedure, 187-21
PIECEWISE member procedure
   of ANYDATA TYPE, 281-8
   of ANYDATASET TYPE, 282-8
PING procedure, 53-29
PL/SQL
datatypes, 56-3
   numeric codes for, 56-4
functions
   DBMS_MGWADM package
      subprograms, 103-10
   DBMS_MGWMSG package
      subprograms, 104-18
procedures
   DBMS_MGWADM package
      subprograms, 103-10
   DBMS_MGWMSG package
      subprograms, 104-18
PL/SQL package
   DBMS_HS_PARALLEL, 80-1
PLAINTEXT function
   of HTF package, 215-62
PLAINTEXT procedure
   of HTP package, 216-60
plan stability, 111-1
PLSQL_TRACE_VERSION procedure, 175-6
point-in-time recovery
   Oracle Streams, 165-80
pointer to
   CTX_ANL package, 4-1
POISSON_DIST_FIT procedure, 161-3
POLICY_CONDITIONS Table Type, 179-2
POPULATE procedure, 84-3, 99-2
POPULATE_DIVERGENCE Procedure, 187-21
PORT_STRING Function, 183-29
POST_INSTANTIATION_SETUP procedure, 165-134
PRE_INSTANTIATION_SETUP procedure, 165-139
PRECLOSE function
   of HTF package, 215-63
PRECLOSE procedure
   of HTP package, 216-60
predictive analytics, 121-1
PREOPEN function
   of HTF package, 215-63
PREOPEN procedure
   of HTP package, 216-60
PREPARE Procedure, 187-23
PREPARE_COLUMN_VALUES procedures, 162-146
PREPARE_COLUMN_VALUES_NVARCHAR2
   procedure, 162-149
PREPARE_COLUMN_VALUES_ROWID
   procedure, 162-150
PREPARE_CONSOLIDATED_REPLAY
   Procedure, 187-22
PREPARE_GLOBAL_INSTANTIATION
    procedure, 32-29
PREPARE_SCHEMA_INSTANTIATION
    procedure, 32-30
PREPARE_SYNC_INSTANTIATION function, 32-32
PREPARE_TABLE_INSTANTIATION procedure, 32-33
PREVIEW_ILM Procedure, 82-6
PRINT function
   of HTF package, 215-64
PRINT procedure
   of HTP package, 216-61
PRINT_BACKTRACE procedure, 53-29
PRINT_CGI_ENV procedure, 231-11
PRINT_INSTANTIATIONS procedure, 53-30
PRINT_MULTI procedure, 230-3
PRINT_POST_PROCESSED_SOURCE
   Procedure, 122-4
PRINT_POST_PROCESSED_SOURCE
   Procedure, 122-4
PRINTS procedure
   of HTP package, 216-62
PRINTWARNINGS Procedure, 207-7
privileges
   GoldenGate administrator, 73-2, 73-6
   Oracle Streams administrator, 167-2, 167-5
   XStream administrator, 213-2, 213-6
PRN function
   of HTF package, 215-64
PRN procedure
   of HTP package, 216-62
PROBE_VERSION procedure, 53-31
PROCESS_CAPTURE Procedure, 187-27
PROCESS_CONNECT_STRING Procedure, 54-6
PROCESS_PENDING Procedure, 202-5
PROCESSLINKS Procedure, 194-19
PROCESSXSL function, 211-5
PROGRAM_INFO Record Type, 53-12
PROP_ITEM_T Record Type, 48-20
PROP_ITEMS_T Table Type, 48-23
PROPAGATEORIGINALEXCEPTION procedure, 204-7
propagations
   altering, 126-2
   creating, 126-4, 165-27, 165-37, 165-43, 165-53, 165-63, 212-8, 212-21, 212-33, 212-42
   DBMS_PROPAGATION_ADM package, 126-1
dropping, 126-7
   propagation user, 165-2
   rules
      defining global, 165-27, 212-8
      defining message, 165-37
      defining schema, 165-43, 212-21
      defining subset, 165-53, 212-33
      defining table, 165-63, 212-42
      for LCRs, 165-7
      for user messages, 165-7
   starting, 126-9
   stopping, 126-9
PROPANY Functions, 48-48
PROPERTIES_T Table Type, 48-23
PROPERTIESH2T Function, 48-49
PROPERTIEST2H Function, 48-49
PROPERTY_T Record Type, 48-21
PROPNUMBER Function, 48-50
PROPRAW Function, 48-50
PROPTIMESTAMP Function, 48-51
PROPVARCHAR2 Function, 48-51
PS procedure
   of HTP package, 216-63
PUBLISH_PENDING_STATS Procedure, 162-152
PULL_TABLESPACES procedure, 170-24
PURGE procedure, 119-16
PURGE Procedure, 149-5
PURGE_COMPARISON Procedure, 33-19
PURGE_CONTEXT Procedure, 69-9
PURGE_FILE_GROUP procedure, 66-17
PURGE_LOST_DB_ENTRY procedure, 176-4
PURGE_MIXED procedure, 176-6
PURGE_PROCESSED_CHUNKS Procedure, 113-13
PURGE_QUEUE_TABLE Procedure, 22-34
PURGE_REFRESH_STATS procedure, 107-2
PURGE_REFRESH_STATS Procedure, 171-10
PURGE_SOURCE_CATALOG procedure, 165-144, 212-80
PURGE_SQL_DETAILS Procedure, 188-34
PURGE_STATS procedure, 162-153
PURGE_TOMBSTONES procedure, 72-10
PURGEALL Procedure, 48-51, 49-16
PURGELDAPCACHE function, 198-6
PURGEPATH Procedure, 48-52, 49-16
PURGERESOURCENMETADATA Procedure, 194-19
PURGESCHEMA Procedure, 206-10
PUT procedure, 264-20
PUT procedures, 112-9
PUT_COL procedure, 173-38
PUT_FILE procedure, 67-6
PUT_LINE Procedure, 264-20
PUT_LINE procedures, 112-10
PUT_LINE_NCHAR procedure, 264-21
PUT_NCHAR procedure, 264-22
PUT_RAW function, 264-26
PUT_ROW_SET procedure, 173-42
PUTF procedure, 264-23
PUTF_NCHAR procedure, 264-25
PUTPATH Procedures, 48-52, 49-17
Q
QNAMEIDTABLE Function, 39-4
QUALIFIED_SQL_NAME Function, 25-4
QUERY_LINK_INFO procedure, 182-7
QUERY_NODE_INFO procedure, 182-8
QUEUE_PROPS_T Type, 284-13
queues
   AnyData
      creating, 165-167, 212-98
      removing, 165-147, 212-83
   QUICK_TUNE Procedure, 16-37
   QUIT function and procedure, 276-20
QUOTED_PRINTABLE_DECODE function, 263-5
QUOTED_PRINTABLE_ENCODE function, 263-6
Index
REMOVE_AUTO_CDR_DELTA_RES procedure, 72-11
REMOVE_CAPTURE Procedure, 187-30
REMOVE_CATEGORY procedure, 102-22
REMOVE_COLORED_SQL Procedure, 188-36
REMOVE_DISCOVERY_RESULT Procedure, 178-8
REMOVE_FILE procedure, 66-17
REMOVE_FROM_ILM Procedure, 82-6
REMOVE_HOST_ACE Procedure, 108-18
REMOVE_LINK procedure, 116-5, 117-3
REMOVE_MSGSYSTEM_LINK Procedure, 103-38
REMOVE_OPTION Procedure, 103-39
REMOVE_PAIR member procedure, 297-5
REMOVE_PIPE function, 119-18
REMOVE_PROXY procedure, 102-23
REMOVE_QUEUE procedure, 165-147, 212-83
REMOVE_REFERENCE procedure, 159-19
REMOVE_RULE procedure, 142-18, 165-148, 212-84
REMOVE_SCHEDULE_ORDERING Procedure, 187-30
REMOVE_SCHEMA procedure, 102-23
REMOVE_SQLSET_REFERENCE procedure, 160-51
REMOVE_STMT_FROM_HANDLER procedure, 168-8
REMOVE_STMT_HANDLER procedure, 20-32
REMOVE_STREAMS_CONFIGURATION procedure, 165-149
REMOVE_SUBSCRIBER Procedure, 103-40
REMOVE_SUBSET_OUTBOUND_RULES procedure, 212-85
REMOVE_TABLE_FROM_APPLICATION, 69-11
REMOVE_WALLET_ACE Procedure, 108-18
REMOVE_XSTREAM_CONFIGURATION procedure, 212-86
REMOVEALL procedure, 17-5
REMOVEANYSTORAGE Procedure, 207-8
REMOVEATTRIBUTE procedure, 200-75
REMOVEATTRIBUTENODE function, 200-76
REMOVECHILD Function, 200-77
REMOVEDEFAULTTABLE Procedure, 207-8
REMOVENMAINTAINDOM Procedure, 207-9
REMOVENAMEDITEM function, 200-77
REMOVEOUTOFLINE Procedure, 207-9
REMOVEPARAM procedure, 211-8
REMOVESQLCOLLTYPET Procedure, 207-10
REMOVESQLNAME Procedure, 207-11
REMOVESQLTYPET Procedure, 207-12
REMOVESQLTYPETMAPPING Procedure, 207-12
REMOVETABLEPROPS Procedure, 207-13

REMOVETIMESTAMPWITHTIMEZONE Procedure, 207-14
REMOVEXSLTPARAM procedure, 204-8, 205-7
RENAME_BASELINE Procedure, 188-36
RENAME_COLUMN member procedure, 289-11
RENAME_COLUMN procedure, 165-151, 212-88
RENAME_COLUMN Procedure, 89-7
RENAME_SCHEMA procedure, 165-154, 212-91
RENAME_TABLE procedure, 165-156, 212-93
RENAMECOLLECTIONTABLE Procedure, 208-9
RENAMEPATH Procedure, 49-18
RENAMEPATH Procedures, 48-55
RENAMERESOURCE Procedure, 194-19
REPLACECHILD function, 200-78
REPLACEDATA procedure, 200-79
REPLAY_SQL_TESTCASE Function, 157-26
REPLICATION
Oracle Streams configuring, 165-15, 165-93, 165-98,
  165-109, 165-112, 165-125,
  165-134, 165-139
split and merge, 165-129, 165-132, 165-169,
  212-74, 212-77, 212-100
REPOPULATE procedure, 84-4
REPORT Function, 186-9, 187-31
REPORT_ADVISOR_TASK function, 162-155
REPORT_ANALYSIS_TASK Function, 158-8
REPORT_AUTO_EVOLVE_TASK Function, 153-22
REPORT_AUTO_TUNING_TASK function, 29-3
REPORT_AUTO_TUNING_TASK Function, 160-52
REPORT_COL_USAGE Procedure, 162-159
REPORT_DIAGNOSIS_TASK Function, 157-28
REPORT_EVOLVE_TASK Function, 153-23
REPORT_GATHER_AUTO_STATS Function, 162-160
REPORT_GATHER_DATABASE_STATS Functions, 162-162
REPORT_GATHER_DICTIONARY_STATS Functions, 162-168
REPORT_GATHER_FIXED_OBJ_STATS Function, 162-172
REPORT_GATHER_SCHEMA_STATS Functions, 162-174
REPORT_GATHER_TABLE_STATS Function, 162-179
REPORT_PERFHUB Procedure, 118-2
REPORT_REPOSITORY_DETAIL Function, 28-2
REPORT_REPOSITORY_DETAIL_XML Function, 28-3
REPORT_REPOSITORY_LIST_XML Procedure, 28-3
REPORT_SESSION Function, 118-4
REPORT_SINGLE_STATS_OPERATION Function, 162-184
REPORT_SQL_DETAIL Function, 160-53
REPORT_SQL_MONITOR Function, 155-4, 160-57
REPORT_SQL_MONITOR_LIST Function, 155-8, 160-62
REPORT_SQL_MONITOR_LIST_XML Function, 155-10
REPORT_SQL_MONITOR_XML Function, 155-8
REPORT_STATS_OPERATIONS Function, 162-186
REPORT_TUNING_TASK function, 160-65
REPORT_TUNING_TASK_XML function, 160-67
REQ Type, 265-11
REQUEST function, 94-8, 265-52
REQUEST_CONTEXT_KEY Type, 265-12
REQUEST_PIECES function, 265-55
RESET_ADVISOR_TASK procedure, 162-188
RESET_ANALYSIS_TASK Procedure, 158-10
RESET_BUFFER procedure, 119-18
RESET_COL_USAGE Procedure, 162-189
RESET_DIAGNOSIS_TASK Procedure, 157-29
RESET_OEVOKE_TASK Procedure, 153-22
RESET_GLOBAL_PREFS_DEFAULTS Procedure, 162-189
RESET_JOB Procedure, 103-41
RESET_PACKAGE Procedure, 146-16
RESET_PARAM_DEFAULTS Procedure, 162-190
RESET_SQLWKLD Procedure, 16-38
RESET_SUBSCRIBER Procedure, 162-190
RESET_TASK Procedure, 16-39
RESTARTQUERY procedure, 201-6
RESTORE function, 163-7
RESTORE_DATAFILE_PERMISSIONS Procedure, 60-2
RESTORE_DATABASE_STATS procedure, 162-190
RESTORE_DEFAULTS Procedure, 35-5
RESTORE_DICTIONARY_STATS procedure, 162-191
RESTORE_FIXED_OBJECTS_STATS procedure, 162-191
RESTORE_SCHEMA_STATS procedure, 162-192
RESTORE_SYSTEM_STATS procedure, 162-193
RESTORE_TABLE_STATS procedure, 162-194
RESTOREALL Procedure, 48-55, 49-19
RESTOREPATH Procedure, 48-56, 49-19
RESUME_ADVISOR_TASK procedure, 162-195
RESUME_ANALYSIS_TASK Procedure, 158-10
RESUME_DIAGNOSIS_TASK Procedure, 157-29
RESUME_EVOLVE_TASK Procedure, 153-22
RESUME_PROFILER function and procedure, 125-7
RESUME_REPLAY Procedure, 187-31
RESUME_TASK Procedure, 113-14
RESUME_TUNING_TASK Procedure, 160-69
RETURN_RESULT Procedures, 154-60
REUSE_REPLAY_FILTER_SET Procedure, 187-31
REVERSE function, 273-20
REVOKE_ADMIN_PRIVILEGE procedure, 73-6, 167-5, 213-6
REVOKE_OBJECT_PRIVILEGE procedure, 66-18, 142-19
REVOKE_REMOTE_ADMIN_ACCESS procedure, 167-7, 213-9
REVOKE_SWITCH_CONSUMER_GROUP procedure, 135-3
REVOKE_SYSTEM_PRIVILEGE procedure, 66-19, 135-4, 142-20
ROLLBACK procedure, 176-7
ROLLBACK Procedure, 130-13
ROLLBACK_FORCE procedure, 176-7
ROLLBACK_PLAN Procedure, 139-10
ROLLBACK_SAVEPOINT procedure, 176-7
row migration, 165-53, 165-57, 212-33, 212-37
ROW_LIST DATA TYPE, 230-1
ROW_SET_T Type, 173-27
ROWID datatype extended format, 140-10
ROWID_BLOCK_NUMBER function, 140-5
ROWID_CREATE function, 140-5
ROWID_INFO procedure, 140-6
ROWID_OBJECT function, 140-7
ROWID_RELATIVE_FNO function, 140-8
ROWID_ROW_NUMBER function, 140-9
ROWID_TO_ABSOLUTE_FNO function, 140-9
ROWID_TO_EXTENDED function, 140-9
ROWID_TO_RESTRICTED function, 140-12
ROWID_TYPE function, 140-12
ROWID_VERIFY function, 140-13
RSET Function and Procedure, 276-22
rule sets adding rules to, 142-2
creating, 142-11
dropping, 142-14
removing rules from, 142-18
rule-based transformations setting, 165-163
rules

action contexts
  - adding name-value pairs, 297-5
  - getting name-value pairs, 297-5
  - getting value for name, 297-5
  - removing name-value pairs, 297-5
  - transformations, 165-163
altering, 142-6
creating, 142-10
DBMS_RULE package, 141-1
DBMS_RULE_ADM package, 142-1
dropping, 142-13
evaluation, 141-3
  - iterators, 141-2, 141-10, 141-11
evaluation contexts
  - altering, 142-4
  - creating, 142-8
  - dropping, 142-12
fast evaluation, 141-11
inbound servers, 165-7
object privileges
  - granting, 142-14
  - revoking, 142-19
outbound servers, 165-7
propagations
  - removing, 165-148, 212-84
RE$ATTRIBUTE_VALUE type, 297-3
RE$ATTRIBUTE_VALUE_LIST type, 297-3
RE$COLUMN_VALUE type, 297-4, 297-7
RE$COLUMN_VALUE_LIST type, 297-4
RE$NAME_ARRAY type, 297-4
RE$NV_ARRAY type, 297-4
RE$NV_LIST type, 297-5
RE$RULE_HIT type, 297-7, 297-8
RE$RULE_HIT_LIST type, 297-8, 297-9
RE$TABLE_ALIAS type, 297-9
RE$TABLE_ALIAS_LIST type, 297-10
RE$TABLE_VALUE type, 297-10
RE$TABLE_VALUE_LIST type, 297-11
RE$VARIABLE_TYPE type, 297-12
RE$VARIABLE_TYPE_LIST type, 297-14
RE$VARIABLE_VALUE type, 297-14
RE$VARIABLE_VALUE_LIST type, 297-14
subset
  - defining, 165-53, 165-57, 212-33, 212-37
system privileges
  - granting, 142-16
  - revoking, 142-20
system-created, 165-7
global apply, 165-32, 212-12
global capture, 165-32, 212-12
global propagation, 165-27, 212-8
global schema, 165-48, 212-26
message, 165-40
message propagation, 165-37

rules (continued)

system-created (continued)
  - removing, 165-148, 212-84
schema capture, 165-48, 212-26
schema propagation, 165-43, 212-21
subset apply, 165-57, 212-37
subset capture, 165-57, 212-37
subset propagation, 165-53, 212-33
table apply, 165-67, 212-46
table capture, 165-67, 212-46
table propagation, 165-63, 212-42

types, 297-1
RUN procedure, 88-8
RUN_CHECK Procedure, 78-2
RUN_TASK Procedure, 113-16
RUNTIME_INFO Record Type, 53-13

S

S function
  - of HTF package, 215-65
S procedure
  - of HTP package, 216-64
SAM (SQL Aggregation Management), 40-2
SAMPLE function
  - of HTF package, 215-66
SAMPLE procedure
  - of HTP package, 216-64
SAVE function, 163-7
SAVE Procedure, 196-19
SAVEPOINT procedure, 176-8
SCHEDULE_PROPAGATION Procedure, 22-37, 103-42
SCHEMA_NAME Function, 25-5
SCHEMAELEM_RES_ACL Function, 193-8
SCHEMAELEM_RESCONTENT_BINARY Function, 193-8
SCHEMAELEM_RESCONTENT_TEXT Function, 193-9
SCHEMAMURL_ACL Function, 193-8
SCHEMAMURL_RESOURCE Function, 193-9
SCHEMAMURL_XDBSCHEMA Function, 193-9
SCHEMVALIDATE procedure, 300-9
SCOPESXMLREFERENCES Procedure, 208-10
SCRIPT function
  - of HTF package, 215-66
SCRIPT procedure
  - of HTP package, 216-65
SCRIPT_ADVISOR_TASK function, 162-196
SCRIPT_TUNING_TASK Function, 160-72
SDO_CS package, 232-1
SDO_CSW_PROCESS package, 233-1
SDO_GCDR package, 234-1
SDO_GEOM package, 235-1
SDO_GEOR package, 236-1
SDO_GEOR_ADMIN package, 237-1
SDO_GEOR_AGGR package, 238-1
SDO_GEOR_RA package, 239-1
SDO_GEOR_UTL package, 240-1
SDO_LRS package documentation, 241-1
SDO_MIGRATE package, 242-1
SDO_NET package documentation, 243-1
SDO_NFE package, 244-1
SDO_OLS package, 245-1
SDO_PC_PKG package, 246-1
SDO_SAM package, 247-1
SDO_TIN_PKG package, 248-1
SDO_TOPO package, 249-1
SDO_TOPO_MAP package, 250-1
SEND_ATTACH_RAW procedure, 270-3
SEND_ATTACH_VARCHAR2 procedure, 270-4
SEND_MESSAGE function, 119-19
SENDCOMMAND Procedures, 50-13
SERV_MOD_ACT_STAT_DISABLE procedure, 105-5
SERV_MOD_ACT_STAT_ENABLE procedure, 105-5
SERV_MOD_ACT_TRACE_DISABLE procedure, 105-7
SERV_MOD_ACT_TRACE_ENABLE procedure, 105-8
SESSION_TRACE_DISABLE procedure, 146-16
SESSION_TRACE_ENABLE Procedure, 146-16
SET Procedure, 75-3
SET_ATTRIBUTE Procedure, 156-11
SET_ADVISOR_TASK_PARAMETER Procedure, 187-32
SET_AUDIT_TRAIL_LOCATION procedure, 26-24
SET_AUDIT_TRAIL_PROPERTY procedure, 26-25
SET_AUTHORIZATION procedure, 229-3
SET_BASE_TABLE_NAME member procedure, 289-2
SET_BASE_TABLE_OWNER member procedure, 289-2
SET_CLIENT_INFO procedure, 471-4
SET_CLIENT_SERVICE Procedure, 30-6
SET_COLUMN_STATS procedures, 162-201
SET_CLIENT_SERVICE Procedure, 30-6
SET_CNS_EXCEPTION_LOG procedure, 134-25
SET_CONSUMER_GROUP_MAPPING procedure, 134-25
SET_CONS_EXCEPTION_LOG procedure, 41-4
SET_COLUMN_STATS procedures, 162-201
SET_COMMAND_TYPE member procedure, 289-35
SET_CONSUMER_GROUP_MAPPING procedure, 134-25
SET_CONSUMER_GROUP_MAPPING procedure, 134-25
SET_CONSUMER_GROUP_MAPPING procedure, 134-25
SET_PURGE_JOB_STATUS procedure, 26-31
SET_REPERROR_HANDLER procedure, 20-79
SET_REPLAY_DIRECTORY Procedure, 187-33
SET_REPLAY_TIMEOUT Procedure, 187-33
SET_RESPONSE_ERROR_CHECK procedure, 265-69
SET_ROLE Procedure, 146-20
SET_ROLE_EXPLICIT procedure, 116-9, 117-7
SET_ROWID_THRESHOLD Procedure, 36-20
SET_RULE_TRANSFORM_FUNCTION procedure, 165-163
SET_SCHEMA_INSTANTIATION procedure, 20-80
SET_SCHEMA_PREFS procedure, 162-225
SET_SEGADV_ATTRIB Procedure, 151-12
SET_SENDFROM Procedure, 23-2
SET_SESSION_LONGOPS procedure, 19-5
SET_SESSION_TIMEOUT procedure, 137-3
SET_SOURCE_DATABASE_NAME member procedure, 289-35
SET_SQL_MAPPING Procedure, 187-34
SET_SQL_TRACE Procedure, 146-21
SET_SQLWKLD_PARAMETER Procedure, 16-42
SET_SURROGATE_CONTROL Procedure, 223-4
SET_SYSTEM_DEFAULT Procedure, 107-4
SET_SYSTEM_STATS procedure, 162-233
SET_TABLE_INSTANTIATION procedure, 20-83
SET_TABLE_PREFS procedure, 162-235
SET_TABLE_STATS procedure, 162-244
SET_TAG member procedure, 289-35
SET_TAG procedure, 164-12, 165-166, 212-97
SET_TASK_PARAMETER Procedure, 16-43
SET_THRESHOLD procedure, 144-8
SET_TIMEOUT function, 53-33
SET_TIMEOUT procedure, 137-4
SET_TIMEOUT_BEHAVIOUR procedure, 53-34
SET_TRANSFER_TIMEOUT procedure, 265-70
SET_TRANSFORM procedure, 44-68
SET_TRIGGER_FIRING_PROPERTY procedure, 52-7
SET_TUNING_TASK_PARAMETER Function, 160-87
SET_UP_QUEUE procedure, 165-167, 212-98
SET_UPDATE_CONFLICT_HANDLER procedure, 20-86
SET_USER_EXPLICIT procedure, 116-10, 117-7
SET_USER_MAPPING Procedure, 187-35
SET_VALUE function, 53-35
SET_VALUE member procedure, 289-11
SET_VALUE_DEPENDENCY procedure, 20-89
SET_VALUES member procedure, 289-11
SET_WALLET procedure, 265-70
SET_WALLET_ACL Procedure, 108-20
SET_WARNING_SETTING_STRING procedure, 184-5
SET_WATERMARK Procedure, 22-39
SET_XML_INFORMATION member procedure, 289-11
SET* member procedures
  of ANYDATA TYPE, 281-8
  of ANYDATASET TYPE, 282-9
SETACL Procedure, 194-20, 196-20
SETANYSTORAGE Procedure, 207-14
SETATTRIBUTE procedure, 200-80
SETATTRIBUTENODE function, 200-81
SEAUTHOR Procedure, 196-20
SETBASEDIR procedure, 203-7
SETBATCHSIZE procedure, 205-7
SETBINDVALUE procedure, 204-8
SETCHARACTERSET Procedure, 196-20
SETHOOKSET Procedure, 200-81
SETCOLLIDATTRNAME procedure, 204-9
SETCOMMENT Procedure, 196-21
SETCOMMITBATCH procedure, 205-8
SETCONTENT Procedures, 196-21
SETCONTENTTYPE Procedure, 93-64, 196-22
SETCONVTSPECIALCHARS procedure, 201-7
SETCUSTOMMETADATA Procedure, 196-22
SETDATA procedure, 200-82
SETDATAHEADER procedure, 204-9
SETDATEFORMAT procedure, 204-9, 205-8
SETDEFAULTACL Procedure, 48-56
SETDEFAULTASOF Procedure, 48-57
SETDEFAULTCONTEXT Procedure, 48-57
SETDEFAULTOWNER Procedure, 48-58
SETDEFAULTPRINCIPAL Procedure, 48-59
SETDEFAULTTABLE Procedure, 207-15
SETDISPLAYNAME Procedure, 196-23
SETOCTYPE procedure, 203-8
SETOCTYPE Procedure, 200-82
SETDVERSION procedure, 200-87
SETENCODINGTAG procedure, 204-10
SETERROREXT Procedure, 203-8, 211-10
SETERRORTAG procedure, 204-10
SETFTPPORT Procedure, 192-13
SETFORCEألعاب Procedure, 192-14
SETGLOBALPORTENABLED Procedure, 192-13
SETHTTPCONFIGREALM Procedure, 192-15
SETHTTPPORT Procedure, 192-15
SETHTTPSPORT Procedure, 192-15
SETIGNORECASE procedure, 205-9
SETINFO member procedure
  of ANYTYPE TYPE, 283-2
SETKEYCOLUMN procedure, 205-9, 209-5
SETLANGUAGE Procedure, 196-23
SETLISTENERENDPOINT Procedure, 192-16
SETLISTENERLOCALACCESS Procedure, 192-16
SETMAXROWS procedure, 201-7, 204-11
SETMETAHEADER procedure, 204-11
SETRANDOMITEM function, 200-83
SETNODEVALUE procedure, 200-84
SETNODEVALUEASBINARYSTREAM Function ((amp)) Procedure, 200-84
SETNODEVALUEASCARACTERSTREAM Function ((amp)) Procedure, 200-85
SETNULLHANDLING Procedure, 201-8
SETOPTIONS Procedure, 93-65
SETOUTOFLINE Procedure, 207-15
SETOWNER Procedure, 196-24
SETPARAM procedure, 211-10
SETPATH Procedure, 49-21
SETPATH Procedures, 48-59
SETPREFIX procedure, 200-86
SETPRESERVEWHITESPACE procedure, 203-9, 205-10
SETRAISEEXCEPTION procedure, 204-11
SETRAISENOROWSEXCEPTION procedure, 204-12
SETREMOTEHTTPPORT Procedure, 192-17
SETREMOTEHTTPSPORT Procedure, 192-17
SETRENDERPATH Procedure, 199-25
SETRENDERSTREAM Procedure, 199-26
SETROWIDATTRNAME procedure, 204-12
SETROWIDATTRVALUE procedure, 204-12
SETROWSETTAG procedure, 201-8, 204-13
SETROWTAG procedure, 204-13, 205-10, 209-5
SETRAISESTATANOTATIONS Procedure, 207-17
SETRAISVALIDATED procedure, 300-9
SETSKIPROWS procedure, 201-10, 204-14
SETSQCOLLTYPE Procedure, 207-18
SETSQNAME Procedure, 207-19
SETSQTALEXMLNAMEESCAPING procedure, 204-14, 205-11
SETSQTYPE Procedure, 207-20
SETSQTYPEMAPPING Procedure, 207-21
SETSTANDALONE procedure, 200-86
SETSTATS Procedure, 48-60
SETSTOREPROPERTY Procedure, 50-14
SETSTYLESCHEMATIC Property, 204-14
SETTABLEPROPS Procedure, 207-22
SETTAGCASE procedure, 204-15
SETTIMESTAMPPWITTIMEZONE Procedure, 207-23
SETTRACEL Procedure, 48-61
SETUPDATECOLUMN procedure, 205-11, 209-6
SETVALIDATIONMODE procedure, 203-9
SETVALUE procedure, 200-87
SETXSLT procedure, 204-15, 205-11
SETXSLTPARAM procedure, 204-16, 205-12
SHOW_BREAKPOINTS procedures, 53-36
SHOW_EXTENDED_STATS_NAME Function, 162-247
SHOW_FRAME_SOURCE procedure, 53-37
SHOW_SOURCE procedures, 53-37
SHOW_STATS procedure, 277-15
SHOW_STATS_HTML procedure, 277-18
SHOWPAGE procedure, 231-12
SHOWSOURCE procedure, 231-12
SHOWWARNINGS procedure, 203-9, 211-11
SHUTDOWN Procedure, 103-46
SIGNAL procedure, 17-6
SIGNATURE procedure, 231-13
SIMPLE_SQL_NAME Function, 25-5
SIZE procedure, 149-6
SKIP_CORRUPT_BLOCKS procedure, 132-11
SLEEP procedure, 146-21
SMALL function of HTF package, 215-67
SMALL procedure of HTP package, 216-65
SODA Type
  REMOVE_ONE Function, 298-7
  REPLACE_ONE Function, 298-8
  REPLACE_ONE_AND_GET Function, 298-9
SODA Types, 298-1
  CLOSE Function, 298-22
  COUNT Function, 298-16
  CREATE_INDEX Function, 298-3
  DROP_INDEX Function, 298-4
  FILTER Function, 298-16
  FIND Function, 298-4
  FIND_ONE Function, 298-5
  GET_BLOB Function, 298-10
  GET_CLOB Function, 298-10
  GET_CREATED_ON Function, 298-16
  GET_CURSOR Function, 298-17
  GET_DATA_GUIDE Function, 298-5
  GET_DATA_TYPE Function, 298-11
  GET_KEY Function, 298-11
  GET_LAST_MODIFIED Function, 298-12
  GET_MEDIA_TYPE Function, 298-6
  GET_METADATA Function, 298-6
  GET_NAME Function, 298-6
  GET_ONE Function, 298-17
  GET_VARCHAR2 Function, 298-12
  GET_VERSION Function, 298-12
  HAS_NEXT Function, 298-22
  INSERT_ONE Function, 298-6
  INSERT_ONE_AND_GET Function, 298-7
  KEY Function, 298-17
  KEYS Function, 298-18
  LIMIT Function, 298-19
  NEXT Function, 298-22
  Overview, 298-1
  REMOVE Function, 298-19
SODA Types (continued)

REPLACE_ONE Function, 298-19
REPLACE_ONE_AND_GET Function, 298-20
Security Model, 298-1
SKIP Function, 298-21
SODA_Collection_T Type, 298-2
SODA_Cursor_T Type, 298-22
SODA_Document_T Function, 298-13
SODA_Document_T Type, 298-9
SODA_Operation_T Type, 298-14
Summary, 298-2
VERSION Function, 298-21
SOURCE_LINES_T Table Type, 122-2
SPACE_ERROR_INFO function, 137-4
SPACE_USAGE procedure, 48-61, 49-21
split streams, 165-169, 212-100
SPLIT_STREAMS procedure, 165-169, 212-100
SPLITPATH Procedure, 194-20
SPLITTEXT function, 200-88
SQL Apply
  managing logical standby databases, 97-1
  managing with DBMS_LOGSTDBY package, 97-1
SQL generation, 289-11
SQL Performance Reporting Subprograms, 160-10
SQL_HASH Function, 156-16
SQL_ID Function, 156-16
SQL_OBJECT_NAME Function, 25-6
SQL*Plus
  creating a sequence, 1-5
SQLID_TO_SQILASH Function, 183-30
SQLSET_ROW Object Type, 159-2, 160-5
SQLSET_ROW type, 159-2
SQLTEXT_TO_SIGNATURE function, 160-89
STACK_BIN_CAT procedure, 44-69
STACK_BIN_NUM procedure, 44-71
STACK_CLIP procedure, 44-73
STACK_COL_REM procedure, 44-75
STACK_MISS_CAT procedure, 44-77
STACK_MISS_NUM procedure, 44-79
STACK_NORM_LIN procedure, 44-81
staging
  queues
    creating, 165-167, 212-98
    removing, 165-147, 212-83
START__CONSOLIDATED_REPLAY Procedure, 187-36
START_APPLY procedure, 20-90
START_CAPTURE procedure, 32-51
START_CAPTURE Procedure, 186-10
START_COVERAGE Function, 120-2
START_OUTBOUND procedure, 212-104
START_POOL Procedure, 35-4
START_PROFILER functions and procedures, 125-8
START_PROFILING Procedure, 79-4
START_PROPAGATION procedure, 126-9
START_REDEF_TABLE procedure, 130-14
START_REPLAY Procedure, 187-36
START_REPORT_CAPTURE Procedure, 28-5
START_SERVICE procedure, 145-15
STARTTLS Function and Procedure, 276-23
STARTUP Procedure, 103-46
STARTUP_EXTPROC_AGENT procedure, 214-2
STATUS Function, 136-7
STATUS_LINE procedure, 231-13
STEP_ID function, 176-8
STOP_APPLY procedure, 20-91
STOP_CAPTURE procedure, 32-52
STOP_COVERAGE Procedure, 120-3
STOP_ILM Procedure
  DBMS_ILM, 82-7
STOP_OUTBOUND procedure, 212-105
STOP_POOL Procedure, 35-5
STOP_PROFILER function and procedure, 125-8
STOP_PROFILING Procedure, 79-5
STOP_PROPAGATION procedure, 126-9
STOP_SERVICE procedure, 145-15
STOP_TASK Procedure, 113-18
STORE_T Record Type, 48-21
STORE_VALUES procedure, 227-3
stored outlines
  DBMS_OUTLN, 111-1
  OUTLN_PKG package, 111-1
STOREPUSH Procedure, 50-17
STREAM2MULTI procedure, 230-4
Streams
  removing configuration, 165-149
STREAMS$_TRANSFORM_FUNCTION, 165-165
STRIKE function
  of HTF package, 215-67
STRIKE procedure
  of HTP package, 216-66
STRING function, 128-4
STRONG function
  of HTF package, 215-68
STRONG procedure
  of HTP package, 216-66
STYLE function
  of HTF package, 215-68
STYLE procedure
  of HTP package, 216-67
SUB procedure
  of HTP package, 216-67
SUBMIT procedure, 88-8
SUBMIT_PENDING_AREA procedure, 134-29
SUMMARY procedure, 161-3
SUP function
  of HTF package, 215-70
SUP procedure
  of HTP package, 216-68
SUPPORTED_TYPE function, 173-48
SWITCH_CONSUMER_GROUP_FOR_SESS procedure, 134-29
SWITCH_CONSUMER_GROUP_FOR_USER procedure, 134-29
SWITCH_CURRENT_CONSUMER_GROUP Procedure, 146-22
SWITCH_DESTINATION procedure, 182-9
SWITCH_PLAN procedure, 134-30
SYNC_INTERIM_TABLE procedure, 130-16
SYNCHRONIZE function, 53-39
SYNCHRONIZE_partitions_FOR_HIVE Procedure, 74-2
synchronous capture
  altering, 32-11
  instantiation
  preparing a table for, 32-32
  rules, 165-7
SYNCINDEX Procedure, 202-6
synonyms
  comparing, 33-1
SYS.MGW_MQSERIES_PROPERTIES Object Type, 103-5
SYS.MGW_PROPERTIES Object Type, 103-7
SYS.MGW_PROPERTY Object Type, 103-8
SYS.MGW_TIBRV_PROPERTIES Object Type, 103-9
T
TABLE_COLUMNS_T Type, 173-27
TABLE_CREATE procedure, 42-20
TABLE_T Type, 173-26
TABLE_TO_COMMA Procedures, 183-30
TABLECAPTION function
  of HTF package, 215-70
TABLECAPTION procedure
  of HTF package, 216-68
TABLECLOSE function
  of HTF package, 215-71
TABLECLOSE procedure
  of HTP package, 216-69
TABLEDATA function
  of HTF package, 215-71
TABLEDATA procedure
  of HTP package, 216-69
TABLEHEADER function
  of HTF package, 215-72
TABLEHEADER procedure
  of HTP package, 216-70
TABLEOPEN function
  of HTF package, 215-73
TABLEOPEN procedure
  of HTP package, 216-71
TABLEPRINT function, 231-14
TABLEROWCLOSE function
  of HTF package, 215-73
TABLEROWCLOSE procedure
  of HTP package, 216-72
TABLEROWOPEN function
  of HTF package, 215-74
TABLEROWOPEN procedure
  of HTP package, 216-72
tables
  comparing, 33-1
  table items as arrays, 154-26
tablespace repositories
  attaching tablespaces, 170-6
  cloning tablespaces, 170-12
  detaching tablespaces, 170-18
TABLESPACE_FIX_BITMAPS procedure, 151-12
TABLESPACE_FIX_SEGMENT_STATES procedure, 151-13
TABLESPACE_HEAT_MAP Procedure, 76-6
TABLESPACE_MIGRATE_FROM_LOCAL procedure, 151-14
TABLESPACE_MIGRATE_TO_LOCAL procedure, 151-14
TABLESPACE_REBUILD_BITMAPS procedure, 151-15
TABLESPACE_REBUILD_QUOTAS procedure, 151-16
TABLESPACE_RELOCATE_BITMAPS procedure, 151-17
TABLESPACE_SET type, 170-2, 170-3
TABLESPACE_VERIFY procedure, 151-18
tags
  GET_TAG function, 164-11, 165-82, 212-71
  SET_TAG procedure, 164-12, 165-166, 212-97
TARGET_FILE function, 42-21
TARGET_LOB function, 42-21
TARGET_PROGRAM_RUNNING procedure, 53-39
TARGET_TABLE function, 42-22
TARGET_TRACE function, 42-22
TELETYP function
  of HTF package, 215-74
  of HTP package, 216-72
TERMINATE procedure, 128-5
TEXT_DECODE function, 263-6
TEXT_ENCODE function, 263-7
The DBMS_XDB_REPOS package, 194-1
TIME_TABLE Table Type, 154-23
TIME_WITH_TIME_ZONE_TABLE Table Type, 154-23
TIMESTAMP_TABLE Table Type, 154-23
TIMESTAMP_WITH_LTZ_TABLE Table Type, 154-23
TIMESTAMP_WITH_TIME_ZONE_TABLE Table Type, 154-24
TITLE function
  of HTF package, 215-75
TITLE procedure
  of HTF package, 216-73
TO_CURSOR_NUMBER Function, 154-61
TO_REFCURSOR Function, 154-63
TO_STRING function, 296-9
TODATE function, 231-17
TOOBJECT procedure, 300-10
TOUCRES Resource Procedure, 194-21
TRACE procedure, 41-5
TRAE Procedure, 48-62
TRACEENABLED Function, 48-63
tracking LCRs, 165-80, 165-163, 212-70
TRANSACTION_BACKOUT Procedures, 68-7
TRANSFER_STATS Procedure, 162-247
TRANSFER_UNIFIED_AUDIT_RECORDS procedure, 26-32
TRANSFORM function, 300-10
transformations
  rule-based (continued)
    renaming a column, 165-151, 212-88
    renaming a schema, 165-154, 212-91
    renaming a table, 165-156, 212-93
STREAMS$ TRANSFORM FUNCTION, 165-165
TRANSFORMNODE function, 211-11
TRANSLATE function, 273-22, 296-10
TRANSLATE_ERROR Procedure, 156-17
TRANSLATE_SQL Procedure, 156-18
TRANS-LITERATE Function, 273-24
TRANSPORT_SET_CHECK procedure, 180-2
TRIM Procedures, 93-68
TUNE_MVIEW Procedure, 16-59
TYPE_BUILD function, 42-23
TYPE_DIMENSION_COMPILE function, 42-23
TYPE_OPERATIONS function, 42-24
TYPE_REJECTED_RECORDS function, 42-24
types
  COLUMN_T, 173-24
  DBMS_HIERARCHY package, 77-1
  MGD ID, 296-1, 296-3
  MGD ID COMPONENT, 296-2
  MGD ID COMPONENT VARRAY, 296-2
  Oracle Multimedia
    ORDAudio, 290-1
    ORDDicom, 291-1
    ORDDoc, 292-1
    ORDDImage, 293-1
    ORDDVideo, 295-1
    SQL/MM Still Image, 294-1
  TABLE T, 173-26
U
ULISTCLOSE function
  of HTF package, 215-75
ULISTCLOSE procedure
  of HTP package, 216-73
ULISTOPEN function
  of HTF package, 215-76
ULISTOPEN procedure
  of HTP package, 216-74
UNASSIGN_ACL Procedure, 108-20
UNASSIGN_WALLET_ACL Procedure, 108-21
UNCHECKOUT function, 195-7
UNCL_ARRAY Table type, 183-4
UNCONFIGURE_NODE procedure, 182-10
UNDERLINE function
  of HTF package, 215-76
UNDERLINE procedure
  of HTP package, 216-74
UNESCAPE function, 279-4
UNESCAPEURI function, 286-17
UNIFORM_DIST_FIT procedure, 161-4
UNIQUE_SESSION_ID Function, 146-24
UNIQUE_SESSION_NAME function, 119-21
UNIT_LINE Function, 260-9
UNKEEP procedure, 149-7
UNLOCK_MAP procedure, 163-7
UNLOCK_PARTITION_STATS Procedure, 162-248
UNLOCK_SCHEMA_STATS procedure, 162-249
UNLOCK_TABLE_STATS procedure, 162-249
UNLOCKRESOURCE Function, 194-21
UNMAP_DAD Procedure, 63-15
UNMARKHOT Procedure, 149-8
UNMOUNTSTORE Procedure, 48-64
UNMOUNTVOLUME Procedure, 60-2
UNPACK_MESSAGE procedures, 119-21
UNPACK_STGTAB procedure, 159-34
UNPACK_STGTAB_BASELINE Function, 153-25
UNPACK_STGTAB_DIRECTIVE Function, 152-8
UNPACK_STGTAB_SQLPATCH Procedure, 157-30
UNPACK_STGTAB_SQLPROF procedure, 160-90
UNPACK_STGTAB_SQLSET procedure, 160-91
UNREGISTERDEPENDENTOBJECT procedure, 130-17
UNREGISTERFOREIGN_QUEUE Procedure, 103-47
UNREGISTER_NODE procedure, 182-10
UNREGISTER_REMOTE_DATABASE procedure, 188-38
UNREGISTERSTORE Procedure, 48-65
UNREGISTERURLHANDLER procedure, 286-18
UNSCHEDULEPROPAGATION Procedure, 103-48
UNUSED_SPACE procedure, 150-15
UPDATE_BY_CAT procedure, 111-4
UPDATECATEGORY Procedure, 134-31
UPDATE_CDB_AUTOTASK_DIRECTIVE Procedure, 134-31
UPDATE_CDB_DEFAULT_DIRECTIVE Procedure, 134-32
UPDATE_CDB_PLAN Procedure, 134-33
UPDATE_CDB_PLAN_DIRECTIVE Procedure, 134-34
UPDATE_CDB_PROFILE_DIRECTIVE Procedure, 134-35
UPDATE_CONSUMER_GROUP procedure, 134-36
UPDATE_CREDENTIAL Procedure, 134-37
UPDATE_DATAFILE_INFO procedure, 188-38
UPDATEFULLREDACTIONVALUES Procedure, 129-23
UPDATEOBJECT procedure, 275-6
UPDATE_OBJECT Procedure, 16-61
UPDATE_OBJECT_INFO Function, 188-39
UPDATEPLAN procedure, 134-36
UPDATEPLAN_DIRECTIVE procedure, 134-37
UPDATE_POLICY_EXPRESSION Procedure, 129-24
UPDATEREC_ATTRIBUTES Procedure, 16-63
UPDATESIGNATURES procedure, 111-4
UPDATESQLSET procedures, 159-35, 160-93
UPDATESQLWKLD_ATTRIBUTES Procedure, 16-64
UPDATESQLWKLD_STATEMENT Procedure, 16-65
UPDATERTASK_ATTRIBUTES Procedure, 16-67
UPDATERESOURCENAME_METADATA Procedures, 194-22
UPDATEXML function, 205-13, 209-6
UPGRADE_AW Procedure, 40-48
UPGRADE_DATABASE Procedure, 61-7
UPGRADE_SCHEMA Procedure, 61-9
UPGRADE_STAT_TABLE procedure, 162-250
UPGRADE_TABLE Procedure, 61-10
URI Types
description, 286-1
UriFactory package, 286-16
ESCAPEURI function, 286-16
GETURL function, 286-16
methods, 286-16
REGISTERURLHANDLER procedure, 286-17
UNESCAPEURI function, 286-17
UNREGISTERURLHANDLER procedure, 286-18
UriType supertype, 286-1
GETBLOB function, 286-2
GETCLOB function, 286-3
GETCONTENTTYPE function, 286-3
GETEXTERNALURL function, 286-3
GETURL function, 286-4
GETXML function, 286-4
methods, 286-1
UROWID_TABLE Table Type, 154-24
USE_FILTER_SET Procedure, 187-37
USE_ROLLBACK_SEGMENT procedure, 176-8
USEBINARYSTREAM Function, 200-89
USEPORT Procedure, 192-17
USEITEMTAGSFORCOLL procedure, 201-10
USENULLATTRIBUTEINDICATOR procedure, 201-11, 204-16
USER_EXPORT procedures, 88-10
USETYPEFORCOLLELEMENTTAG procedure, 204-17
UTL Streams Types, 299-1
UTL_BINARYINPUTSTREAM Type, 299-1
UTL_BINARYOUTPUTSTREAM Type, 299-2
V

V$STREAMS_MESSAGE_TRACKING view, 165-80, 212-70
v$vpd_policies, 138-3
VALIDATE Procedure, 183-31

VALIDATE_ANALYTIC_VIEW function
of DBMS_HIERARCHY package, 77-3
VALIDATE_CHECK_SUCCESS function
of DBMS_HIERARCHY package, 77-5
VALIDATE_DIMENSION procedure, 58-2
VALIDATE_HIERARCHY function
of DBMS_HIERARCHY package, 77-6
VALIDATE_PENDING_AREA procedure, 134-42
VALIDATE_REWRITE_EQUIVALENCE Procedure, 15-6
VALIDATE_SCHEME function, 102-26
VALIDATE_XML procedure, 40-49
VALUE functions, 128-5
VALUEOF procedure, 211-11
VARIABLE function of HTF package, 215-77
VARIABLE procedure of HTP package, 216-75
VARIABLE_VALUE_PKG Procedure, 154-66
VRFY function, 276-24
WAIT_ON_PENDING_DML Procedure, 183-32
WAITANY procedure, 17-7
WAITONE procedure, 17-7
WBR function of HTF package, 215-77
WBR procedure of HTP package, 216-75
WEIBULL_DIST_FIT procedure, 161-5
WHAT procedure, 88-10

Virtual Private Database. See VPD, 138-1
WHO_CALLED_ME procedure, 231-17
WRAP Functions, 52-9
WRITE Procedures, 93-70
WRITE_DATA procedure, 276-25
WRITE_LINE function, 278-18
WRITE_LINE procedure, 265-72
WRITE_RAW function, 278-19
WRITE_RAW procedure, 265-73
WRITE_RAW_DATA procedure, 276-26
WRITE_LINE function, 278-18
WRITE_LINE procedure, 265-72
WRITE_RAW function, 278-19
WRITE_RAW procedure, 265-73
WRITE_RAW_DATA procedure, 276-26
WRITE_TEXT function, 278-19
WRITE_TEXT procedure, 265-73
WRITEAPPEND Procedures, 93-72
WRITETOBUFFER procedure, 200-89
WRITETOCLOB procedure, 200-90
WRITETOFILE procedure, 200-91

X

XA_COMMIT Function, 189-6
XA_END Function, 189-7
XA_FORGET Function, 189-8
XA_GETLASTOER Function, 189-8
XA_PREPARE Function, 189-8
XA_RECOVER Function, 189-9
XA_ROLLBACK Function, 189-9
XA_SETTIMEOUT Function, 189-10
XA_START Function, 189-11
XDBEvent Type Subprograms, 199-3
XDBHandler Type Subprograms, 199-4
XDBHandlerList Type Subprograms, 199-4
XDBPath Type Subprograms, 199-5
XDBRepositoryEvent Type Subprograms, 199-3
XDBSCHEMA_PREFIXES Function, 193-9
XDBUriType, 286-12
XDBUriType function, 286-15

XDBUriType subtype, 286-12
CREATEURI function, 286-12
GETBLOB function, 286-13
GETCLOB function, 286-13
GETCONTENTTYPE function, 286-14
GETEXTERNALURL function, 286-14
GETURL function, 286-14
GETXML function, 286-14
methods, 286-12
XDBUriTYPE function, 286-15

XDBZ Constants, 198-1
XFORM_BIN_CAT procedure, 44-83
XFORM_BIN_NUM procedure, 44-85
XFORM_CLI procedure, 44-88

XFORM_COL_REM procedure, 44-89
XFORM_EXPR_NUM procedure, 44-91
XFORM_EXPR_STR procedure, 44-93
XFORM_MISS_CAT procedure, 44-95
XFORM_MISS_NUM procedure, 44-98
XFORM_NORM_LIN procedure, 44-99
XFORM_STACK procedure, 44-102
XML_TO_LCR Function, 104-27
XMLType

CREATENONSCHEMABASEDXML function, 300-2
CREATESCHEMABASEDXML function, 300-2
CREATEXML function, 300-3
EXISTSNODE function, 300-5
EXTRACT function, 300-5
GETBLOBVAL function, 300-6
GETCLOBVAL function, 300-6
GETNUMBERVAL function, 300-7
GETROOTELEMENT function, 300-7
GETSCHEMAURL function, 300-7
GETSTRINGVAL function, 300-7
ISFRAGMENT function, 300-8
ISSCHEMABASED function, 300-8
ISSCHEMAVALID function, 300-8
ISSCHEMAVALIDATED function, 300-9
NAMESPACE function, 300-7
SCHEMAVALIDATE procedure, 300-9
SETSCHEMAVALIDATED procedure, 300-9
TOBJECT procedure, 300-10
TRANSFORM function, 300-10
XMLTYPE function, 300-11

XPATH2TABCOLMAPPING Function, 208-11

XRange function, 273-26
XSD_ATTRIBUTE Function, 193-10
XSD_COMPLEX_TYPE Function, 193-10
XSD_ELEMENT Function, 193-10
XSD_GROUP Function, 193-10
XSTORE_CLEAR, 173-50
XSTORE_EXISTS function, 173-50
XSTORE_GET procedure, 173-50
XSTORE_REMOVE procedure, 173-54
XSTORE_SET procedure, 173-54

XStream

DBMS_XSTREAM_ADM package, 212-1
privileges, 213-1
removing configuration, 212-86
security, 212-2
XStream administrator, 213-1